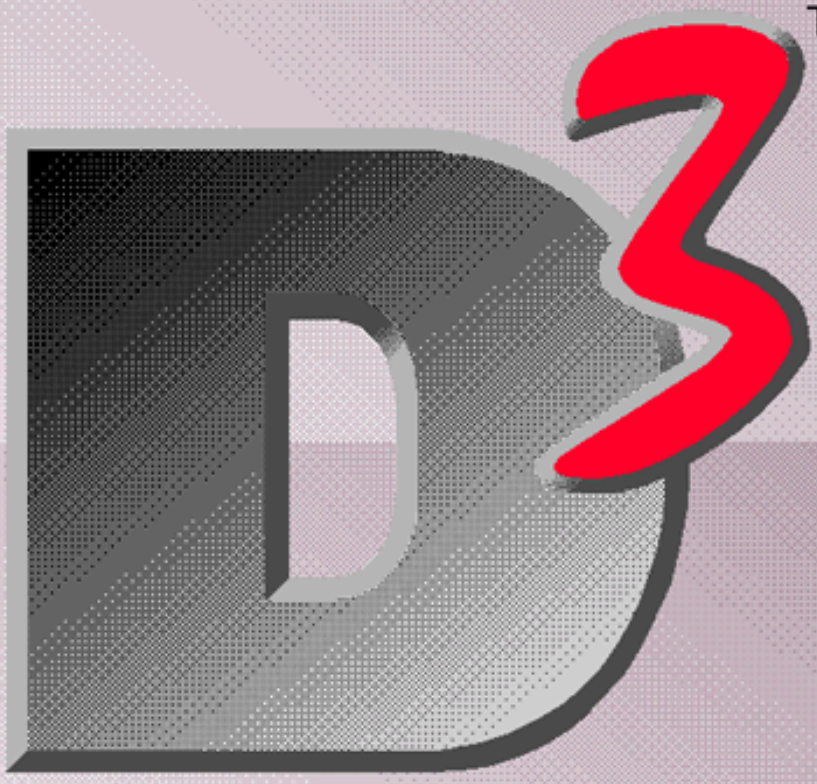
Pick Systems Reference Manual



D3 Version 7.1.0

Doc Version 07/22/97

Table of Contents

About This Book

Overview

The Pick Systems Reference Manual contains entries specific to the features of D3.

The manual is divided into major sections such as AQL, FlashBASIC, C Functions, TCL, etc. Each section is comprised of up to three sub-sections: an introductory explanation, a definition of terms and discussion of related concepts, and an alphabetical listing of applicable entries.

As appropriate, each entry contains the entry name, a description, the syntax, options and examples. Note that references to certain commands may appear in more than one section.

The index alphabetically lists all tokens in the manual.

Ordering

Direct orders for additional manuals to:

Pick Systems
1691 Browning
Irvine, CA 92606

Attention: Order Entry
714/261-7425
714/250-8187 (FAX)

Document Conventions

Different typefaces and type styles are used throughout this book to indicate specific kinds of information.

Typefaces

Words “in this typeface” are the normal text of this manual used for explanation and description.

Words “`in this typeface`” are used to indicate specific D3 words, and may also be used to indicate a dialog between the computer and the user.

Type styles

root	Words or characters in **boldface** are to be entered (typed) as shown. These boldface words are commands, file names, options and other keywords recognized by the system.
speed	Words in *italic* are variables to be replaced by the value. It might be an actual name, word, or number. For example, *baud_rate* might be replaced by **9600**.
n{-m}	The letters *n* and *m* represent numbers. Numbers are usually input as a single string without commas. Numbers may also be used to define a range designation. The starting and ending numbers are typed with a hyphen between. For example, *n{-m}* might be replaced by **10** or by **1-31**.
{*size*}	Information or commands displayed within braces ({}) is optional.
on|**off**	The vertical bar or pipe sign (|) separates available choices
[**0**|*line*|**?**]	Available choices are displayed within brackets ([ ]).

Entering Data

Enter	Input the specified commands or text as shown in the instruction and then press the carriage return key, usually labeled as <Enter>, <Return>, or <New Line> on the keyboard.
Type	Input the specified commands or text as shown in the instruction. Do not press the carriage return key unless instructed.
Press	Press the single specified character or combination of characters as instructed.

Terms

The following terms are used throughout:

login	This term refers to the Unix login. A user logs in by giving a Unix user name and an optional password.
connect	A Unix user establishes a connection with a virtual machine. Normally, this displays the D3 logon message. As noted above, the login and connect steps can be made automatic so a D3 user does not need a Unix login name and password. If several virtual machines exist on the same system, the user identifies the virtual machine using a symbolic name such as **ap_newyork_branch**. This is a *logical* connection, and has nothing to do with the serial port. There is *no* process-to-process communication or data movement of any kind.
logon	This term refers to the D3 logon. A user logs on by giving a D3 user name and an optional master dictionary and password. On a turnkey system, this is the only interface the end-user sees.
disconnect	A D3 user logged on to the D3 virtual machine can disconnect temporarily, dropping back to the normal Unix shell. The user is still logged on to the D3 virtual machine. The user's D3 environment is suspended until the user reconnects to the virtual machine and resumes activity.

Introduction

Data Management

This section presents an overview of the design philosophy behind D3.

D3 is specifically designed for data management. All data in D3 is stored as items within files. Items within these files are divided into sets called attributes which contain one, multiple, or no values.

Subsetting the D3 model into the tabulating model (punch cards, batch processing); items would be called records, attributes would be called fields, and item-ids would be called record keys. Items, attributes, and values can be of variable length. Items and attributes can contain multiple values or sets of values and are separated by specific delimiters. Items are delimited by segment marks (hex FF). Attributes are delimited by attribute marks (hex FE). Multiple values are delimited by value marks (hex FD).

Because delimiters are used, the length of values within attributes doesn't have to be more than the actual physical length of the data. This gives D3 a distinct advantage over traditional systems in that items are of variable rather than fixed length. This not only improves efficiency since disk access time is reduced because of the smaller items, but the amount of disk space required to support the data base is substantially reduced.

Traditional Terminology	D3 Terminology
Disk	Physical Level
Partition	Partition
Cluster	Frame
Block	Block
File System	
Directory	Dictionary
File	File
Record (Primary key)	Item (Item-id)
Field (Alternate key)	Attribute (Index-key)
Values	Values

Items that are stored in a file may be accessed in several ways, including:

- Directly, using the item-id as the key.
- Sequentially in the hashing sequence.
- Sequentially in a sorted sequence.
- By index key.

The direct file access technique, using the item-id to locate the item within the file, is an efficient method of locating data and lends itself to the on-line nature of the D3 System. The system overhead required to access an item using direct file access is independent of the actual file size. Items may also be indexed by attributes and accessed by index key.

D3 dictionary defining items allow both simple and complex conversions and correlatives to be defined using the a-type processing code, among others. The instructions for the formation of index keys are specified using these same a-processing codes. As many indexes as necessary

may be created for a file. The root fid (frame id) and a processing code for each index are stored as a single value in attribute 8 (correlative) of the file-defining item.

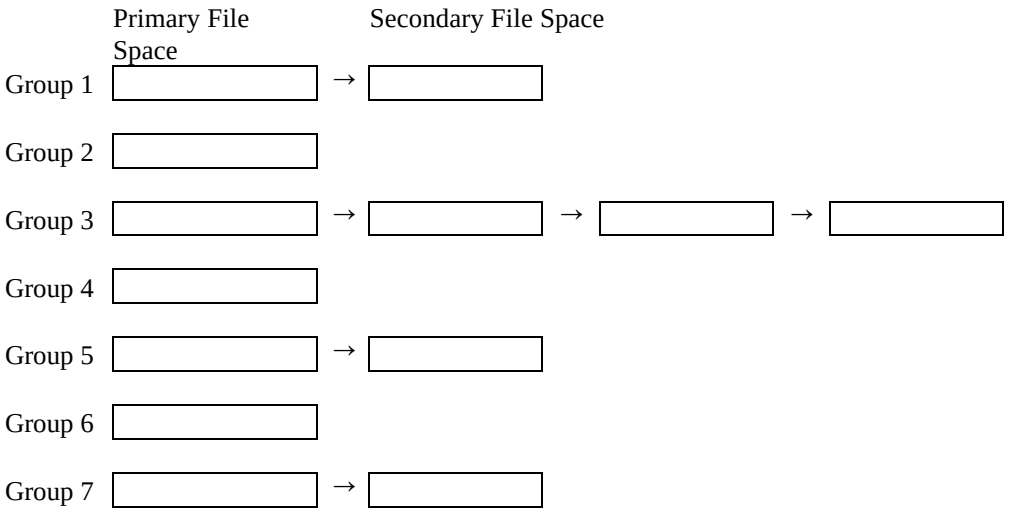
Indexes are created by the TCL verb, create-index, and stored on the disk in ASCII sequence within a B-tree (balanced-tree) structure. Whenever the file is changed, the indexes, if affected, are also changed automatically by the system.

For example, if an item is deleted, all index entries that point to that item are also deleted and changed to point to the next item. Indexes can be used in FlashBASIC programs through the key and root statements. Indexes are also used by verbs such as list, sselect, and sort, if there is an index corresponding to the sort keys.

Files can be updated through the bridge correlative processing codes, by FlashBASIC programs, and through the Update processor (UP).

Files are stored on disk in blocks called frames. The frames are uniform in size (1048 bytes). The primary file space physically consists of a contiguous set of disk frames. The beginning frame is the base frame and the number of contiguous frames or groups (including the base frame) is the modulo of the file. The modulo is defined at the time the file is created or resized. The system automatically adds or removes frames to or from groups as the amount of data (number and/or size of items) within the group expands or contracts. The frames added automatically by the system are added to what is called the secondary file space. This means you do not need to redimension a file as the amount of data in that file increases or decreases.

Consider this example file which was initially allocated seven contigious frames (modulo 7) in Primary Space. As items were added to the file, they were allocated to groups through the hashing algorithm. Some groups required additional frames. These frames were allocated from Secondary File Space. Frames within a group were explicitly linked together.



Items are distributed to the various groups within a file based on a hashing algorithm that calculates the frame identification number (fid) of the first frame in the group. Items are distributed quasi-randomly between groups and sequentially within a group. This quasi-randomness is achieved by using the item-id directly in the hashing algorithm. Because of the

nature of the mathematical relationship defining the hashing algorithm, modulo numbers that are multiples of 2 or 5 should not be assigned.

To enable data transfer to and from disk to occur at optimum efficiency, remember to set the modulo of the file to the nearest prime number above that required to set the number of frames per group below unity (50 to 75 percent utilization traded for single disk access speed, etc.) based on the amount of data storage anticipated. This feature makes the system more efficient in that the probability of two or more users accessing the same group at the same time is reduced because of the algorithm of data distribution between groups.

Number-of-Items X Average-size-of-Items (Bytes) / Frame-Size (Bytes)

The result should be increased to the next largest prime number.

The frame size for a particular release of D3 can be determined by executing the "what" TCL command. The number of available bytes within a frame is listed on the first line of the report under "dfsize". The actual frame size is determined by rounding the "dfsize" up to the next power of 2. The difference between "dfsize" and actual frame size is used to hold the frame linkages (forward and backward pointers).

When more than 50 percent of the groups have more than one frame, or the utilization gets below 50 percent, reallocate the files.

One brute-force method of resizing the file is:

- Creating a new file with the desired modulo.
- Copying all items from the old file into the new file.
- Deleting the old file.
- Renaming the new file to the old file name.

When resizing in this manner, you must explicitly copy the index and other data from the old file's file-defining item to the new file's file-defining item before renaming all index and subroutine calls from the old file.

Files may also be reallocated using the system's save and restore commands. When the save and restore commands are used, the indexes are handled automatically. Prior to saving the system on magnetic media, attribute 13 of the file dictionary file-defining item may be set to the new modulo for that file. When the system is restored, all files will be reallocated according to the new modulo specified in attribute 13. When attribute 13 is not specified, the file is restored exactly as it was saved. The save and restore process allows reallocation of many files at one time.

Note that items within files are saved to tape in group sequence. Because restoring to a new modulo redistributes items within the file space, the number of disk reads is considerably increased during a restore to a new modulo, slowing this process noticeably. F-resize is a program provided with the system to automatically calculate new modulos and mark attribute 13 of the files appropriately using the current statistics in the file-of-files as modified by the last file-save.

Specific TCL verbs (system-level commands) exist to manage files as listed below. Refer to the section "Terminal Control Language" for an overview of the terminal control language. Refer to the separate entries for each verb in the body of the Pick Systems Reference Manual for more detailed information.

clear-file	create-file	rename-file
copy	delete-file	steal-file

Dictionaries

Dictionary items are used by the D3 System to describe, define, locate, and, in general, operate on data within the files to which they point. Many operations are preprogrammed functions that process the data in associated files at the system level instead of at the program level, enhancing overall system performance. Some of these operations include the definition of relationships between and within files.

Relationships between files are expressed using bridge, index, or translate processing codes. Features such as these processing codes provide the ability to cruise and zoom through the data base. Refer to the section "Update processor" for more information about cruising, zooming, and using UP.

Relationships within files are expressed through the structure controlling attribute (attribute 4) of an attribute-defining item.

Each data file in the system has one dictionary. A dictionary may have several files associated with it. Dictionaries associated with data files contain items such as attribute-defining items, file-defining items (for the associated data files) and synonym-defining items (additional views of attributes after different processing), as well as compiled FlashBASIC programs.

Attribute 1, the dictionary code attribute (referred to as the d/code) identifies the item type. If the attribute contains an "a", it is an attribute-defining item. If the attribute contains a "d", it is a file-defining item. If the attribute contains a "q", it is a synonym-defining item.

There are three types of dictionaries: system, master, and file.

System Dictionary - There is only one system dictionary per system. Items within the system dictionary (mds) point to account master dictionaries. The "mds" file can only be accessed on the dm account.

Master Dictionary - There is one master dictionary (md) for each account. When a new account is created, a standard set of vocabulary items are copied into the new account's md. Items within Master Dictionaries point to file dictionaries.

File Dictionaries - There are multiple, distributed file dictionaries among various accounts. Items within file dictionaries point to data files. File-defining items, synonym-defining items and attribute-defining items are also present in file dictionaries. Pointers to compiled FlashBASIC programs are only present in file dictionaries.

Dictionaries as Operators

The D3 dictionary is a file consisting of items that contain 18 attributes. Each dictionary item can be considered as a vector operator of 18 elements, some of which contain operations to be performed on the specified attribute in the associated file. One element identifies the attribute within the associated file that is the operand.

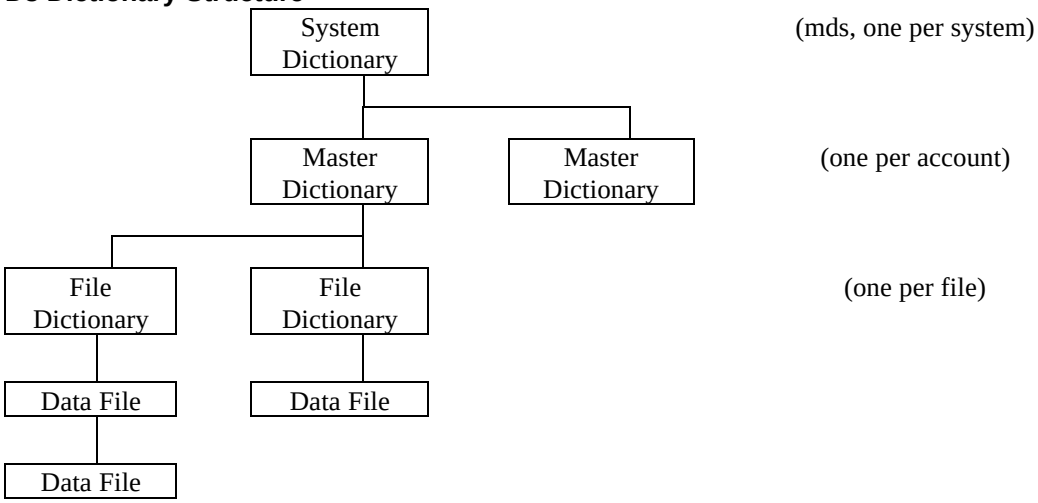
File-defining items operate specifically on attribute 0 of the associated file. File-defining items also contain system information specific to the associated file. Attribute-defining items may operate on any other attribute within the associated data file or dictionary.

Functions (or programs) defined at the system level (or by the user) can be assigned to the appropriate attributes of the attribute-defining item allowing an operation to be defined in

minutes at the system level. In many cases the generation of complex programs in a high level language can be avoided, and this vector provides a shorthand language for generating programs.

Data can be entered through UP or a FlashBASIC program which passes data through the specified attribute-defining item within the data file dictionary for modification and stores the data in the specified attribute of the data file. When the data is viewed using UP or retrieved using Access Query Language (AQL) or FlashBASIC, it passes from the attribute in the data file through the data file dictionary for modification prior to output. Secondary files may be involved if the operation specified in the dictionary item is a translation.

D3 Dictionary Structure



Master Dictionary

There is one master dictionary (md) file for each account. When a new account is created, a standard set of vocabulary items are copied into the new account's master dictionary. The following types of items are contained in master dictionaries:

- Attribute-defining items
- File-defining items
- Synonym-defining items
- Macros
- Menus
- Verbs
- Connectives
- Cataloged FlashBASIC program pointers
- PROCs

File-defining items (d-pointers) and synonym-defining items (q-pointers) are two types of file pointers found in an md. File-defining items point to files within the current account. Synonym-defining items can point to files either within the current account or within other accounts.

There is generally only one file-defining item within a dictionary. Attributes 2 and 3 of file-defining items in an account md contain the base fid and modulo respectively of the file dictionary to which the item points. If the dictionary references more than one data file,

attributes 2 and 3 contain base fid and modulo data respectively for the associated data file. The usage and definition of the attributes in the master dictionary file-defining item are:

Attribute 0	(Item-id) Name of the file being defined.
Attribute 1	Dictionary Code d and other options.
Attribute 2	Base frame number of the associated file.
Attribute 3	Number of contiguous frames in the primary file space of the file.
Attribute 4	Reserved and unavailable.
Attribute 5	Retrieval locks.
Attribute 6	Contains update locks.
Attribute 7	Password required to access the md.
Attribute 8	Reserved and unavailable.
Attribute 9	Attribute "type" or justification.
Attribute 10	Number of character spaces to be allocated for displaying the data within the attribute on AQL reports.
Attribute 11	Reserved and unavailable.
Attribute 12	Reserved and unavailable.
Attribute 13	Reallocation, used in the save and restore process to redefine the value of the modulo of the associated file.
Attribute 17	Description of the file.

To change the information in the "mds" file, use the Update processor. Items can be added, deleted, or modified by executing one of the following command sequences from the dm account:

u mds account.name

create-account account.name

account-maint account.name

UP displays the contents of attributes 1 through 10 (with default values in place for new accounts). To add to the contents of attribute 13, move the cursor to the end of attribute 10, press <return> three times and add the new modulo surrounded by parentheses to resize the master dictionary in attribute 13.

Attribute-Defining Items

Attribute-defining items (ADIs) define views of attributes and processing to be applied to them when used in Update, AQL, and BASIC. For a complete explanation, see the entry "attribute-defining items".

File-Defining Items

File-defining items (fdi) define files, default views of them, indexes to be maintained for the files, audit trail information, as well as some of the processing that automatically takes place when items in files are updated. For a complete explanation, see the entry "file-defining items".

Synonym-Defining Items

Synonym-defining items, also known as q-pointers, are used in account master dictionaries to point to other files. The files may be dictionaries or data files within the current or in other

accounts. In general, only the first four attributes are used (attributes 0, 1, 2, and 3). For a complete explanation, see the entry "synonym-defining items".

System Security

The D3 System maintains several levels and forms of security. There are user and master dictionary passwords, system privilege levels, file retrieval and update lock codes, and restricted access to the system level. For a complete explanation, see the entry "security".

Pick User Groups

There are more than 60 Pick user groups throughout the United States and the world. Contact the group nearest you for information on how to be a part of the activities:

Alaska Pick Users	AK	907/562-3170
Pick Users Group of Arizona	AZ	602/225-9090
Southern Arizona Pick Users Group	AZ	602/326-0250
Fifth Thursday Pick Technical Workshop	CA	714/494-1092
Los Angeles Pick Users	CA	818/906-2952
National Association of Pick	CA	619/594-7590
Northern California Pick Users	CA	510/525-7425
Orange County Association Of Pick	CA	714/972-1727
San Diego Pick Users Group (SDPUG)	CA	619/697-3112
Sierra Pick Information Exchange	CA	916/939-4065
Colorado Pick Users	CO	303/329-9025
Connecticut Pick Professionals	CT	203/658-1900
Pick Users Of Florida Inc	FL	305/254-9611
Atlanta Area Pick Users Group	GA	404/934-5138
Eastern Idaho Pick Users Group	ID	208/525-7800
Treasure Valley Pick Users Group	ID	208/939-8009
Chicago Pick Users	IL	708/963-6510
MICRU International	IL	708/381-9190
Iowa Pick Users Group	IA	319/396-3789
Baltimore Pick Information Connection	MD	410/528-9650
West Michigan Pick Users	MI	616/940-9531
Minnesota Pick Users	MN	612/333-6431
Twin Cities Area Pick Users	MN	612/851-7020
Kansas City Pick Users Group	MO	913/894-9090
Pick Mailers User Group	MO	212/268-3400
St Louis Area Pick Users	MO	314/382-1525
Northern New England Pick Users	NH	207/439-2700
Association Of Pick Professionals	NJ	908/781-0808
New Mexico Pick Users Group	NM	505/345-1450
Long Island Pick Society	NY	516/755-2250
Upstate New York Pick Users	NY	315/488-3681
Pick Users Of North Carolina	NC	919/556-6664
Central Ohio Pick Users	OH	614/836-5790
Pick Users Of Northeastern Ohio	OH	216/781-3700
Southwest Ohio/Tri-State Pick Users	OH	800/543-5033

Oklahoma Pick Users	OK	918/835-6911
Oregon Pick Users Group	OR	503/645-0675
Central Pennsylvania Pick Users Group	PA	717/938-6967
PickSBURGH: Pittsburgh Pick User Group	PA	412/266-7000
The Communication Link	PA	215/828-2560
Mid-south Pick Users	TN	901/722-8571
Pick Users Of North Texas	TX	214/692-8484
Pick Users Of Texas	TX	713/660-9910
Southwest Association of Pick Users	TX	915/594-1444
Utah Blue Chips PC User Group	UT	801/484-4481
Utah Pick Users Group	UT	801/355-3885
National Capital Area Pick User Group	VA	703/273-9675
Richmond Area Pick User Group	VA	804/358-1714
Seattle Area Pick Unix Group	WA	206/277-8666
Seattle Area Pick User Group	WA	206/277-8666
Wisconsin Pick Users Group	WI	414/543-7979
Grupo Usarios Pick Argentina Argentina	[54]	127-4863
Argentine Pick User Group Argentina	[54]	854-0045
IPUA - Sydney Chapter Australia	[61]	25-97-1416
IPUA - International Pick Users Australia	[61]	38-23-2266
IPUA - Melbourne Chapter Australia	[61]	35-47-0477
Pick Users Of Brazil Brazil	[55]	31-224-7800
Forum Pick Brazil	[55]	11-814-3759
Alberta Pick Users (APU)	Canada	403/468-5871
Pick Users Group Of British Columbia	Canada	604/438-7361
Manitoba Pick Users Group	Canada	204/256-3767
Canadian Pick Users (CPU)	Canada	416/665-1669
USSR Pick Users Association - Moscow CIS	[7]	095-334-8709
Groupe De Promotion Du Systeme Pick France	[33]	86-91-1270
Hong Kong User Group Hong Kong	[852]	8-81-0838
Japanese Pick Database Users Japan	[81]	33-239-3525
New Zealand Pick Users (NZPU) N.Zealand	[64]	93-77-4890
Compul IBIS Portugal	[351]	1-284-5877
Seines Portugal	[351]	1-415-0707
Pick User Group Of South Africa PUGSA S Africa	[27]	11-646-3670
Pick Users Association Spain	[34]	33-21-0266
United Kingdom Pick Users Association UK	[44]	8144-68171

What Is D3?

D3 is an on-line computing environment that allows many users to share a computer and a data base. It is a generalized data base management language and system that provides tools to manage information in a flexible, efficient, and timely manner. The essence of developing an application with D3 is to define the problem in terms of information and relationships between information.

Pick has been used to develop thousands of application packages used by more than two million people since the late 1960's. Some of the reasons for D3's success are:

- Applications can be developed faster and easier with D3 than with traditional systems.
- Applications written in D3 run faster than those written in traditional languages.
- D3 uses fewer resources - cpu, disk, and memory - due to its unique data structure and processing techniques. D3 makes micros perform like mainframes.
- D3 makes users more productive. Users conserve time and resources and can operate applications more effectively and efficiently.
- The multibillion dollar D3 marketplace is growing nearly 50 percent per year with more than 4,000 commercial applications available.
- D3 is available on more computers than any other single environment. Usage of the D3 standard guarantees lifetime hardware freedom. Applications written in D3 today will run on the computers of the future.

D3 is a multi-user system. It supports many terminals and runs many jobs simultaneously. Processing originates with commands entered at the keyboard using Terminal Control Language (TCL) and AQL statements, which are a combination of verbs and parameters contained in files called master dictionaries (MDs). Each set of interrelated files (a data base) in D3 is organized around a master dictionary. The master dictionary contains the verbs (both provided with the system and cataloged programs that are part of an application), macros, menus, connectives, and file names needed to process the information in its data base.

The system can be customized to fit the needs of each user. The master dictionaries to which a user has access and the items in those master dictionaries determine the features available to that user. For example, the master dictionary for personnel might include all employee information and can be accessed by one set of users, while the procedures master dictionary might contain procedures on company policy and can be accessed by a different set of users. Furthermore, access to features within a master dictionary may be limited through various security features, such as user privilege levels and update and retrieval lock codes.

D3 retains all the functionality of earlier releases, while adding tools supporting dramatic improvements in the way applications are developed.

In D3, an application is created with minimal source code. This is achieved by utilizing the inherent file structures and many application generation tools provided by the operating system. These tools include the Update and Output processors, processing codes, extended AQL, and FlashBASIC capabilities, as well as macro and menu utilities.

Level pushing provides the ability to interrupt a process and invoke any of these other tools on a new level. The interrupted process is pushed onto a stack and can be resumed at the exact location it was interrupted. Up to 16 levels can be pushed. This enhanced system mobility is a boon to development activities.

The improved B-tree structure allows for indexing and enhanced file searching capabilities. Processing codes, residing in three specific attributes of a dictionary item (the input conversion, correlative, and output conversion attributes), provide the ability to maintain and update files without the need of a FlashBASIC program. These processing codes are used by D3 processors to define operations on data within the associated files at the system rather than program level.

The processing codes include:

- Algebraic functions.
- Assembly code executions.
- FlashBASIC subroutine calls.
- Edit patterns.
- Input and output formats.
- Data conversions.
- If..then..else constructs.

The primary user interface to D3 is the Update Processor. The Update Processor is a dictionary-driven, full-screen editor and data processor. Refer to the "Update Processor" Introductory topic at the front of this book and the entry "Update Processor" in the body of the Reference Manual for more information.

Installing D3

Pick Systems has provided a state-of-the-art multidimensional database management system which is portable to many different hardware configurations, giving the user the most up-to-date system-level capabilities, plus the ability to upgrade hardware without the fear of needing to redevelop software. Refer to the Installation Guide that came with your D3 package for detailed installation instructions.

Definitions

<ctrl>

indicates the control key must be pressed and held down while the following character is entered. Update processor commands are entered with control keys. Commands with more than one character do not need the control key held down on characters after the first.

abs

used to reference absolute frames. These are the frames that do not change, since no data is stored within them. The abs area is where the D^3 virtual code resides.

account

a collection of logically-related files. Each account has a master dictionary, where the vocabulary for the account resides. This includes items such as verbs, PROCs, menus, macros, connectives, modifiers, file-defining items (d-pointers) and synonym-defining items (q-pointers). For example, the dm account contains the files needed for routine system administration.

active list

a list of strings to be used in a subsequent process that handles items one at a time. Generally a list contains item-ids for subsequent processing. For example, the statement **select entity** creates a list of item-ids, while the statement **select entity name** creates a list from the contents of each entity item name attribute.

A **list** may be created by any of the list-generating verbs, such as **get-list**, **select**, **sselect**, etc., or by a FlashBASIC program. The **end** command may be used to deactivate an active list.

Example

sselect entity by name
[404] 18281 items selected out of 18281 items.
>

When this process returns to the TCL prompt, a list is active. The list may be used by any subsequent process that processes items.

```
sselect entity by name
[404] 18281 items selected out of 18281 items.
>save-list entity.by.name
List 'entity.by.name' in file 'pointer-file' saved.
```

This example creates an active list, which is immediately saved. Once a list is saved, it may be used repeatedly, until it is manually deleted.

```
sselect entity by name
[404] 18281 items selected out of 18281 items.
>save-list entity.by.name
List 'entity.by.name' in file 'pointer-file' saved.
get-list entity.by.name
[404] 18281 items selected out of 1 items.
>select entity with city "newport beach"
[404] 287 items selected out of 2718281 items.
```

A list may also be generated from an active list.

adi

acronym for **attribute-defining item**.

amc

attribute mark count.

any key

refers to a key on the keyboard, usually in response to a prompt like "press any key to continue". Every key can be used as a response, except the <BREAK>, and in some circumstances, the <Esc> key, the flow control keys <ctrl>+s and <ctrl>+q.

ASCII codes

American Standard Code for Information Interchange.

The table below lists the first 128 characters of the standard ASCII character set. This includes all printable characters and control characters. **Hex** is the hexidecimal equivalent of the ASCII character, **Dec** is the decimal equivalent of the ASCII character, and **Ch** is the corresponding keyboard character. In the following, the ^ character indicates that the <Ctrl> key is to be held depressed along with the character which immediately follows.

Hex	Dec	Ch	Hex	Dec	Ch	Hex	Dec	Ch	Hex	Dec	Ch
00	00	^@	20	32	sp	40	64	@	60	96	`
01	01	^a	21	33	!	41	65	A	61	97	a
02	02	^b	22	34	"	42	66	B	62	98	b
03	03	^c	23	35	#	43	67	C	63	99	c
04	04	^d	24	36	$	44	68	D	64	100	d
05	05	^e	25	37	%	45	69	E	65	101	e
06	06	^f	26	38	&	46	70	F	66	102	f
07	07	^g	27	39	'	47	71	G	67	103	g
08	08	^h	28	40	(	48	72	H	68	104	h
09	09	^I	29	41	)	49	73	I	69	105	i
0A	10	^j	2A	42	*	4A	74	J	6A	106	j
0B	11	^k	2B	43	+	4B	75	K	6B	107	k
0C	12	^l	2C	44	,	4C	76	L	6C	108	l
0D	13	^m	2D	45	-	4D	77	M	6D	109	m
0E	14	^n	2E	46	.	4E	78	N	6E	110	n
0F	15	^o	2F	47	/	4F	79	O	6F	111	o
10	16	^p	30	48	0	50	80	P	70	112	p
11	17	^q	31	49	1	51	81	Q	71	113	q
12	18	^r	32	50	2	52	82	R	72	114	r
13	19	^s	33	51	3	53	83	S	73	115	s
14	20	^t	34	52	4	54	84	T	74	116	t
15	21	^u	35	53	5	55	85	U	75	117	u
16	22	^v	36	54	6	56	86	V	76	118	v
17	23	^w	37	55	7	57	87	W	77	119	w
18	24	^x	38	56	8	58	88	X	78	120	x
19	25	^y	39	57	9	59	89	Y	79	121	y
1A	26	^z	3A	58	:	5A	90	Z	7A	122	z
1B	27	esc	3B	59	;	5B	91	[	7B	123	{
1C	28		3C	60	<	5C	92	\	7C	124	|
1D	29		3D	61	=	5D	93	]	7D	125	}
1E	30		3E	62	>	5E	94	^	7E	126	~
1F	31		3F	63	?	5F	95	_	7F	127	del

ASCII, definition

an acronym for American Standard Code for Information Interchange.

assembler

changes D^3 assembly language source statements into machine executable object code.

assembly language, D3

defines the language of the D3 virtual environment. It is used to write the D3 core functionalities.

assembly languages

computer programming languages usually dealing directly with the particular cpu's primitives

attribute

a field used to store data within an item.

Attributes represent the highest level data component within an item in a file. An item can contain any number of attributes, each of which can contain any number of values, which in turn can be composed of any number of subvalues.

attribute mark

refers to the specific metacharacter used within D3 items to indicate the end of an attribute, and the beginning of another. The attribute mark is a reserved character and may not be used for any other purpose. The representations for the attribute mark are: ascii value ^, hexadecimal value **fe**, and decimal value **254**.

b-tree

refers to a *balanced tree* access method which D3 supports throughout the system. A b-tree is essentially a system-maintained, ordered-list based on any attribute or combination of attributes from items in a file. It allows quick retrieval of items based on this secondary key.

With minor modifications, current D3 applications can make use of b-trees. Any D3 process which modifies an item automatically maintains the tree. This includes update, FlashBASIC, edit, copy, t-load, and restore.

AQL retrieves keys from a b-tree when an associated attribute-defining item is used in a **by** clause, such as **sort entity by last.name**. Last.name points to an attribute which is also referenced by an **i** (index) processing code in the **adi**. If more than one by clause is used in the same request, AQL ignores the tree and scans the entire file. A **with** clause used without a **by** clause also uses the tree.

Another approach to incorporating b-trees in data retrieval is to use the update processor or the FlashBASIC functions like **key** and **root**.

backward link zero

describes a system error. A frame containing data, either workspace or file system data was not properly linked to a previous frame of data. Instead, this backward pointer was zero. The system was traversing data backward and ran off the beginning of the data.

Each frame of data has both forward and backward links. If the frame has a zero backward link it is usually the first frame detailing some information or document, etc. Any frame with a backward link can be chased through the backward chain of frames to find the first frame (beginning of the data).

From TCL, use **dump** with the **lu** options to travel backward through links. When in the system debugger use the **l** command to see the links of a frame and **<ctrl>+p** to move backward one link.

Syntax

Backward Link Zero; reg = {number} abort @ {abs mode}

Example

```
debug
I ut.go.debug:07E
!l740221<cr>
 see, displayed:
!l740220  740220    0 :  740221     740219 : 0  =
```

At the = sign a <ctrl>+p will display the link fields for the previous frame 740219 and a <ctrl>+n will display the link fields for the next frame 740221.

BASIC

acronym for Beginners All Purpose Symbolic Instruction Code. A third generation programming language originally developed at Dartmouth College in the early sixties.

binary files

accessing binary items using standard D3 utilities. Through OSFI, it is possible to access binary items as if they were D3 items, using AQL, Pick /BASIC, FlashBASIC, etc.

Conventions

Since a standard D3 item is generally accessed in a different manner than a binary item, several conventions are observed:

- The following characters in binary items are converted to the corresponding escape sequences when read through the binary interface. This conversion is reversed when writing an item. Note that this translation scheme is identical to that used by the Unix interface, making it possible to store foreign data inside a D3 binary item exactly as it appeared in the foreign file system.

```
SM        DLE       _
AM        DLE       ^
VM        DLE       ]
SVM       DLE       \
DLE       DLE      DLE
```

- An attempt to read a non-binary item returns the item without translation.
- When writing an item through the binary interface, the resulting item is always made binary. The length of the actual data (after applying the above translations in reverse), is stored in the item header and is transferred by standard utilities including **copy**, **t-dump**, **t-load**, **save** and **restore**.
- Older Pick binary items which do not have binary lengths may return a number of character zeros at the end of the data (zero filled) when that item is read.

Q-Pointer Format

The format of the binary Q-pointer is:

file.name
001 Q
002
003 binary:filename

where **binary** is the name of the binary host in the **dm,hosts,** file, and **filename** is the name of the D3 file. The use of the binary Q-pointer is generally unnecessary for non-D3 files since the

drivers accessing those files already convert binary data into a non-binary, escape format identical to that used by the binary driver. Optionally, the file may be opened directly by adding the **binary:** prefix to the file name.

Example

The following example displays the text of some BASIC object code.

```
:ct dict binary:bp test
001 ............]}.............:. B..B..who._.. u0.P..E._..
    test.]9885.[38516.]7.]bh/]brad.]6.2.0.M0.A2.D2.F5.].^._.
    ........................................(many more dots)
```

The following example copies a binary image from a DOS machine (via an NFS link) into a standard D3 item. Note that the standard D3 item is not a binary item since any binary information representing escape sequences would be processed as escape sequences (whether they are or not) when the data is transferred from DOS to D3. The second command copies this item into a true D3 binary item. The data in this binary item corresponds exactly to the data in the original DOS image.

```
copy /dos PIC001.GIR
(pictures
   1 PIC001.GIR
copy pictures PIC001.GIR
(bin:pictures2
   1 PIC001.GIR
```

blocked IO

describes the mechanism by which a D3 process does blocked IO in order to improve disk performance.

Processes doing sequential disk accesses, like the **file-save**, or AQL statements, have the possibility of grouping several disk reads into one larger disk access. This has the advantage of reducing the number of disk accesses and, therefore, improve performance. For example, selecting a 10,000 frame file normally requires 10,000 disk accesses. If the process can read four frames at a time, it will do only 2,500 disk accesses. Even if reading 4 frames (8 kilobytes on a 2 K frame system) takes a little bit longer than reading only 1 frame (2 kilobytes), reducing the number of disk accesses usually results in a noticeable performance improvement.

The cost of using blocked IO is a greater memory demand on the system, and might have some impact on multiuser performance.

The net effect of grouping the disk reads is reducing the number of 'frame faults' (i.e., when a D3 process requires a page which is not currently in memory). This can be monitored by the TCL command buffers.

The blocking factor (or number of frames read in one disk access) can be set by the TCL command blkio for the whole system, or as an argument in the configuration file.

blz

backward link zero.

branching

a fundamental concept in programming and programming languages dealing with control of program execution.

ccb

see cursor control block and assignfq.

clipboard

refers to a temporary storage area where data can be placed from an application and later retrieved from another application.

See the picknat command for more information.

command

is used to indicate a stored procedure which is inherent to one of the processors in D3.

For example, de, is a valid command in the FlashBASIC Debugger and who is a TCL command.

command and verb are essentially the same thing when used in reference to TCL. For example, who is both a verb as well as a command. Not all verbs are verbs by definition, but they are all commands.

conversion

generally refers to a known set of system subroutines used to convert data to alternate formats. These are listed under the topic of processing codes.

For purposes of all of our documentation, we now refer to the whole category simply as processing codes, rather than trying to distinguish between conversions and correlatives.

correlative

references attribute 8 of the dictionary item being processed by AQL or UP.

There are many processing codes which can be called from correlative on both file-defining items and attribute-defining items.

These processing codes are invoked prior to selection, sorting or displaying and always prior to the processing associated with the dictionary attribute known as output-conversion.

CRT

is an acronym for Cathode Ray Tube, the display portion of almost all dumb terminals.

cruising

ability to scan forwards and backwards through items in a file.

Cruising on a local file using the file index provides the ability to move from item to item in the current file. In the Update processor, "<ctrl>+f" moves "forward" to the next item in the local index and "<ctrl>+d" moves "backward" to the previous item. The terminal "beeps" when reaching the beginning or end of the index.

A "remote" index provides the ability to logically link a "remote" file to the current file with the capabilities of performing the following:

"table lookup"

To perform a "table"-like lookup of items in the remote index using either a full or partial key, enter the key followed by a "<ctrl>+u" to call up the first match. If no match is found, the next item in the remote list is called up.

"table forward"

To move "forward" to the next item in the remote index, enter a "<ctrl>+u". The terminal "beeps" when reaching the end of the remote index.

"table backward"

To move "backward" to the next item in the remote index or file, enter a "<ctrl>+y". The terminal "beeps" when reaching the beginning of the remote index.

"zooming"

To enter or jump into the remote file using the currently displayed reference point, enter a "<ctrl>+g".

"item binding"

To file or store any "link" modifications to the current item, including changes in its "remote" references that might have occurred using one of the above commands, enter "<ctrl>+xf". (See also "bridges", for maintaining automatic two-way referential integrity.)

A classic situation for using remote indexes, for example, would occur while entering an order item; the order file would likely be linked to an inventory file, and the operator could both "lookup" and/or "table" through the data in the inventory file until the desired item is located. If need be the operator could "zoom" over to the item in the inventory file and examine or modify it, and then return to the current order item. When finished with the order, the operator merely files the item and all its line-item references would be maintained with the order.

Note: There is no restriction from the "remote index" being used on the same file that the current item is also residing in. This would be typical in files where an explicit hierarchy exists between items, such as a "parts.master" file which has a bill of materials, an "entity" file which has both companies and employees, or a "projects" file containing sub-tasks, etc.

"Double-clutching" is a term given when a field contains both a local and remote index. An example of this is using the order example mentioned in the previous paragraph. In addition to the "remote" index to the inventory file, a "local" index could also exist to find the next item in the order file that had ordered a particular inventory item.

Syntax

<ctrl>+f

<ctrl>+d

<ctrl>+u

<ctrl>+y

cursor

the underline or block which moves around the screen when characters are typed.

Cursors have various forms depending on the terminal: highlighted, blinking, reverse video, etc. In all cases, the cursor indicates where the next character typed is displayed on the screen. Cursor movement commands are used to position the cursor anywhere in the item without altering the text. The cursor actually moves on the screen of the CRT, but since the screen is only a window into the item, the cursor may be moved off the screen, forcing a screen redisplay or scrolling to the next screen.

cursor control block

a binary item, which contains the codes and control strings pertinent to the terminal in question. Each system functions (@(-(n))) has its own array position, and the system cursor function searches this item for control strings such as clear-screen, clear to end of screen, etc.

cut and paste

is the facility of marking a block of data to be stored in a holding location (cut) in order to copy it into another location (paste). The cut operation can remove the block of data from the item or leave the original data intact.

d-pointer

defines a real file's location.

delimiter

delineates parts of a statement, command or data.

System Delimiters include the four special metacharacters reserved for use by the D^3 Data Model: attribute marks, value marks, subvalue marks and segment marks.

AQL and FlashBASIC both use (') (single-quotes), (") (double-quotes), and (\) (backslashes) as string delimiters, marking the beginning and ending point of a literal string.

Data itself may be delimited by any non-reserved delimiters mentioned above, such as (*) or (-).

dialer

defines a subsystem which allows D^3 systems to communicate over serial lines to transfer items, execute commands on remote sites and to synchronize data bases by transferring updates. This section is a general introduction to the dialer subsystem. See the section dialer, TCL for a detailed description of the TCL command.

Overview :

The dialer subsystem is a set of processes running on each system, which communicate with each other at predetermined times set by the System Administrator, in a batch mode. The main functions are:

- Copying items from system to system(s) (See the section dialer-copy, TCL.

- Submitting commands for remote execution on a system (See the section dialer-submit, TCL.

- Synchronizing a data base across several machines. The data base is replicated on each system, and updates made on the various systems are propagated to the other systems. For example, if the attribute 2 of item A is modified on system M1, and the attribute 3 of the same item A is modified on the system M2, the dialer subsystem will transmit both updates to the machines so that they are applied to all systems.

The various systems on the network are identified on each system by a name and the phone number(s) where they can be reached.

All communication is protected by an authentication process which makes sure the remote system are allowed to call in, to submit commands and to make updates to the data base.

Definitions :

Local System

The local system represents the system where the user is currently connected to.

Remote System

A remote system is a system which is accessible from the local system. Note that all systems must be known to all other systems, for security reason. On large network, the maintenance of the list of systems can be cumbersome, so it might be advised to maintain the list of all systems on one central administration machine which has to be declared manually to all other systems, and use the dialer copy capability to copy the list to all other sites.

Serial Device

The System Administrator designates one or more serial devices to be used by the dialer subsystem. The devices can be dedicated to the dialer subsystem, or they can be shared (not at the same time, of course) with regular terminal activities. For example, a modem line can be used during the day for remote logon, and used at night for data transfer. The serial devices are identified by a unique name. See the section 'Sharing a Serial Device' below.

Serial devices can be designated as input only (they cannot call another system), output only (they refuse incoming calls), or mixed (they can either call or accept incoming calls). Input only channels are useful for configurations where a system may receive calls from many other systems. It reduces the probability that calls will be refused because the system is busy calling other systems. Note that an input only serial CAN transmit data to another system: it does so only after this other system has sent all it had to send.

IMPORTANT: The device MUST be able to support 8 bit characters to be able to transmit the D^3 system delimiters (char(253), char(254), char(255)).

IO Daemon

An IO daemon is a D^3 background phantom process which performs all IO. There is one IO daemon per enabled serial device used for the communication.

Dialer

A dialer is a program responsible for managing a communication device (a modem). It understands the particular protocol required to instruct the modem to dial a number, hangup, etc... A dialer is normally associated to a serial device. Currently, only the 'hayes' dialer is provided. It may be necessary to modify the dialer program to adapt to a special modem type.

Update Posting

For the dialer subsystem to transmit an update to other systems, the system must be made aware of the fact that an update took place. This is done by inserting a CALLX call to dm,bp, dialer.post in the correlative (attribute 8) of the D pointer of the file being updated. The dialer front end menu allows to do this on whole accounts. When an item is filed, the dialer.post subroutine compares the old and the new item and build a 'difference string' which describes what has changed: the original value, the old value and where it has changed. This not only allows a description of the change, but it also reduces the amount of data that has to be transmitted. The changes are determined at the value level. In other words, if the second value of an attribute has been changed, for example, only the second value is transmitted. If a subvalue is changed, the whole value is transmitted. If more than one change is made to an item, the changes are merged, so that only one record of the change is kept, and only one transmission occurs. For example, consider the item 'A' updated as follows:

```
Old      New
---      ---
A: -----> A:
001 a     001 A
002 b     002 b
    bb        bbb
    bbb   003 c
003 c
```

The data transmitted is:

- Change attribute 1 from 'a' to 'A'

- Change value 2 in attribute 2 from 'bb' to 'bbb'

- Delete value 3 in attribute 2

Conflicts

When a system receives an update from another system, the updates are applied to the local item. However, the receiving system checks for conflict by making sure that the 'old value' received from the remote system is identical to the 'old value' on the local system. For example, consider two systems M1 and M2, where the item 'A' is updated:

```
System M1
---------
A: -----> A:
001 a     001 a
002 b     002 b
    bb        BB
    bbb       bbb
003 c     003 c
```

```
System M2
---------
A: -----> A:
001 a     001 a
002 b     002 b
    bb        bb
    bbb       BBB
003 c     003 c
```

In this example, there is no conflict, and the resulting item will have both changes in the 2nd and 3rd values on the second attribute. However, consider the following:

```
System M1
---------
A: -----> A:
001 a     001 a
002 b     002 b
    bb        BB
    bbb       bbb
003 c     003 c
```

```
System M2
---------
A: -----> A:
001 a     001 a
002 b     002 b
    bb        BBB
    bbb       bbb
003 c     003 c
```

In this example, there is a conflict, because the system cannot determine whether 'BB' or 'BBB' is 'right' for the 2nd value of the third attribute. The update will be logged as conflict on BOTH

systems, will NOT be applied to either data base and the System Administrator will have to resolve it by determining which, if any, is correct. A 'copy' operation can then be done to transfer the correct release to the other system. The dialer menu has an option to show the conflicts, at what times (adjusted by the different time zones) they occurred, and what the conflict is.

System Map

The system map defines to which systems an update is to be transmitted. The granularity is the account, but it is possible to define an exception list, to send selected files to a different system. Updates can be sent to more than one system. The system map is created using a dialer menu option. For performance reason, the system map is kept in a named common. Therefore, if the system map is changed, the user processes must be logged off and logged back on.

IMPORTANT: The dialer.post routine must be able to determine the actual account in which the file being updated resides. The account in which the application is run must either own the file (i.e., have a D pointer), or have a valid direct q-pointer to the actual file. Explicit path names (i.e., 'dm,bp,') are not permitted, nor q-pointers to q-pointers.

Billboard

The billboard is a system wide file which contains a record for each posted update not yet being transmitted. This file is checked every time an update is posted to be able to merge the various changes made to an item. On large systems, it might be necessary to resize that file.

Spool

The spool file contains the actual data to be transmitted. The spool items are referenced by other elements of the dialer subsystem.

"Queue"

A special queue file is created for each remote system. This queue contains a linked chain of requests to be sent to a given system. The body of the request is not stored in this file. In the case of a 'copy' operation (copy an entire item), the body of the item is either left in the original file, or is copied into a 'spool' file. In the case of an update item (result of posting an update), the difference string is stored in the spool file.

Sharing a Serial Device :

Most systems will not be able to have one serial port and modem dedicated to the dialer subsystem. More likely, the modem port will be used for remote maintenance, remote logon for most of the day. It is however possible to instruct the dialer subsystem to take over the serial port for a specified period, for example from midnight to 2:00 a.m. every day, to call remote systems, or to accept incoming calls from other systems. This way, the port can be used normally during the day. This is done when declaring the serial devices to the dialer subsystem, using the following convention:

- A dedicated device number is prefixed by a 'S'

- A shared device number is the D^3 process number to which the device is normally linked to.

For example, 'S119' designates a dedicated serial port. '200' represents a shared serial device. The dialer subsystem, when it is time to establish a communication, will steal the device associated to the D^3 process '200' (usually the device '200') by doing an 'unlink-pibdev' to free

the device. If the process was logged on to a user, it is logged off prior to the stealing and the modem is hung up.

Dialer Programs :

A dialer program is a FlashBASIC subroutine which handles the modem specific protocol. These programs must be stored and compiled in the file 'dm,bp,dialers'. They do not need to be cataloged in any account. The module 'hayes' handles the Hayes AT protocol. This module can be used as an example to develop custom dialer programs. The commands the subroutine must be able to handle are:

- DIAL$IDENTIFY

Return an identification string (type and release number). This string is for information purpose only. It should return at least the dialer name and release, and may interrogate the modem itself to get its type and revision number.

- DIAL$RESET

Reset the modem. This command must set the modem in an appropriate state for a dial command to succeed and to accept incoming calls. This command may be sent in the middle of a communication, therefore, it must hangup any on going session.

- DIAL$CALL

Dial out. The argument is the phone number, including any special characters required for the modem and/or the phone system (prefix, pauses, ...). The return code, is positive, is an indication of the communication speed, expressed in bauds. If unknown, the dialer must return 0 for a successful dial out.

- DIAL$HANGUP

Hangup the modem. This command is sent in the middle of a communication to interrupt it.

- DIAL$DEFAULT

Reset the modem to the factory defaults.

dictionaries

are used by the D3 system to describe, define, locate, and, in general, operate on data within the files to which they point.

Many dictionary items perform operations that process the data in associated files at the system level instead of in each program that uses the data. Moving the operation into the data dictionary enhances overall system performance and programmer productivity.

Some of these operations include the definition of relationships between and within files, using bridge, index or translate processing codes. Features such as these processing codes provide the ability to "cruise" and "zoom" through the data base.

Multi-valued relationships within files are expressed through the structure "controlling" attribute (attribute 4) of an attribute-defining item.

Each data file in the system has one dictionary. A dictionary may have several files associated with it. Dictionaries associated with data files contain items such as attribute-defining items, file-defining items and synonym-defining items, as well as pointers to compiled FlashBASIC programs.

Attribute 1, the "dictionary code" attribute (referred to as the "d/code") identifies the item type:

If the attribute contains an "a", "s", or "x", it is an attribute-defining item.

If the first character of the attribute is "d", it is a file-defining item.

If the attribute contains a "q", it is a synonym-defining item.

There are three types of dictionaries:

SYSTEM-- only one per system. Items within the system dictionary (mds) point to account master dictionaries. The "mds" file can only be accessed on the dm account.

MASTER-- only one per account. Items within master dictionaries (md) point to file dictionaries. The following types of items are contained in master dictionaries:

attribute-defining items

file-defining items

synonym-defining items

macros

menus

verbs

connectives

cataloged FlashBASIC program pointers

PROCs

FILE -- multiple, distributed among various accounts. Items within file dictionaries point to data files. File-defining items, synonym-defining items and attribute-defining items are also present in file dictionaries. Pointers to compiled FlashBASIC programs are only present in file dictionaries.

dictionaries as operators

describes using dictionaries as operators.

The D^3 dictionary is a file consisting of items that contain up to 30 attributes. Each dictionary item can be considered as a vector operator of 30 elements, some of which contain operations to be performed on the specified attribute in the associated file. One element identifies the attribute within the associated file is the operand.

There are two types of dictionary entries, file-defining items and attribute-defining items.

File-defining items operate specifically on attribute 0 of the associated file and contain system information specific to the associated file.

Attribute-defining items may operate on any attribute within the associated data file or dictionary. Functions (or programs) defined at the system level (or by the user) can be assigned to the appropriate attributes of the attribute-defining item.

In many cases, the generation of complex programs in a high level language can be avoided, and this vector provides a shorthand language for generating programs.

Data can be entered through UP or a FlashBASIC program which passes data through the specified attribute-defining item within the data file dictionary for modification and stores the data in the specified attribute of the data file.

When the data is viewed using UP or retrieved using AQL or FlashBASIC, it passes from the attribute in the data file through the data file dictionary for modification prior to output. Secondary files may be involved if the operation specified in the dictionary item is a translation.

dirty bits

refer to a status byte associated with each item in the D3 file system indicating the item is new or changed.

These bytes are used to determine which items are backed up during an incremental restore and are reset by a full file-save.

dm

data manager account for the D3 system containing files necessary for basic system operation.

The D3 system makes extensive use of its own data management capabilities to manage system-related data in the "dm" account.

There are three files on the dm account which are used by the system and the users. These files are 'tcl-stack', 'users', and 'pibs' and should be "t-dump"ed prior to performing an operating system upgrade. After a full restore, these files should be "t-load"ed.

double space

describes different methods to double space a document.

To double space AQL reports, use the "dbl-spc" modifier. To double space OP documents, change the vertical motion index by using the ".vmi" OP command.

EBCDIC, definition

an acronym for "Extended Binary Coded Decimal Interchange Code".

element, data

term used to describe an atomic datum variously known as field, attribute, value, attribute value, etc.

entity

represents a system element, like a D3 process, a serial port, a device, with which data can be exchanged.

This section introduces the fundamental notions necessary to understand the principles of data flow within the D3 environment.

Entity ID

Each entity in the system is allocated a unique 32 bit entity id. Some entity IDs are created by the system, and therefore may vary in time, for example a TCP/IP connection gets a new entity ID every time a connection is created, and some are constant, like the D3 process entity ids. At the user level, entity IDs are encoded as a one letter code followed by a null or positive number:

P D3 process or PIB.

S Serial device.

T Telnet terminal emulation.

E TCP/IP connection.

U UDP/IP connection.

Q Message queue.

P11 for D3 process 11, S24 for the serial device 24, E1657 for the TCP/IP connection '1657', are examples of entity IDs. If specifying a D3 process, the 'P' can be omitted. For example, '11' is also the entity ID of D3 process 11.

Standard Input/Standard Output

Data can be sent to an entity by any other entity in the system. An entity can send its output only to ONE other entity.

For example, a D3 port, or PIB, is an entity in the system: its input can be a TCL command or data to an FlashBASIC INPUT statement, and its output can be the result of a TCl command or from a FlashBASIC PRINT statement. The D3 process can receive data from two different terminals, as in tandem, but can send its output only to one terminal. A serial port is also an entity, though the notion of 'input' and 'output' might be less intuitive: its input is the data which must be transmitted to the serial device and its output is the data it receives from the device.

Data can also be exchanged through the attachment of another entity (see below).

Forwarding

An entity input data can optionally be forwarded to an another entity in the system. This mechanism can be viewed as a duplication of the incoming data. For example, in tandem, the input to the serial port associated to the terminal in tandem, is forwarded to the terminal which initiated the tandem connection, thus displaying the D3 process output to both terminals.

Output Link

The output link is the way by which an entity sends its output to another entity. There can be only one output link for each entity, but an entity can be the target, or destination, of any number of output links. If an entity does not have an output link, its data flow is stopped until an output link is created. In the case of a D3 process, for example, the process would be suspended. In the case of a serial device, an XOFF would be sent to stop the incoming data flow once the internal buffers are full.

Forwarding Link

The forwarding link is the way by which an entity forwards (duplicates) its input to another entity. There can be only one forwarding link for each entity, but an entity can be the target, or destination, of any number of forwarding links.

Association

An association is a combination of entities, along with the various links which regulate the data flow among them, that is treated as a whole. For example, in tandem, the D3 process and the two terminals interacting with it constitute an association.

Special Entities

The system creates automatically the following special entities:

BB Byte Bucket. All data sent or forwarded to this entity is discarded. For example, the output link of a D3 phantom process is the byte bucket. The entity ID of the byte bucket is -1.

CO Console. This entity can be reassigned to any entity in the system. By default, data sent or forwarded to this entity is logged into a Unix file (on Unix implementation), or discarded (other implementations). The entity ID of the console is -2.

NULL Null entity. This special entity does not accept any data. In other words, if an entity attempts to sent its output to this entity, it will be suspended. The entity ID of the NULL entity is 0.

Attachment

An entity can attach to another entity to obtain exclusive usage of this entity. For an entity to be attached successfully, it must not be already attached, it must not have an output link, and no other entity can have a forwarding or output link to it. Note an attached entity can have a forwarding link to another entity. Once an entity has attached another entity, it can send data to it and receive data from it, independently from the normal data flow going through the links. In other words, If entity A has its output link to entity B, and attaches to entity C, then it can send data through its standard output to B, and data to C using the attachment. At the application level, data can be exchanged with an attached entity through the FlashBASIC GET and SEND statements.

Notes: Once an entity has been attached, it is permitted to create any link from or to this entity.

Data sent to or read from an entity through the attachment mechanism, is not passed through the input/output translation mechanism (see "set-imap" and "set-iomap"), and data sent to it cannot be forwarded to another entity.

Entity Classes

The system distinguishes the following classes of entities:

Cloning: A cloning entity is an entity which 'clones' or duplicates itself, when another entity attempts to establish a link or to attach to it. For example, inside a given virtual machine, there is only one TCP/IP entity, which has the ID 'E0'. When a D3 process, for example, needs to get a TCP/IP connection, it attaches to E0, which clones itself, allocates a free entity ID, for example E1657. Communications then takes place with this clone. When the D3 process detaches from the clone, the clone is destroyed.

Cloned: A cloned entity is the result of the duplication of a cloning entity. Its lifespan is usually limited and it is destroyed when it is detached.

Special: A special entity is either the NULL entity, the Byte Bucket or the Console.

Normal: A 'normal' entity is defined as not being any of the previous classes. PIB, Serial Ports are example of 'normal' entities.

expression

a variable, literal, function, expression enclosed in parentheses, or a combination of these, separated by operators.

An expression may also be recursively constructed of other expressions and operators.

external format

describes data that has been "externally converted" using one of the "conversion" or "processing" codes provided with D3. As a general rule, "external format" means data is in a "human-readable" format.

The main advantage of an "internal" format is that it takes much less time to compare two values.

For instance, time values are not stored as the human readable form. Rather, they are converted using the "mt" input processing code, which takes a legal "external" time such as "14:00" (2:00 p.m.) and calculates the number of seconds past midnight for this value, which would be the "internal" value. In this case, the "internal" form would be "50400". The same "mt" code used as an output processing code may be used to convert the internal value to its "external" form.

Dates are also more easily manipulated if stored in an "internal" format. A input processing code of "d" will take an external date, use a calculation to determine the number of days that have elapsed since December 31st, 1967 (day 0 (zero) on the D3 calendar), and store an "internal" julian date. A similar output processing code will convert an "internal" date to its human-readable "external" form. See the date processing codes for the various output formats available.

In addition, numerical values are more easily handled by the system if they are all stored as integer amounts, and the decimal point (if any) is inserted only upon output. Typical American dollar amounts, say, "$100.00" is converted upon input to an "internal" form of "10000" using the "MR2" input processing code. A similar output processing code will re-display the figure as "100.00". See the "m" processing codes for all its variations.

One final reason for using an "internal" format is to save storage space. Such an example would be to convert the external form of "SMALL", "MEDIUM", and "LARGE" into "S", "M", and "L". The internal values can then be re-constituted upon output.

fcb

see "file control block".

fdi

acronym for file-defining item.

fid

a contraction for "frame-id"; the unique identifier to each virtual frame on the disk.

Frames begin at frame 1. The last addressable frame is called "maxfid".

file

description of D3 files.

Everything in the D3 data base is an item in a file.

File naming conventions:

Every item-id, filename, and account name abide by the same set of rules concerning valid and invalid characters. D3 provides an enormous amount of flexibility on how things are named.

It is easier to indicate which characters are NOT valid in account name's, filename's and item-id's than to individually explain which characters ARE valid:

The characters that should be avoided include:

space The space is a standard TCL command line delimiter.

quotes ("double quotes" and 'single quotes'). Quotes are used in the retrieval language to indicate literal values and specific item-id's.

backslash ("\") The backslash behaves like quotes.

caret (^) The caret is used in the retrieval language as a "wildcard" character.

Left and right parenthesis ("()")

Parentheses are used to indicate that options follow.

Control characters

They don't print out and can cause all sorts of strange problems.

While the above list is not without exceptions, avoiding the use of these characters can prevent problems later.

Examples of valid item-id's:

S1000, abc-1898.1981, %g$1@*#

file control block

defines a file control block.

Every file on the system has a special frame (or frames) attached to it called a "file control block" or "fcb". These frames contain information about the file, including index pointers and their compiled a (algebraic) processing codes, the file "d-pointer" information for editing, pointers and workspace used by the system when calling FlashBASIC subroutines from file dictionaries.

Also contained in this "file control block" are important flags and fields indicating that the file is open, closed, in the process of being deleted, etc. This provides the ability to protect the system from potentially disastrous situations, such as the case of one process deleting a file while another process is using the same file.

file lock codes

description of retrieval/update locks.

Attributes 5 and 6 of file-defining items can contain retrieval and update lock codes respectively. These codes are used to restrict access to certain data files and master dictionaries. Lock codes are sets of characters used as codes. Multiple lock codes are separated by value marks. The first lock code (retrieval or update) in an md or file dictionary must be matched in attribute 6 (key) of the user's item in the users file to allow access to the file or md. If the lock code does not match, access is denied.

Locks can be placed at any file pointer level, system, master dictionary, or dictionary. The system pointer controls access to a master dictionary; the master dictionary pointer controls a file dictionary, and a file dictionary pointer controls the data file.

Example

```
1)  In this example, a company has four departments; finance, admin, mis, and
ops.  Each department is maintained in a separate master dictionary.  Users
remain attached to a specific master dictionary, but must be prevented from
"q-pointing" or using path names to access files on any other master
dictionary.
The following are the locks as they appear in the system-level
d-pointers and the user definitions:
In the "mds" file:
id: finance.mstr  admin.mstr  mis.mstr  ops.mstr
001 d             d           d         d
002 67889         786554      45000     23007
003 11            27          13        11
004
```

```
005 finance       admin       mis       ops
006 finance       admin       mis       ops
007
008
009 l             l           l         l
010 10            10          10        10
The following are a few of the users and their locks "keys".
user:     toms    glendaj     sama   carlak   gandalf
006(keys) admin   finance     mis    ops      admin]mis]ops
Every user is restricted the those files found on the local master
dictionary except for "gandalf". He can access files on 3 of the four master
dictionaries.
2)  Update and retrieval locks can be set for specific files. For
this example, two files on the "admin" account must be restricted.
The files are called, "payroll" and "reviews".  A  new category of
"supervisor" is added.  "supervisor" files are only accessible by
users with this key regardless of the host master dictionary.
On the "admin" account, these pointers define the dictionaries of the files:
id: payroll       reviews
001 d             d
002 56678         344567
003 23            13
004
005 supervisor    supervisor
006 supervisor    supervisor
007
008
009 l             l
010 10            10
"gandalf" is now a supervisor:
user:      gandalf
006(keys)  admin]mis]ops]supervisor
Now, "gandalf" can retrieve "payroll" and "reviews" but "toms" is
still restricted.
```

file paths

see "file.reference".

file-defining item

describes the mechanism by which the D^3 system establishes the disk address of a file.

This item contains ASCII-numeric data which describes the file location in virtual memory.

These numbers are the "base" frame id and the modulo" which resolves to drive, cylinder, sector and head.

File-defining items are often called "fid" or "d-pointers" in reference to the dictionary code in attribute 1.

File-defining items define the nature of the item-id (attribute 0). File-defining items define the relationships between files. File-defining items contain default macros for AQL and Update.

Information about the file itself, its physical extents, status, indices, etc., is stored in a separate data structure called the "file control block". This data structure is unavailable to the user except indirectly through the file-defining item.

Attribute 0: Item-Id

Attribute 0 contains the name of the file being defined.

Attribute 1: Dictionary Code (d/code)

For file-defining items this attribute must contain a "d". The following characters may be added to the dictionary code to further define the item:

l Logs any updates to the transaction logger.

n Disable update protection on a file even if global update protection is enabled. May not be used with the "u" option.

p Primary filespace contains only pointer items.

s Item-ids are case sensitive.

u Enable update protection on a file even if global update protection is disabled. May not be used with the "n" option.

x Does not save file contents on file save. The file does not exist on restore.

y Does not save file contents on file save. An empty file of the same modulo exists after restore.

Attribute 2: Base

Attribute 2 contains the frame identification number of the associated file. The value assigned to the base of the file is reflected in the message that displays to the screen when the file is created.

Attribute 3: Modulo

Attribute 3 contains the number of contiguous frames in the primary file space of the file. The value assigned to the modulo of the file is reflected in the message that displays to the screen when the file is created. A file can not be created unless there is a large enough block of contiguous space to handle the requested file size.

Attribute 4: Reserved and Unavailable

Attribute 5: Retrieval Lock

In the md, this attribute controls access to dictionaries. In dict this attribute controls access to files. One of the user's lock codes in attribute 5 of the users item in the users file must match the first lock code specified in this attribute in order to access the appropriate dictionary. Multiple lock codes are treated as multiple values and are separated by value marks within this attribute.

Attribute 6: Update Lock

The update lock attribute controls access to account master dictionaries or file dictionaries for update. One of the user's lock codes in attribute 6 of the users item in the users file must match the first lock code specified in this attribute in order to update the appropriate dictionary. Multiple lock codes are treated as multiple values and are separated by value marks within this attribute.

Attribute 7: Reserved and Unavailable

Attribute 8: Correlative (see "correlative").

Attribute 8 contains processing codes described in "Processing Codes". Processing codes specified in this attribute allow a file to function as an application by pre- and post-processing data, and are applied when an item in this file is updated.

Attribute 9: Justification

The attribute type contains codes used to specify justification for the item-id:

l Left justification with word wrap on overflow. This option will break on a word.

r Right justification with overflow to the left.

t Left justification with word wrap on overflow. This option word wraps, only breaking on a space.

u Left justification with overflow to the right.

w Processes through OP before listing. Any valid OP command may follow this code or be used in this attribute. This code only works for non-columnar formats. "Dot" commands following the 'w' are prefixed to the attribute. A second value may contain "dot" commands to be postfixed to the attribute for Output Processing.

ww Processes all attributes from the current attribute through the end of the item through OP before listing. This code works for non-columnar formats only. "Dot" commands following the 'ww' are prefixed to the first attribute. Thus, they are executed before the output data. A second value may contain dot commands to be postfixed to the item for Output Processing.

x Used with "l", "r", or "t" to expand display field to fill the terminal or printer width as specified in the term command.

Attribute 10: Column Width

The column width attribute is used to define the number of character spaces to be allocated for displaying the item-id on AQL reports.

Attribute 11: Reserved and Unavailable

Attribute 12: Reserved and Unavailable

Attribute 13: Reallocation

The reallocation attribute is used in the save and restore process to redefine the value of the modulo of the associated file.

Attribute 14: Input-Conversion (see "input-conversion")

This attribute contains processing codes to be applied to data prior to entry.

Attribute 15: Macro

The "macro" attribute contains the attribute names to be used as default input specifications for the Update processor. The attribute-defining items listed here are used as the default output list by AQL if no "output-macro" is specified, and are automatically listed when any AQL verb that allows attribute-defining items is invoked and no attribute names are specified. Attribute names in the list are separated by spaces. If the specified attribute name does not exist in the dictionary, it is ignored.

The "macro" attribute may be multi-valued. When multi-valued macros are used, the macro to be used by AQL or the Update processor is determined by calling a FlashBASIC program from the "input-conversion" of the "d-pointer" that assigns a value number to access(18). This value number corresponds to the value number of the macro to be used.

To automatically prompt for an item-id, the "id-prompt" modifier or the "i" UP option may be used.

Attribute 16: Output-Macro

If present, this becomes the default "output" macro. Otherwise, attribute 15, if present, is used as the output macro. The purpose of having both the "macro" and the "output-macro" is to use different default attribute-defining items for AQL and UP.

Attribute 17: Description

This attribute is used for comments and descriptions concerning the function of the current file. "?" gets the help message from the item in the Update processor.

Attribute 18: Reserved

Attribute 19: Reserved

Attribute 20: hotkey.all

See "hotkey.all"

Attribute 21: hotkey1

See "hotkey1".

Attribute 22: hotkey2

See "hotkey2".

Attribute 23: hotkey3

See "hotkey3".

Attribute 24: hotkey4

See "hotkey4".

Attribute 25: hotkey5

See "hotkey5".

Attribute 26: hotkey6

See "hotkey6".

Attribute 27: hotkey7

See "hotkey7".

Attribute 28: hotkey8

See "hotkey8".

Attribute 29: hotkey9

See "hotkey9".

Attribute 30: hotkey0

See "hotkey0".

file-locks

see "file lock codes"

file.name

the name of a D3 file.

file.reference

a standardized mechanism for designating a particular file within D3. This explains the forms that a file reference can take.

There are a number of ways to reference files within D3. Any verb that requires or allows a file reference accepts any of the following references:

1) filename

This form of a file reference accesses the data section of a given file (see example 1).

2) dict filename

DICT filename

These forms access the dictionary level of the file (see example 2).

3) dict.filename,data.filename

This accesses a data section in the given dictionary, when the data section's name is different than the dictionary's name. This is the case when a file has multiple data sections (see example 3).

In all of the above cases, it does not matter to D3 if the file is "local", meaning that the "d-pointer" resides in the current md, or "remote", meaning that the file "d-pointer" exists elsewhere, yet this account has a "q-pointer" already present the current md.

D3 File References and File Paths:

4) account.name,,

This accesses another md (account), or any file-defining item found in the "mds" file (see example 4).

Alternately, the "fully-qualified" form is also valid for referencing another md:

mds,md.name

5) account.name,filename,

This accesses the data section of the specified file in the designated account. The account name must exist as a "d-pointer" in the "mds" file and the filename must exist as a "d-pointer" in the given account (see example 5). Note the "," at the end of the reference; the file reference is invalid without it.

The data level reference is not required, but the trailing comma must be included:

list dm,bp,

This is the same as:

list dm,bp,bp

6) dict account.name,filename,

This accesses the dictionary section of the specified file in the designated account. The account name must exist as a "d-pointer" in the "mds" file and the filename must exist as a "d-pointer" in the given account (see example 6). Note the "," at the end of the reference; the file reference is invalid without it.

Specifying Alternate File Names:

File references within certain processes, such as "copy" and "filing" operations, require a leading "(" (left parenthesis) in cases where an alternate file reference is to be designated. (see examples 7 and 8).

A variety of commands allow or require the "(" character, including: "copy" (TCL), "me" ("merge", in line editor), "<ctrl>+cr" (UP), "<ctrl>+cw" (UP), "<ctrl>+zr" (UP), "<ctrl>+zw" (UP), "sreformat" (AQL), "reformat" (AQL), "fi" (editor), and "fs" (editor).

Example

```
list entity
list dict entity
list invoices,archive
```

```
list dm,,
list dm,bp,
list dict dm,entity,
copy entity '100'
to:(customer 120
The "(" (left parenthesis) is vital in this operation. It indicates that the
string immediately following it ("customer", in this case) is to be treated as
an alternate file name. Without the "(" character, "customer" is treated as an
item-id and item-id "100" would subsequently be copied to "customer" in the
data section of the "entity" file.
copy entity '100'
to:(production.account,entity, '120'
As in example 7, the "(" parenthesis character indicates that the
copy is to be directed to a different file.
```

files

stored on disk in blocks called frames.

The frames are uniform in size, and are usually 512, 1024, 2048, or 4096 bytes. The primary file space physically consists of a contiguous set of disk frames. The beginning frame is the base frame and the number of contiguous frames or groups (including the base frame) is the modulo of the file. The modulo is defined at the time the file is created or resized. The system automatically adds or removes frames to or from groups as the amount of data within the group expands or contracts. The frames added automatically by the system are added to what is called the secondary file space. This means the user does not need to redimension a file as the amount of data in that file increases or decreases.

As items are added to the file, some groups require additional frames. These frames are allocated from Secondary File Space.

Frames within a group are explicitly linked together. Items are distributed to the various groups within a file based on a hashing algorithm that calculates the frame identification number (fid) of the first frame in the group. Items are distributed quasi-randomly between groups and sequentially within a group. The quasi-randomness is achieved by using the item-id directly in the hashing algorithm. Because of the nature of the mathematical relationship defining the hashing algorithm, modulo numbers that are multiples of 2 or 5 should not be assigned.

To enable data transfer to and from disk to occur at optimum efficiency, the user needs only to remember to set the modulo of the file to the nearest prime number above that required to set the number of frames per group below unity (50 to 75 percent utilization traded for single disk access speed, etc.) based on the amount of data storage anticipated.

This feature makes the system more efficient, in that the probability of two or more users accessing the same group at the same time is reduced because of the algorithm of data distribution between groups.

Number-of-Items X Average-size-of-Items (Bytes)

Frame-Size (Bytes)

The result should be increased to the next largest prime number. The frame size for a particular release of D^3 can be determined by executing the "what" TCL command. The number of available bytes within a frame is listed on the first line of the report under "dfsize". The difference between "dfsize" and actual frame size is used to hold the frame linkages (forward and backward pointers).

When more than 50 percent of the groups have more than one frame, or the utilization gets below 50 percent, reallocate the files:

One brute-force method of resizing the file is:
1. Creating a new file with the desired modulo.
2. Copying all items from the old file into the new file.
3. Deleting the old file.
4. Renaming the new file to the old file name.

When resizing in this manner, the user must explicitly copy the index and other data from the old file's file-defining item to the new file's file-defining item before renaming all index and subroutine calls from the old file.

Files may also be reallocated using the system's "save" and "restore" processes. When the "save" and "restore" commands are used, the indexes are handled automatically. Prior to saving the system on magnetic media, attribute 13 of the file dictionary file-defining item may be set to the new modulo for that file. When the system is restored, all files will be reallocated according to the new modulo specified in attribute 13. When attribute 13 is not specified, the file is restored exactly as it was saved. The save and restore process allows reallocation of many files at one time.

"f-resize" is a program provided with the system to automatically calculate new modulos and mark attribute 13 of the files appropriately using the current statistics in the file-of-files as modified by the last "file-save".

flusher

is responsible for scheduling and writing all "write-required" ("dirty") frames back to disk.

All write-required buffers are periodically flushed to disk in the normal sequence of events. The "flush" command is provided to ensure data integrity at any given moment.

The flush interval is set by the "set-flush" command. Roughly ten percent of the write-required frames are flushed at each activation.

The 'buffers' command reports the number of frames waiting to be forced flushed in the 'Enqueued Writes' counter.

font

character typeface.

frame

description of a frame.

A frame is the basic storage area of the system. Each frame is addressed by its numeric "fid", beginning with frame 0, and ending at a frame called "maxfid". Maxfid is determined by the actual size of the disk and the size of data frames.

The number of bytes for the linkage and data areas of a frame are implementation specific.

The D^3 Unix implementations use 2048-byte (2K) frames, with the first 48 bytes being reserved for the linkage fields.

The "what" command contains the size of the data area of the frames on your system under the column heading "dfsize". Note that only the data portion shows. For instance, if the "dfsize" on your system is shown as 2000, this indicates a 2K (2048 byte) frame size.

full duplex

simulataneous bidirectional communications. Both sides send and receive with virtually no turnaround time.

On "ASCII" terminals used with the D3 system, "full duplex" is the mode in which characters entered from a keyboard are transmitted to the D3 host but not displayed to the screen. Normally, D3 ports are configured for "full duplex" also called "echo-plex". Every character received is "echoed" to the terminal. Upon a keystroke, the character viewed on the screen has made a round trip from the system and back to the screen.

full restore

is used as the opposite of a "full save" or "file-save". A "full restore" is the process of reloading the entire file system from either a file-save tape or from the original sys-gen media provided with your system.

The full restore is usually performed from the "options" prompt, where the choices are typically: "a" for ABS load, "f" for full restore, or "x" to coldstart.

gfe

abbreviation for Group Format Error.

gfe handler

processes 'group format error' (gfe) encountered in the D3 file system.

The gfe is first logged as an event in the "errors" file, residing in the DM account, after which the user is given the option of either repairing the damage manually, or letting the gfe handler attempt to resolve the problem by fixing the error and logging any additional information about the gfe into "dm,errors,gfe" file.

The system may be configured so that the gfe handler always performs its repairs without requiring user intervention, by placing an "m" in the options field in the users file for each user that will operate in this automatic fix mode. This mode effectively allows deferring any exception processing until someone more knowledgeable is able to look at the data, at the same time allowing the process which encountered the GFE to resolve the structural inconsistency of the file and continue to operate. It is also convenient in support situations where an end user site needs to continue operating at all costs.

Up to 2 frames of raw data in the group, starting at the beginning of first item affected by the GFE, is copied and stored in the 'dm,errors,gfe' file where it can be examined using the dump utility.

For GFEs that are fixed automatically by the system, the system attempts to reconstruct and recover as much of the group or item as possible using whatever information is deemed reliable and available to it and truncates any remaining structures from that point.

group

partition in a file consisting of one or more linked frames used for storing and retrieving items.

When a file is created it is given a "modulo" which specifies how many one-frame groups are initially allocated for this file as the primary space. The group number is calculated by hashing the item-id with the modulo creating an integer "remainder" in the range zero to modulo minus

one. The group number is added to the file's "base" giving the frame number of the group where the item is stored.

The hashed to group is sequentially searched to locate a specific item. By choosing the modulo appropriately items can usually be located in a single disk access regardless of the number of items in the file. Overflow frames are automatically linked to a group to expand its storage capacity dynamically. Also, these frames become released back to the overflow pool as the group contracts in size.

group format error

also called GFE, a state representing structural inconsistency within a certain "group" of a file.

GFE's are exception conditions and primarily caused by the following:

- a non-planned system shutdown: this could entail a system power-outage, a system crash, or when the power to the system is inadvertently turned off (without first having been performed a "shutdown" necessary to guarantee synchronization of the virtual memory.) Normally, this only results in a problem if the hard disk was busy immediately prior to the outage and wasn't able to completely reflect some of the unwritten changes that existed in its memory to the physical disk.

- indiscriminately clearing group locks (see clear-group-locks verb) can open a failure window potentially causing a condition where two processes are simultaneously modifying the same internal structures, each unaware of the other.

- overriding built-in file-protections when using "delete-file" opens a window for problems. Normally the system traps the condition of attempting to delete a file that is actively open by another process, but for compatibility with older releases the user and/or programmer have the option of overriding this "warning" condition.

- incorrect use of the System Debugger. The System Debugger gives the user total control of all the virtual file space and internal control tables used by the system. This is a lot of power, and it is recommended that caution be used by the person examining or changing spaces containing critical structures. See section on the System Debugger for a description of its commands as well as how to disable its use.

When a process encounters a GFE, the D3 system invokes the "GFE handler" to allow the process to attempt to resolve the condition before continuing. Each user can be configured as to whether or not the system should automatically correct the GFE condition or stop and prompt the operator to choose the method for correction. This feature is controlled by the presence of "m" in the options attribute in the "users" file. Subject to change, the current system-wide default is 'to prompt the user' for their intended action.

RELATIONSHIP CONSIDERATIONS.

The D3 file system is comprised of items in files, with each file being divided into one or more groups in order to allow more efficient access to the items. Each group is consists of one or more linked frames of virtual space used to hold the items, or in some cases, pointers to the items. Each item stored in the group also has a small amount of additional information associated with it, called the "item header". The determination of which item goes into which group is based on a "hashing" algorithm that first calculates a checksum using the item's id and then performs a modulo function based on the number of groups that are in a file (called the "file's modulo".) More than one item may hash to the same group, in which case the group becomes a potentially endless chain of items and/or item pointers, with a well allocated file not exceeding one virtual

frame per group. Memory used in D3 for both file storage and workspace is actually mapped into the physical disks in static units called frames (typically sized 1K to 4K, depending on the implementation). Since the main computer doesn't work directly with this physical disk space, it is necessary for it to first be loaded into a temporary high-speed memory for accessing and performing any necessary changes to it before writing these changes back to the disk. The D3 Virtual Memory Manager is responsible for bringing these "physical disk" frames into real memory and then writing those that are modified back to disk. Additionally, ownership locks are asserted throughout the system on the items in files, on their groups, etc. to prevent another process from making changes to the above structures when someone else is viewing the same.

In summary, any problem where the system is unable to properly associate or perform any one of the above, can manifest itself as a GFE.

h

is exactly the same as pressing the "backspace" key.

Syntax
<ctrl>+h

half duplex

describes communications in a single direction at a time.

One side sends and the other side receives. When the roles reverse; the sending side now receives and the receiving side now sends.

On "ASCII" terminals used with the D3 system, "half duplex" is the mode in which characters entered from a keyboard are displayed on the terminal screen and simultaneously transmitted to the D3 host.

Normally, D3 ports are configured for "full duplex" also called "echo-plex". Every character received is "echoed" to the terminal. The system expects the terminal to also be in "full duplex". If a terminal is inadvertently set to "half duplex", each character is displayed twice for each keystroke.

hangup

is a condition by which a device, usually a modem or a network, indicates to the application that communication is no longer possible.

Overview :

A user terminal can be connected in one of two ways to a device:

- Direct connection.

- Modem connection.

In the first case, the connection is always established. In the second case, however, the connection can be interrupted, due to a modem failure to maintain the communication with the remote modem. In this case, the application may need to be made aware of this incident.

Terminal emulation over local or wide area networks are an extension of a modem connection. A network can 'hang up' (close) an open communication. This event is presented to the application as a 'hangup', or Data Carrier Detect (DCD) loss.

Hangup Signal Handler :

The application controls the behavior of the system in case of hangups with the TCL "dcd" command: when the dcd protocol is off (dcd-off), the application is not notified of a hangup.; when the protocol is on (dcd-on), the default behavior is to log off the process. On Unix implementations, it is possible to run an application specific FlashBASIC program, or macro to handle the event. This program, usually called a signal handler, is specified by the TCL "trap" command.

While the modem is hung up, all terminal output is discarded, and any input waits for the connection to be re-established, if possible (see the restrictions below, depending on the type of device).

The macro or FlashBASIC program assigned to handle the loss of DCD depends on the type of the device used.

Hangup Signal Handler on Serial Device :

On a serial device, the hangup signal handler can be:

"OFF" The process is logged off. It stays at logon until the carrier comes back.

"EXIT" The process is logged off and disconnected from the D3 virtual machine. If the port was marked as 'respawn', Unix will wait for the carrier to be reestablished before creating the new D3 process. If the D3 process was started manually (i.e., from a shell), then Unix will restart a Shell when the carrier comes back.

Other The user provided signal handler can either do an INPUT, which waits until the carrier is reestablished, or just do a stop, in which case whatever command was running while the carrier was lost, just continues (but the terminal output is discarded), until an input (at TCL or a FlashBASIC INPUT) is required. At this point, the process will wait, still logged on, until the carrier is established. See below an example of application dealing with the possibility of a hangup.

Hangup Signal Handler on a Network :

When a process is running on a pseudo terminal, such as the one provided by cloning drivers (rlogin, telnet, etc...), the hang up signal usually means that, not only the connection has been terminated, but also that the pseudo tty (clone) has been destroyed, and possibly allocated to another session. It is therefore impossible to keep the process running on the same pseudo terminal. The choice for the hangup signal handler is limited to:

"EXIT" The process is logged off and disconnected from the D3 virtual machine. The underlying shell, if it existed, was most likely already killed.

Other The user provided signal handler MUST terminate by doing an TCL 'exit' or 'disc' command (such as 'chain 'exit'). There must be NO input, else this will generate an open error.

Example

```
Hangup Signal Usage example :
A global flag in a named common is used to indicate a hangup
occurred, so that the main application can exit of a menu in case of hangup,
to abort a command for example. A second global flag
'display.menu' is used to tell the signal handler that the user was sitting at
a menu when the modem hung up, and that the signal handler should redisplay
the menu before terminating.
* Global flag
common /SYSTEM/hangup,display.menu
* Tell we did not hang up
hangup=0
```

```
loop
  * Tell we want to redisplay the menu
  display.menu=1
  * Display menu
  call set.menu
  * Get command
  input command
  * Tell we are done with the menu
  display.menu=0
  * Do the desired command
  begin case
    ...
  end case
  * See if hung up during processing
  if hangup then
    * Process some more the hangup
    ...
    * Tell we are done with it
    hangup=0
  end
repeat
Signal Handler :
The signal handler sets the global flag 'hangup' in the named common, waits
for the carrier to come back. If the user was in the menu, the handler
redisplays it after the reconnection.
* Global flag
common /SYSTEM/hangup,display.menu
* Tell we are hungup
hangup=1
* Ignore further hangups
execute "dcd-off"
* Do some application specific cleanup
* (abort current commands, etc...)
...
* IF on a network, the handler MUST exit
* chain "exit"  ;* Network only
* Wait for the connection to come back
* An INPUT with a time out of zero ensures
* that the input will return as soon as the
* connection is re-established, without
* requiring that the user actually types in
* something.
input dummy for 0 then null
* Connection is back. See if we need
* to redisplay a menu
if display.menu then
  * We can show the menu again
  call set.menu
  display.menu=0
end
* End of hangup.
* Note we must capture the result to avoid
* disturbing the screen
execute "dcd-on" capturing dummy
* The hangup handler terminates
* The application continues
stop
```

hashing

the file-access method used by D3 to find items in a file

The characters composing the item-id are combined to produce an internal number which is then divided by the "modulo" of the file to determine which "group" the item resides in. The group is then linearly searched until the item is found.

The hashing is done by default in a case insensitive way, i.e., 'a' and 'A' yield the same hash value. This can be changed on a file by file basis by changing the file-defining item from a type "D" to a type "DS".

header files

accessing header information using normal D3 utilities.

Through the OSFI, it is possible to access information about an item such as update stamps, permissions, ownership, and driver-specific data. The header driver translates this information into a format which looks like a standard D3 item.

Note that access to the header information via this driver is read-only. The only way to modify the header information is through the normal update routines. This is enabled via the "Y" correlative for D3 files. With the "y" correlative, the user, pib, and/or timedate may be stamped by appending a "u", "p", and/or "t" to the "y" character. No other information is currently available for normal items. Header information updates on non-D3 items depend on the behavior of the remote file system to which those items belong.

Any utility which physically moves the data (like "copy") changes the header information.

The "save" and restore utilities save and restore the header information as well.

Raw Attribute Definitions :

When reading items via the header driver, the items are returned as a dynamic array with the following raw attribute definitions:

Description

1 User ID - The D3 user name or the Unix user number in hex of the last user to update this item.

2 Pib - The D3 PIB (in hexadecimal) of the last user to update this item. This field is undefined for non-D3 drivers.

3 Time/date - A hexadecimal representation of the number of seconds elapsed since 12:00 AM December 31, 1967 and the time the item was last updated.

4 Permissions - A hexadecimal number representing the permissions on the item. This currently only applies to non-D3 items.

5 GroupID - This is the Group ID (in hexadecimal). This is currently used only by the Unix driver, but may be used by other drivers in the future.

Other attributes are driver-specific.

Q-Pointer Format :

The format of the header Q-pointer is:

```
file.name
001 Q
002
003 hdr:filename
```

'hdr' is the name of the 'hdr' host in the 'dm,hosts,' file.

'filename' is the name of the target file to examine. This may be a local D3 file (assuming the Y correlative has been added to the D-pointer), or a remote file (Unix or Dos).

The file may also be opened by pre-pending the filename with the string "hdr:".

hot backup

describes a 'hot backup' configuration, where one machine is in a standby mode, ready to take over the load from a failing system.

Introduction

It is often required to have a system configuration where down time due to a hardware or software failure cannot be tolerated, or must be reduced to a very short time. A solution more affordable than fault tolerance is to double all necessary hardware resources and maintain the data base on two normal systems. This document examines the issues involved in this 'hot backup' configuration, its advantages, its limitations and the system administration procedures.

'Hot Backup' Solution Overview

The 'hot backup' configuration involves two systems: one 'master' system, which is the system in operation, and a 'slave' system, which is in stand-by mode. Both machines are connected by a fast TCP/IP connection. Users are normally connected to the main system. The backup system is also booted, and has a copy of the data base on the main system. The two machines do not need to be absolutely identical: the backup machine just needs the necessary resources (disk, memory, connectivity, ...) to support the application(s).

During normal operations, all updates to the data base on the main system are applied to the backup system, over the network.

In case of a failure of the main system, the users are switched to the backup machine, and the application is restarted. The down time is limited to the switch over time (may be just the time for the terminal concentrators to establish an ethernet connection to the other machine), and the data loss limited to the updates not yet transmitted to the backup machine. This loss is usually limited to a few seconds worth of work.

Note that the backup machine is not necessary idle. Other applications can be loaded on the backup machine. Also, since the backup machine has an exact copy of the data base, it can be used for editing reports, doing the file saves, etc...

Advantages

- The cost is less than a traditional fault tolerant solution, when the absolute fault tolerance is not required. The second machine does not need to be as powerful as the main system. A slightly slower machine can be used, as long as it can provide an acceptable level of service should the main system become unavailable.

- The backup system is not necessarily idle. As long as the main data base is not updated on the backup machine, it can be used to edit reports, do the file saves, which alleviates the needs of saving the main system, can be used for developments, etc...

- The machines do not have to be physically close to each other. The machines can be in two different locations, which provides protection against major accidents.

- Since the updates are applied at the logical level, as opposed to mirroring of the data on disks, by a system process which is different from the application process used on the main machine, operating system failures are less likely to create corruptions on the backup system.

- The slave system can be the backup of more than one master system. A slave system with a very large disk capacity can act as a on-line archive system for several applications.

Disadvantages

The system administration and recovery procedures require some manual interventions. The system relies heavily on Unix networking which must be understood by the System Administrator.

Main System Failure Recovery

This section outlines the operation required to recover from a failure of the main system. After the failure occurred, the users have been switched to the backup system and the application restarted. The main machine is repaired and must now be set back to the same level as the backup machine.

While the main machine is down, the data base on the backup machine is naturally evolving. To record all changes on the backup machine, all updates are recorded, using the transaction logger mechanism. If the repair time of the main machine is expected to be short (a few hours), the transaction journal can be left on the disk. If the repair time is expected to be longer, it is probably better and safer to write the transactions on tape.

Assuming the main machine's data base has been completely destroyed, following a multiple disk crash, re-synchronizing the main machine 'simply' involves doing a full save on the backup machine, and restoring it on the main machine, and switching the users back to the main system. The problem is that the file save and restore operation can be very long, taking potentially days. It would obviously be unacceptable to stop the operations during this. Therefore, while the save and restore proceeds, updates to the data base must be logged. The updates can be stored to tape, since multi-tape is supported. After the restore has been completed on the main machine, the transactions which have been accumulated during the save/restore operation, are applied to the main data base. During this transaction log load, it is likely that more updates will be done on the backup machine, resulting in more transaction tapes. Depending on the volume of data, there may be a few iterations of this process: load a transaction log tape on the main system while more transaction tape are being created on the backup machine. Eventually, the system will be almost in sync. The users are then disconnected from the backup machine, the very last transactions written on the last tape, and this tape is loaded on the main system. All operations must stop for this short time. Both systems are now in sync. Users can be reconnected to the main machine, and the transaction log across the network can be restarted from the main machine to the backup, and the system is now operational again.

If there is enough disk space on the backup machine, and if the down time of the main system (including the file save/restore) is expected to be 'small', it is possible to leave all the updates on disk. Re-synchronizing the two machines is then simpler: After the restore, start the hot backup process across the network from the BACKUP machine, which now acts as the 'master', TO the MAIN machine, now acting as the 'slave'. This will transfer all the updates made to the backup machine. When the queue is emptied, the users can be switched back to the main machine. This avoid tape manipulation, but involves a higher risk factor, should a major problem occur on the backup system.

Backup System Failure Recovery

If the backup system fails, a procedure similar to the one described for the main system recovery must be applied. The only difference is that the users are never stopped. Essentially, a full save is taken out of the main machine, restored on the backup machines, then all the updates applied to the backup machines. The only impact on normal operations are a higher system load due to the file save, and, obviously, a higher risk, since there is no backup.

Making sure it works

This configuration is usually applied to very large data bases, and making sure everything works and that no data loss occurs is of utmost importance. Network reliability is obviously critical. The various processes (servers) involved in the communication constantly check on each others, assign numbers to the messages on the network and also make sure the transaction logging mechanism itself is operating normally by periodically writing some test data and making sure the updates are sent over. The System Administrator can control the data bases by periodically running some application report, and make sure the results are identical. All network incidents, as well as unusual circumstances, are reported to a predetermined list of users, so that an incident does not stay unnoticed for a long time. The section "hot-backup, TCL", is describing the major system incidents and suggests some corrective actions.

System Setup :

To set up a 'hot backup' system, the System Administrator must do the following steps. Each operation is detailed in the section "hot-backup, TCL".

- Establish a network between the two systems. This network must support TCP/IP (eg Ethernet, Token Ring, etc...). The System Administrator must set the network names of both systems, even though only the receiver's host name is used. The hot backup connection only requires access to TCP. Other elements like NFS, FTP, etc... are not required.

- Determine a free TCP/IP port number. Use the "netstat -a" Unix command to see what is currently in use. A value like 2000 or 3000 is usually safe.

- Load the D^3 data base on the main machine. This will include setting the application, the user files, etc...

- On the master system determine which files are going to be set as DL, i.e., for which updates will be sent to the backup machines. It is generally not advised to set the system so that all updates to all files be sent to the backup system. This has the side effect of also mirroring system files. It is better to exclude the 'dm' account from the transaction log. Use the "set-dptr" TCL command to do change the attributes of files and/or accounts.

- Do a save of the main machine, and restore it on the backup system. This can also be done using the network, as detailed in a Pick Systems Reference Manual section "network save/restore, General". Else, the save/restore can be done on tape.

- Setup the servers on both systems (see the section "Server Setup" in the Pick Systems Reference Manual documentation "hot-backup, TCL").

- Start the master and slave servers.

hung port

considered "hung" when it refuses to accept keyboard input and is producing no "new" terminal output.

In the majority of cases, a hung port occurs due to an X-OFF signal being sent to the device. It is advisable to try an X-ON before any other remedies.

:reset-async, reset-port, and reset-user can be used to clear some hung port conditions.

The "logoff" command is provided to force a "hung" port to the "logon" prompt. The last resort, of course, is rebooting the system.

i

is exactly the same as pressing the "tab" key. If tabs are defined, the cursor moves forward to the next tab stop.

Note that in text items, no physical "tab stop" is embedded in the item; rather, a string of spaces is embedded.

Syntax
<ctrl>+i

incremental restore

restores all items from the most recent "incremental save" tape.

The incremental restore process involves restoring the "base" full file-save tape, and then restoring the incremental tape.

At the prompt for "(h)igh or (s)tandard density floppy, a second prompt appears requesting the diskette drive letter (A or B). The first prompt also contains an "8)mm" option to designate eight millimeter tape. Entering <cr> chooses the same device already in use.

A menu of all available devices is displayed. A device number is expected.

incremental save

backs up only the items which have been changed since the last "full" save. See the subtopic of "Dirty Bits" within the TCL "save" subject.

installation

specific installation information is provided with your "System Installation Guide" provided with your D3 system. If you do not have one, please contact Pick Systems at 714/261-7425.

internal format

describes data that has been "internally converted" using one of the "conversion" or "processing" codes provided with D3. As a general rule, "internal format" means the way the data is stored inside the system.

The main advantage of an "internal" format is that it takes much less time to compare two values.

For instance, time values are not stored as the human readable form. Rather, they are converted using the "mt" input processing code, which takes a legal "external" time such as "14:00" (2:00 p.m.) and calculates the number of seconds past midnight for this value, which would be the "internal" value. In this case, the "internal" form would be "50400". The same "mt" code used as an output processing code may be used to convert the internal value to its "external" form.

Dates are also more easily manipulated if stored in an "internal" format. A input processing code of "d" will take an external date, use a calculation to determine the number of days that have elapsed since December 31st, 1967 (day 0 (zero) on the D3 calendar), and store an "internal" julian date. A similar output processing code will convert an "internal" date to its human-readable "external" form. See the date processing codes for the various output formats available.

In addition, numerical values are more easily handled by the system if they are all stored as integer amounts, and the decimal point (if any) is inserted only upon output. Typical American dollar amounts, say, "$100.00" is converted upon input to an "internal" form of "10000" using

the "MR2" input processing code. A similar output processing code will re-display the figure as "100.00". See the "m" processing codes for all its variations.

One final reason for using an "internal" format is to save storage space. Such an example would be to convert the external form of "SMALL", "MEDIUM", and "LARGE" into "S", "M", and "L". The internal values can them be re-constituted upon output.

item

the fundamental directly accessible aggregation of data in D3.

An item may be composed of one or more attributes, which may be composed of one or more values, which may in turn be composed of one or more subvalues.

Each item is statistically "hashed" to a group within a file based on the content and length of the item-id, or key to the item.

item-id

unique identifier by which an item can be retrieved from a file.

levels

describes level pushing.

Any command or program can be interrupted during execution by pressing the active "level pushing" key, (usually a <break> or <escape> key). When a command or program is interrupted, the system stops execution and saves all parameters so that execution can be resumed exactly where it was interrupted. When a process is interrupted at the normal system level, the system prompts with two colons. At this point the command or program is said to be "pushed one level".

Up to 16 levels may be "pushed". The number of colons in the prompt indicates the number of levels pushed. (The normal system level is considered level 1).

Pressing <return> at the "higher-level" TCL prompt returns to the previous level and continues execution of the process at that level.

The "end" command terminates the current process and returns control to the previous level.

"levels" are a function of the D3 virtual machine and should not be confused with spawning a child process (as would occur in the Unix engine). The current virtual task is suspended, a new virtual workspace is established, and "tcl" is activated as the current user.

Each new level grabs a minimum of 4 frames (user control blocks). Since workspace is floating, new workspace is allocated as required. When a level wraps up and returns, all workspace associated with that level is returned to overflow.

linked overflow

"linked" or "dynamic" overflow are the additional frames attached to a file.

As files grow and contract, additional frames are added to or removed from the groups in the primary file space. All of these extra frames are called "linked" or "dynamic" overflow and the number of frames is always changing.

locking scheme

discussion of the various type of locking schemes.

D3 supports 3 types of Group locks and 1 type of item lock. D3 locks groups on a process basis. A port executing a program will be locked from accessing items locked by the previous level.

Group Update Lock:

When an Update lock is placed on a group, no other process is allowed into the group for reading or writing.

Group Read-Only-Lock:

When a Read-Only lock is placed on a group, no Update locks are allowed. There may be multiple Read-Only locks set by multiple processes on a group at any one time.

Item Lock:

When an item lock is placed on an item in a file, other items in that group are allowed to be accessed and updated. A Group lock is placed on the group to prevent shifting, the item is locked and the group is released, leaving the item lock in place. Item locks are used by the FlashBASIC "readu", "readvu" and "matreadu" statements and by the Update processor.

Example

```
The following program shows how the FlashBASIC "locked" clause can be used. It
shows how to display the filename, port.number, and user that has the file
locked:
prompt "";item=""
open '','filename' to f.filename else stop
open '','users' to f.users else stop
readu item from f.filename,ID locked
  execute "who ":port capturing var
  user.id = field(var," ",2)
  readv name from f.users,user.id,1 else stop
  print "Filename file is locked by port ":port:" ":name
  loop until port = 0 do
    sleep 5
    readu item from f.filename,ID locked else null
    port=system(0)
  repeat
end else stop 202,ID
release
end
```

locks

see retrieval locks/update locks

logon

the procedure used to gain access to the system.

The logon procedure consists of a user prompt which is usually "D3 - Enter your D3 user id:". To gain access to the system, the user should type in his or her user ID which corresponds to an item in the D3 users file. After entering the ID, the user may be prompted for some combination of user password, master dictionary, and master dictionary password depending on the user macro.

The "D3 - Enter your D3 user id:" prompt can be modified by changing the "logon" item in the "mds,," file.

The "master dictionary:" prompt can be modified by changing the "md" item in the "messages" file.

The logon process has a mechanism designed to keep noisy lines from causing system degradation due to repeated logon prompts. This mechanism locks the user out of the logon process after three failed logon attempts. At this point, the user must type the sequence "<space><Enter><Enter>" to restart the logon prompt.

The logon lockout can be toggled with the "logon-lock" verb. By default, the lockout is disabled. D3.

Example

```
An example of an alternate "master dictionary" prompt is as follows:
hEnter Master Dictionary Please >>+
```

macros

executes one or more TCL commands.

Macros are stored in the master dictionary with the name of the macro as the item-id.

When the macro name is entered at TCL, it may be followed with any number of parameters. These parameters are added to the end of the first TCL command in attribute 2 as additional language elements and then passed for processing.

The macro functionality is provided for a simple TCL procedure language. In general, a complex procedure should be written as a FlashBASIC program.

The first line of a macro must contain the character m (modify mode) or n (non-stop mode). Each subsequent line is considered a TCL command to be executed.

If the "m" code is used in the first attribute of the macro, the TCL command is displayed so that changes can be made to the TCL command. If "n" is used, the macro runs immediately.

Macro "comments" can be added preceded by a blank after the macro code on attribute 1. Comments on following lines can be added by using the REM statement.

Additional values stored in an attribute are used as stacked input to the TCL command in the first value.

Macro's may be created by using the Update processor or the "create-macro" verb. The "create-macro" verb takes the last statement entered at TCL and converts it to a macro.

master dictionary

file which contains verbs, procs, macros, menus, connectives, default attribute-defining items, and file-defining items - both local ("d-pointers") and synonym ("q-pointers").

There is one master dictionary (md) file for each account. When a new account is created, a standard set of vocabulary items are copied into the new account's md. File-defining items ("d-pointers") and synonym-defining items ("q-pointers") are two types of file pointers found in an md. File-defining items point to files within the current account. Synonym-defining items can point to files either within the current account or within other accounts.

Attributes 2 and 3 of file-defining items in an account md contain the base fid and modulo respectively of the file dictionary to which the item points.

The usage and definition of the attributes in the master dictionary file-defining item are:

Attribute 0: item-id. Contains the name of the file being defined.

Attribute 1: Dictionary Code

This attribute must contain a "d". Along with the character d, the following options are also available:

l Log any updates to the system transaction logger.

p Primary filespace contains only pointer items.

s Item-ids are case sensitive.

x Do not save file contents on file save. File does not exist on restore.

y Do not save file contents on file save. An empty file of the same modulo exists after restore.

Attribute 2: Base. Attribute 2 contains the base frame number of the associated file.

Attribute 3: Modulo. Attribute 3 contains the number of contiguous frames in the primary file space of the file. The default is 37.

Attribute 4: Reserved and Unavailable

Attribute 5: Retrieval Lock

In the md, this attribute controls access to dictionaries. Any one of the user's multi-valued lock codes in attribute 5 of the user's item in the users file must match the first lock code specified in this attribute in order to access the appropriate dictionary. Multiple lock codes are separated by value marks within this attribute.

Attribute 6: Update Lock

The update lock attribute controls access to account master dictionaries or file dictionaries for update. One of the user's lock codes in attribute 6 of the user's item in the users file must match the first lock code specified in this attribute in order to update the appropriate dictionary. Multiple lock codes are treated as multiple values and are separated by value marks within this attribute.

Attribute 7: Password(s). Attribute 7 contains the optional account password(s) required to access the master dictionary. The encrypted password(s) is/are displayed. Passwords may be multi-valued.

Attribute 8: Reserved and Unavailable

Attribute 9: Attribute type: Justification

The attribute type contains codes used to specify justification. See "attribute-type" for a description of justification types.

Attribute 10: Column Width

The column width attribute is used to define the number of character spaces to be allocated for displaying the data within the attribute on AQL reports.

Attribute 11: Reserved and Unavailable

Attribute 12: Reserved and Unavailable

Attribute 13: Reallocation. This attribute is used in the save and restore process to redefine the value of the modulo of the associated file.

Attribute 17: Description. This attribute is used for comments and descriptions concerning the function of the file. To change the information in the "mds" file, use UP. Items can be added, deleted, or modified by executing one of the following command sequences from the dm account:

u mds account.name

create-account account.name

account-maint account.name

UP displays the contents of attributes 1 through 10 (with default values in place for new accounts). To add to the contents of attribute 13, move the cursor to the end of attribute 10, press <return> three times and add the new modulo surrounded by parentheses to resize the master dictionary in attribute 13.

maxfid

refers to the "last" addressable frame on the disk. The "what" command outputs the current "maxfid".

Any reference to a fid higher than "maxfid" results in the message, "referencing illegal frame".

menus

provide a selection of processing choices.

Menus are items in the master dictionary (md). The menu processor automatically formats on the screen and prompts for one of the menu options.

A "help" facility displays any text associated with that menu option when a question mark is included with the option number.

The menu is invoked by entering its item-id at TCL, as if it were a TCL verb.

The format of the menu item in the md is:

001 me {comments}

002 title

003 option 1

help 1

statement 1

004 option 2

help 2

statement 2.1

statement 2.2

:

005 option 3 . . . . .

"me"

is the master dictionay code identifying this item as a menu.

"comments"

are optional text used to document the menu. The comments only serve to document the menu and are never output during menu processing. At least one space must separate the comments from the "me".

"title"

is the text displayed when the menu is activated. The text is centered and displayed in reverse video, so leading and trailing spaces are significant.

"option 'n'"

is the description of menu choice 'n' as it is to appear on the menu. The text is displayed with a leading "n) " before it, where 'n' is the option number the user is to choose. The menu processor will automatically insert the appropriate option number.

"help 'n'"

is the text to be displayed when help is requested. To request help, enter the number of the option for which help is desired, followed or preceded by a question mark (?). If the optional help text is missing and is requested, the menu processor scans the TCL statement to determine if it invokes a macro or menu. The first line of a menu is displayed as the "help" information. Otherwise no help text is displayed. The help text is processed through the Output processor before being displayed. Embedded OP commands may be used to format the help text as necessary.

"statement 'n'"

is a list of one or more TCL statements to be executed when this menu option is selected. This can be any statement that can be entered at TCL, including another menu name, or FlashBASIC program. Multiple TCL statements are separated by value marks.

A menu is invoked by entering the menu's name (or item-id) at TCL.

The menu title is displayed (centered and in reverse video) on top, and the menu options are displayed in columns (TERM width permitting) with a preceding choice number. At the bottom, it will display "Enter number of choice, number? for help, <Enter> to exit menu, or verb:".

To select a choice, enter the option number, press <Enter>, and the associated statement(s) will be performed. After the statement(s) are performed, the process will display a message "Hit any key to return to the menu ". Pressing <Enter> re-displays the menu, and it will wait for another choice.

To invoke "help", enter the option number and a "?", and press <Enter>. The help text for this option (if any) will display. If there is no help text for this option, but the statement to be executed is a macro or a menu, the processor will read the item, and scan its first line for comments and display them as the help text (if any). Entering a "?" without any option number will display the "menu comments" as if it were help text.

To exit the menu, press <Enter>.

While in the menu, an AQL statement or TCL command may be entered. The system will push a level and perform the command. After the command is performed, the process will display a message "Hit any key to return to the menu ". Pressing <Enter> re-displays the menu, and it will wait for another choice.

messages, structure of

see messages file.

meta characters

used to reference one of the four reserved delimiters that D3 uses within its item structure.

The four meta characters include:

- Attribute marks
- Value marks
- Subvalue marks
- Segment marks.

modulo

designates the number of groups (measured in frames) in the primary file space, as indicated during the creation or reallocation of the file.

The modulo is always an integer remainder in the range 0 to modulo-1. The formula to determine the frame number of the initial frame of the group where the item is located is: fid=base+(item.id/modulo). This initial frame number is commonly called the "base fid".

The modulo of a file is a critical component of the "hashing" algorithm access method. It should never be altered in any file-defining item, as this can cause such problems as gfe's to appear.

During the full or account-restore processes, files are resized according to the optional modulo found in the "reallocation" attribute (attribute 13) of the file-defining item. After changing to the new modulo, all items must be re-hashed before they are replaced. This can cause restore processes (also t-load and copy) to run considerably slower.

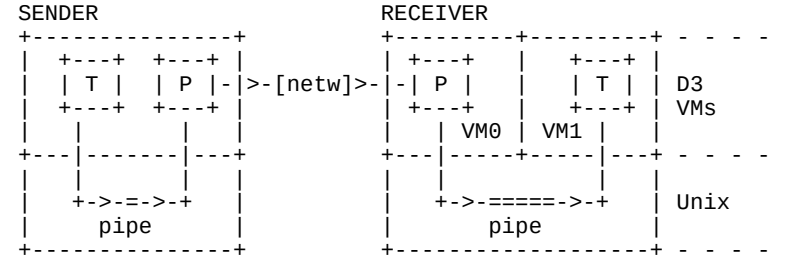
network save/restore

describes how to implement a full save/restore of one machine to another across a network, using "tape-socket". See the documentation "tape socket, General" for general information about his utility, "hot backup, General" for a description of a hot backup utility, and "tape-socket, TCL" for a detailed description of the tape-socket utility.

Overview

The "tape-socket" utility is a very generalized tool which allows to do any kind of tape operations across the network. The hot backup configuration uses this for transaction logging, but this can used to do t-dumps, account save and full saves. In the case of a full save/restore, the problem is a little more complex because the tool requires that D^3 is up and running on the emitting side which is doing the file save (which is no problem), but also on the receiving end, which is a little more difficult since the D^3 data file is being restored. To solve this problem, a small virtual machine is created on the receiver's side, just to act as a host to the tape-socket utility.

This can be represented by the following diagram, where VM0 represents the small virtual machine used only for the transfer, and VM1 the actual virtual machine being restored:



On the sender side, data is saved by the process T into the pipe, picked up by the Output Server P and sent over the network. On the receiver side, data is received by the Input Server P in the small virtual machine VM0, written into the pipe, and picked up by the process P doing the file restore in the real virtual machine VM1.

System Setup

To perform a save/restore across the network, the System Administrator must do the following steps:

- Create both virtual machines on the receiver's side. To do this, the simplest is to install D3 normally, calling the large virtual machine 'pick0', which makes it the default virtual machine for normal usage. Then re-install D3 (using 'installp', 'custom', etc...) and select the option 2 'Display/Change Pick configuration' at the main D3 installation menu. At this menu, select option 1 'virtual machine Name'. Select a new virtual machine name (eg 'vmrest'). CHANGE THE FOLLOWING ELEMENTS: Key (select any number different from the default, which is allocated to the main machine 'pick0'; D3 memory: The 'vmrest' machine needs very little memory. Select for example 8000 K; BASIC memory: This virtual machine does not need FlashBASIC, so enter 0; Disk: Select another disk (16 megabytes is enough); Number of ports: select a small numer (4 or 5 is enough); Number of phantoms: select at least 4.

- Boot and activate the small virtual machine ('ap -0 -n vmrest'). Install the original data set on this machine.

- Create Unix pipes on both systems, using "mknod", and make sure they have appropriate permissions. Put these pipes as 'pseudo-tapes' in the D3 configuration files of both main systems. The small virtual machine 'vmrest' does not need to know the pipe as a tape.

- Establish a network between the two systems. This network must support TCP/IP (eg Ethernet, Token Ring, etc...). The System Administrator must set the network names of both systems, even though only the receiver's host name is used. The hot backup connection only requires access to TCP. Other elements like NFS, FTP, etc... are not required.

- Determine a free TCP/IP port number. Use the "netstat -a" Unix command to see what is currently in use. A value like 2000 or 3000 is usually safe.

- Setup the Servers on both the main system and the small virtual machine 'vmrest' (see the section "Server Setup" in the Pick Systems Reference Manual "tape socket, General"). Make sure that the Servers are NOT linked to the Transaction logger. NOTE: If the system is also used as a hot backup configuration, and this save/restore is used only to synchronize the two systems, it is better NOT to use the default servers for the save/restore operation. Select the 'Other Server" sub-menu operations to set the servers used for the save/restore, and give them some other name, like 'ssave'. Also select different TCP port numbers. Even though they should not be used simultaneously, it is cleaner.

- Start the Output Server on the sender's side and the Input Server in the small virtual machine.

- On the receiver's main machine, start the file restore, selecting the pipe as the 'tape'.

- On the sender's main machine, start the file save, selecting the pipe as the 'tape'. The data is now being restored across the network.

- After the file save is completed, stop the servers on both sides, using the 'shutdown' option in the tape-socket menu from the sender's side. This will terminate the input server on the small machine 'vmrest'.

- The small virtual machine 'vmrest' can then be shut down.

- If the machine just restored is to be used as a 'hot backup' for the sending machine, the servers on both sides, which should now be linked to the Transaction logger, can then be started.

NOTE: If the transfers between both systems are to be used regularly, for T-DUMPs, ACCOUNT-SAVEs, etc..., it is possible to leave the small virtual machine 'vmrest' up, and use it. However, it is probably more efficient to set up an Input Server in the main machine itself.

Open Systems File Interface

provides a standard mechanism for handling non-file entities as if they were standard D3 files.

The high-level interface is through "Super-Q-Pointers" which indicate the host name and some identifier of the data in the remote environment.

The Open Systems File Interface (OSFI) uses the specified host information to access one of many low-level drivers which opens a direct channel into the the remote environment. All standard D3 file system calls are translated into a limited number of internal commands which all low-level drivers understand and can translate into the appropriate operation in their respective environments.

The low-level drivers can be standard ones as provided by Pick Systems, including a D3 remote file driver, a Unix remote file driver, and a D3 spooler driver; or these drivers may be user-written.

osfi

open systems file interface

overflow protection

see scrubber

overflow table

refers to the area that contains a list of unused frames.

The overflow table is handled as a b-tree, rather than as a finite set of overflow table entries.

It is possible to set a "reserve" block of overflow, in case the system runs out of disk (see "set-ovf-reserve").

Most importantly, the overflow table can be rebuilt if it should become necessary (see "rebuild-ovf").

passwords

brief discussion of user and account passwords.

Account passwords may be used to limit the access to an account or to limit access to the system.

"md" (account) passwords are stored in encrypted form in attribute 7 of items within the "mds" file. md passwords are assigned at the time an md is created and can be changed by editing the "mds" file from the dm md at any time by a user with level 2 system privileges (sys2).

Encrypted user passwords are stored in individual users file items in attribute 7.

Both the user and the account passwords may be multi-valued.

Passwords are case-sensitive, regardless of the state of case-sensitivity on the system!

If a password is specified, it is requested at logon. If an incorrect password is entered, the user is denied access to the system. If the proper password is entered, the user is then requested to enter the name of the md to be accessed. If that md requires a password, the user is prompted for it.

The user's password can be changed by editing the users file in the dm account through UP. Passwords may also be assigned or changed by issuing the following command sequence:

u mds account.name password

paths

see "file.reference".

Performance Monitoring

describes various tips, utilities and performance monitoring tools which allow identifying possible bottlenecks in a given configuration.

Introduction

When performance problems are experienced on a system, it is necessary to distinguish problems due to the Unix environment and problems due to a configuration not adapted to the application.

The reader is assumed to have a fairly good understanding of a D3 environment and some knowledge of Unix.

Overview

Unix related performance problems are usually punctual: at one given time, the system performances degrade noticeably, but overall performance should remain satisfactory. These problems are usually fairly easy to track and to fix.

Configuration problems are more insidious, in that they appear repetitively under some circumstances. The basic principle is to monitor the activity of the system over a long period of time during normal system activity. A series of statistics are taken and stored in a log file for later analysis.

The command to monitor the activity is buffers. The command to display the log file is buffers.g.

Unix Related Bottlenecks

The first elements to look at are the results provided by sar to eliminate configuration problems due to an unexpected Unix activity alongside with the D3 activity. Device related problems may also have very visible effects on the overall performance.

SAR Results

See the section 'System Activity Reporting' in the chapter 'System Administration' in the Installation or User's Guide for more details about sar.

CPU usage:

A well balanced system should have a high percentage (above 80-90%) of user cpu usage. High system mode usage indicates too many process switches, or too many system calls. A non null waiting for IO cpu usage indicates disk bottleneck. If the system cpu usage becomes very high, without high IO activity, this may indicate a device problem (see next section).

Paging activity:

The absolute golden rule is to avoid swapping (paging) during normal operations. To avoid swapping, the physical memory must be increased, or the amount of memory allocated to D3 decreased. Surprisingly, if the system swaps, D3 performances may improve by reducing the amount of memory allocated to D3 in the configuration file. Obviously, there are some lower limits which should not be crossed. The D3 activity monitoring should allow determining how far it is possible to go on that path.

If possible, avoid using costly Unix commands during peak hours (compiling is painful, X-window requires a lot of memory, etc...).

If some significant swapping is taking place, control that the memory allocated to D3 (see the verb what) is not bigger than the total amount of physical memory minus the minimum size of memory required for the Unix Kernel (from 2 megabytes for SCO Unix to 6 megabytes for AIX, depending on the Implementation).

To identify which processes are running, do the following (as 'root'):

ps -edalf | grep R

S UID PID PPID STIME TTY TIME CMD

R root 4719 1 ... 07:08:53 24/0 0:05 ap - 24 tty24

R root 8999 10534 ... 07:58:33 89/0 0:00 ps -edalf

S root 10534 4133 ... 08:58:33 89/0 0:00 grep R

R demo 26242 25467 ... 07:10:03 75/0 0:16 demo

The above example shows an extract of the result. This shows that the process 4719 runs D3 on the PIB 24. The process 26242 is a non D3 process which has used three times as much CPU as the D3 process did. By running this command several times, if some processes show several times, it will be possible to identify processes that may be should not be running during peak hours.

Device Problems

The most common problems with TTYs are due to incorrect cabling. When Unix tries to spawn a process (D3 or Unix) attached to a terminal, the device must be ready. If not, Unix 'waits' a bit and tries again. Worse, a port with a DCD in an unstable state can generate many interrupts, which, in turn, generate 'hang up' signals, creating a very important system load. To identify such problem, do the following (as 'root'):

ps -edalf | grep '?'

S root 4184 9047 ... 09:06:26 89/0 0:00 grep ?

S root 25185 1 ... 07:08:52 ? 0:00 ap - 9 tty9

R root 30571 1 ... 07:08:52 ? 23:45 ap - 19 tty19 printer

This command shows the process attached to terminals the system could not open. In the above example,the second line shows a D3 process (pid=25185), in a sleeping state (S): this process does not consume any CPU. The system could not open the terminal /dev/tty9, but the system abandoned tyring to open it. The third line shows a D3 process (pid=30571), in a running state (R): this terminal does use CPU, as the CPU usage '23:45' shows. The system tried to open the device /dev/tty19, failed, as in the first case, but, probably, the cable is incorrect or hanging loose at the other end, and is generating constant signals.

To fix this situation, the terminal must be connected properly or the associated entry in /etc/inittab turned to off instead of respawn. Unfortunately, it is sometimes very difficult to identify which device is in trouble when the above command does not show it explicitly. Only careful checking of the cables or trying to find which ports which did not start as expected, will allow, by elimination, to find the faulty port.

Identifying Configuration Problems

Statistics

The following elements are monitored by the buffers command:

Name Description

Activ Number of Process activations. Each disk read, keystroke, process wake up after a sleep increments this counter. When the number of frame faults is subtracted from this counter, this gives an idea of the volume of data entry.

Idle Idle time. Not supported on Unix Implementations

Fflt Frame faults. This counts the number of disk reads.

Writes Disk Writes. All writes are normally done by the background flush process to update disk from dirty frames in memory. A high number indicates either a lot of updates, but also may be an insufficient memory allocated for the D3 virtual machine.

Bfail Buffer Search Failures. This counters counts the number of failures to allocate a buffer in memory for a new frame. When non zero, this indicates that the memory is insufficient. This counter should never be non zero.

RqFull Disk Read Queue Full. Not supported on Unix Implementations

WqFull Disk Write Queue Full. This counter counts the number of instances where the flusher cannot keep up with the dirtying of frames. This is an indication that either the write queue is too small for the given configuration (see the section 'Flusher Adjustments' later in this appendix) or that the memory is too small.

DskErr Disk Errors.

Elapsd Elapsed time. This is the time in seconds between two sampling. For internal use only.

DblSrc Double Search. This counts the number of collisions between two or more processes frame faulting on the same frame at the same instant. A non zero counter should be exceptional.

Breuse Buffer Re-Use. This counts the number of instances where a memory buffer has been allocated by one process to read one FID and another process allocated the same buffer to contain another FID. A non zero counter should be exceptional.

Bcolls Batch Contentions/Collisions. This counts the number of collisions between a 'batch' process (i.e., a process which is disk intensive) and an 'interactive' process (i.e., a process which is keyboard input intensive). By default, D3 insures that interactive processes are given priority over batch processes in accessing certain resources. See the section 'Batch Processes' in this appendix for more details.

Sem Semaphores Collisions. This counts the number of collisions between two processes trying to access a systemwide internal table.

Vlocks Virtual Locks Failures. This counts the number of cases when a D3 process tried to assert a virtual lock and failed to acquire it because another process had it.

Blocks FlashBASIC Locks Failures. This counts the number of cases when a D3 process tried to assert a FlashBASIC lock and failed to acquire it because another process had it.

B0reg Buffers with no Virtual Registers attached. These are the buffers not currently attached for immediate reference. At any given time, very few buffers are actually attached. It is therefore normal that this number be almost equal to the total buffers in memory.

B1reg Buffers used by more than one process, but not used by its owner any more. These should be in very small number.

B2reg Buffers used exclusively by their owner. On RISC implementations, this situation allows better performance, because there is no conflict on these buffers. Normally, these buffers contain private workspace, data which is not shared, etc...

B>3reg Buffers used both by their owner and other processes. This number represent the number of pages actually shared among processes (data files) at any given time.

ww Write Required. This counts the number of buffers currently modified and not yet written to disk.

IObusy Buffers being read from disk. This counts the number of pending disk reads. This counters is usually null, since the reads are too fast to be picked up.

Mlock Number of buffers memory locked. If the ABS section is locked, this number is at least equal to the ABS size. Also included, are the tape buffers when the tape is attached.

Ref Referenced Buffers. This counts the number of buffers which have been recently used.

WQ Write Queued. Number of buffers currently enqueued for write.

Tophsh Top of Hash. This number measures the quality of the hashing algorithm used to find a frame in memory. This number must be high (above 60% of the total buffers).

avail Available buffers. Number of buffers candidate for replacement. These are the buffers that nobody has been using recently. When this number drops below 10% of the total buffers, performance decreases significantly.

batch Batch Buffers. This is the Number of buffers used by batch processes. A high level (something approaching 50% of disk buffers) indicates that disk intensive activity is taking place by batch processes.

Activity Log File

The activity log is stored in the file buffers.log with a data level per weekday (buffer.log,Monday, buffer.log,Tuesday, etc...). The file is created automatically when the buffers (H) command is used for the first time. Each data level is cleared when changing day, so that the file records a whole week of activity automatically. The itemid is the internal time on five digits.

The buffers command also creates automatically the dictionary attributes corresponding to the various counters, as shown in the table above. The attribute TIME displays the sampling time.

The attribute DESCRIPTION in the D pointers Monday, Tuesday etc... contains the date.

The file is created with a DX attribute.

Monitoring Activity

Logon to the dm account. Type:

buffers {(options}

options

C Clear todays log data level, when used with the (H) option. This option must be used the very first time. To restart the monitoring after having stopped it for a while, do not use the (C) option.

H{n} Record statistics in the log file. If followed by a number n, the process sleeps n seconds between each sample. The default value is 5 seconds. When sampling over long periods, 5 minutes (300 seconds) are a good compromise between accuracy and volume of data.

L{n} Loop sampling and displaying statistics. If followed by a number n, the process sleeps n seconds between each sample. The default value is 5 seconds.

S Display system counters. Without this option, a simplified set of counters is displayed. All counters are always recorded, even without this option.

Examples:

buffers

Take one sample of the non-system statistics.

buffers (sh300c

Loop displaying all counters, recording history and sampling every 300 seconds (5mn). The log file data level corresponding to today is cleared, thus starting a new session.

When looping, buffers polls the keyboard to detect the key "x" to stop or "r" to redraw the screen if it has been disturbed by a message, for instance. Any other key forces buffers to take another sample.

Displaying Log File

Raw display

The history file can be displayed by any access sentence. For example:

sort buffers.log,friday with time >= "11:14:00"

Histograms

The buffers.g command lists the log file as a series of histograms. The syntax is:

buffers.g cntr [day{-{day}}|*] {step {strt.time-{end.time}}} {(option}

cntr Statistic counter name (eg. fflt for the 3rd counter). Must be among the list shown in the table above. If the counter specified is relative to the buffers, percentages of the total buffers are displayed, rather than raw figures.

day Day{s} to list. The day can be one day, expressed either explicitly (monday, tuesday, etc...) or a number from 1 (Sunday) to 7 (saturday). A range of days can be specified by specifying two days separated by a dash (-). If the second day is omitted, Saturday is assumed. The whole week can be listed by using an asterisk (*).

step Specifies the display time step as HH:MM{:SS}. All samples taken within the step are accumulated and averaged. If step is not specified or if the step is 0, or if the step is smaller than the sampling period in the log file, all samples are displayed.

strt.time Starting time. If no starting time is specified, 00:00:00 is assumed.

end.time Ending time. If no ending time is specified, 23:59:59 is assumed.

Options

P Direct output to printer.

Examples:

buffers.g fflt * 01:00:00

List the number of frames faults (disk reads), for the whole week, by step of one hour. In the example below, no history was recorded before Wednesday.

No log for Sunday
No log for Monday

No log for Tuesday

```
20Feb1991; Wednesday; Ctr=fflt, Step=01:00:00, Range=00:00:00-23:59:59
          0   8848  17696  26544  35392  44240  53088  61936
          +------+------+------+------+------+------+------+------+
10:59:28 ************************
11:59:54 ************************************************************
13:00:25 ************************************************************
14:00:52 ***********************************
15:01:18 *************************
16:01:49 ************************************************************
17:02:22 ************************************
18:02:55 ******
19:03:32 *********************************************
20:04:08 *********************************************
21:04:43
22:05:21 ************************************************
23:05:55 *************
Number of samples : 155
Total : 622070
Average per period : 7.1999 / sec.
Max value : 88481
Peak time : 13:00:25
buffers.g ww monday-friday 00:30 08:00-17:30 (p
```

List the percentage of write required write required buffers, for the week days only, during business hours, by steps of 30 minutes.

Interpreting Results

After taking a significant sample, list the results with the buffers.g command . The most useful parameters to survey are:

Fflt This measures the number of frame faults. If this number approaches the disk bandwidth as determined by the manufacturer, the system becomes disk bound. Solutions range from increasing the memory allocated to D^3, to changing disks, or reorganizing the D^3 data base on separate disks to increase parallelism.

Writes This number should stay about one third to a half of the number of frame faults. It is not 'normal' for a system to do more writes than it reads, under normal operation. If this is not the case, see the section 'Flusher Adjustment' in this article.

Bfail This number should never be non zero. If it is not the case, the memory allocated to D^3 is definitely too small.

WqFull This number should not be non-zero 'too often'. If it is the case, and if the number of writes is too big also, there is an abnormal rate of writes. See the section 'Flusher Adjustment' in this article.

Bcolls If this number becomes too high, this indicates that a lot of batch jobs (like selects of big files) are done while other processes are doing data entry. It is also an indicator that indeed interactive jobs are receiving higher priority than batch processes. See the section 'Interactive - Batch Processes' below.

ww This number should never go above 50 % of the whole buffer pool. If this is the case, the flusher is probably not activated often enough. See the section 'Flushed Adjustment' below.

avail This number should never go below 10% of the whole buffer pool. If this is the case, memory must be increased or the flusher must be adjusted.

Flusher Adjustment

The flusher is a background process, started automatically at boot time, which scans the D3 memory and writes back to disk frames which have been modified. It is an important task, not only to ensure that data gets back on disk, but also to make room for new data. Usually, a process reads data, modifies it, but may not need it for a 'long' time. The flusher takes care of writing the data back on disk so that the memory can be reused to read in other data.

This 'cleaning' of the memory is done:

- Periodically, when the disk is not active. If the disk becomes inactive 'for some time', the flusher wakes up and scans the memory writing back all it can unless another a process requires a disk access. This period is defined by the flush statement in the configuration file.

- On demand. When the memory gets 'full', i.e., when a lot of pages in memory have to be written back to disk, the flusher wakes up immediately.

The more often the flusher gets awakened, the more often memory is written back to disk. But this creates disk activity, thus decreasing the disk channel bandwidth available for 'useful' work, and CPU activity, therefore adding system load. Another catch to a high frequency flush is that data which is being modified (workspace, select lists, etc...) may be written several times on disk when only the last time would have been necessary.

The verb set-flush allows changing the flush period (see the section 'TCL commands' in this document. Increase this period, checking with buffers that the 'write queue full' events remains low and that the number of available buffers does not drop too low. Normally, the system is self regulating, increasing the flush frequency in case of high memory usage, so there is no need for a low flush period. 30 seconds should be a high limit.

The configuration file also contains the statement dwqnum which defines the length of the internal write queue. Increasing this queue reduces the probability of the situation in which the flusher awakened on critical demand, thus reducing the number of flushes. The down side to increasing the write queue size is that the flusher works by 'bursts', which may overload the disk channel when this phenomenon occurs. This parameter cannot be changed dynamically, which makes a bit more difficult to monitor.

Interactive - Batch Processes

D3 user processes are divided into two classes, depending on the type of activity they have: interactive processes are processes which typically do keyboard inputs 'frequently'; a batch process is a process which has little keyboard activity, require a lot of disk i/o, and/or is CPU intensive. The system automatically discerns which type of process is running based on internal statistics.

The System Administrator can bias and/or override the default parameters used by the prioritization mechanism. Though not recommended, one can even force any processes, regardless its process activity, to be seen by the system as "interactive", for example. This can be changed dynamically on a per process basis via the set-batch command Also, the TCL command set-batchdly allows the displaying and setting of global values used in the queueing of certain types of process activity.

phantom ports

provides an overview of phantom ports.

D3 provides the capability of executing commands in background on special ports called "phantoms". Phantom ports do not use a serial port.

They can be used to execute background tasks, running in parallel with the user terminals. Jobs are submitted to a phantom port by the TCL command "z". Phantom task activity can be monitored by the TCL command "list-jobs". The activity of the phantom ports is supervised by a special phantom, called the "scheduler", which always run on the last pib of the system.

The number of phantom ports is independent from the number of real ports (i.e., attached to a serial device) or the licensed number of users. The number of phantom ports is defined as follows:

Native Implementation:

The number of phantom ports is fixed (typically 8) and cannot be changed.

Unix Implementations

The number of phantom ports is created at a fixed number, typically 32, but can be changed during the installation procedure to any number. The installation procedure restricts this number to 512, but it can be changed later to a larger number, if required. The TCL command 'config' also allows changing the number of phantoms. The change takes effect the next time the virtual machine is rebooted. The number of phantom ports is described in the configuration file "pib0" by the statement "nphts".

On Unix implementations, the Unix processes running as the D3 phantoms are true daemons, created by the TCL command "start.ss".

pib

an acronym for "process identification block". It is often used, inappropriately, to designate a D3 port number.

pib status

one of the pieces of data returned by the "where" command. The "pibstat" program is used to break the information down into the possible binary states.

Pick Remote Files

a standard D3 file on a non-local D3 machine.

D3 files on remote hardware can be accessed as if they were existent on the local machine. To do this, it is necessary to create a "Super-Q-Pointer" to that remote file. The first attribute of this master dictionary item is a "Q", the second attribute is blank, and the third attribute contains the remote machine's host name, followed by a colon, followed by the COMPLETE D3 path name on the remote machine.

Note that it is first necessary to create an appropriate host item in the "hosts" file to indicate how to communicate with that host.

Once the Super-Q-Pointer has been created, that file can be opened, read, and written just like any local D3 file.

pid

identifies uniquely a Unix process. PID stands for Process ID. In D3, each D3 process is a Unix process is assigned a PID by Unix. D3 processes are usually identified by their D3 port number (from 0 to the maximum number of ports), as usual. It is sometimes necessary to identify the PID of a D3 process. The PID are displayed by the TCL commands "lu", "pid", "psr" and by the Unix command "ap".

pointer item

are internal file "placekeepers", which point to the frame where the actual data resides.

Items are normally stored in a file with a few bytes of overhead data, followed by the item-id, followed by the attributes, and then an end-of-item indicator.

If the size of the item exceeds a given length (860 bytes for a 1-k frame system, 1720 bytes for a 2-k frame system), then D3 uses a "pointer item" in the file, which contains the overhead data, the item-id, and a frame address which points to where the attribute values are stored, so most of the item's data will reside outside of the file's primary filespace.

This method increases performance during most AQL functions, as reading pointer items is considerably more efficient than reading the entire item during such common functions such as LIST or SORT.

A data file may be declared (at create-time) to contain all pointer items, forcing every entry to be a pointer item. There is no converse non-pointer item declaration.

port

used to reference the number of the serial device attached to the hardware. Many commands, such as "logon" and "set-baud", reference a port by its number.

In D3 Unix implementations, "port" is not always the same as the physical device, and can be thought of as the D3 port/number.

port.number

the sequentially-assigned number associated with each physical line or process on the system. The number of ports is determined by issuing the "where" verb with a "z" option.

For documentation purposes, the term "line" is frequently used in place of "port.number". Unless otherwise noted, their meanings are typically the same.

See "phantom" for information on phantom ports.

primary file space

initial space set aside for a file and is specified as a number of frames in the "create-file" command. For instance, if a file data level is set aside using 11 frames as its "modulo", it has 11 frames in its primary file space.

As files grow and contract, additional frames are added to or removed from the groups in the primary file space. All of these extra frames are called "linked" or "dynamic" overflow and the number of frames is always changing.

primary list

refers to an "active list". It is a list of item-id's to be used in a subsequent process that processes them one at a time.

privilege level

see "system privileges".

prompt

a computer request for data entry.

prompt characters

characters that are displayed at input prompts. The number of prompt characters output is equivalent to the level a process is at.

For D3 the TCL prompt is the ":" character. If a list is active then the TCL prompt is changed to ">".

When running FlashBASIC programs, the default input prompt is the "?". Only one input prompt is printed regardless of the process level. The FlashBASIC debugger prompt is the "*" character.

The system debugger prompt is the "!" character.

The Proc default input prompt is the ":". Only one input prompt is printed regardless of the process level.

Example

```
:::  At TCL level 3
>>   At TCL level 2 with an active list
```

q

used in asynchronous communications to resume the flow of data to the device. Its counterpart is <ctrl>+s ("x-off").

Syntax

<ctrl>+q

q-pointer

defines a duplicate (synonym) name for an already-existing file.

QA

see quality assurance

qs-pointer

defines a remote file which is saved as if it were local provided the "e" option is used with the save.

The first attribute of such a pointer must be "QS".

The second attribute must be null.

The third attribute must be a remote host name followed by a colon followed by the remote file name.

Any header information associated with a particular file driver is also saved to guarantee that the data can be restored with the same attributes as it was saved. For example, Unix files are saved with all permissions, ownerships, and update stamps intact.

Example

```
The following qs-pointer causes the file-save to save all accessible
files (not sub-directories) in the Unix /u/john directory.  Note that
the Unix permissions of the D3 user must be sufficient to access
the data on that Unix directory.  See the "Unix Files" item for more
information on specific options.
001 QS
002
003 unix:/u/john
    a
The following qs-pointer causes the remote D3 file "pa,bp," on the
host "prod" to be saved on the local file save.
001 QS
002
003 prod:pa,bp,
```

qs-pointer

defines a remote file which is saved as if it were local.

The first attribute of such a pointer must be "QS". The second attribute must be null. The third attribute must be a remote host name followed by a colon followed by the remote file name.

Any header information associated with a particular file driver is also saved to guarantee that the data can be restored with the same attributes as it was saved. For example, Unix files are saved with all permissions, ownerships, and update stamps intact.

Example

```
The following qs-pointer causes the file-save to save all accessible files
(not including sub-directories) in the Unix /u/john directory.
Note that the Unix permissions of the D3 user must be sufficient to access the
data on that Unix directory.  See the "Unix Files" item for more information
on specific options.
001 QS
002
003 unix:/u/john
    a
The following qs-pointer causes the remote D3 file "pa,bp," on the host "prod"
to be saved on the local file save.
001 QS
002
003 prod:pa,bp,
```

ref

the on-line form of the Pick Systems Reference Manual, provided as an account-save from Pick Systems.

reserved system delimiters

see delimiters

restore

restore backed-up data into a D3 file system.

A restore refers to the loading (restoring) of data into the D3 file system from a backup media created by a "save", or "t-dump"

The "t-load" command can load a single item or a series of items into an existing file from a "t-dump" tape.

The "sel-restore" command can load a single item or a series of items into an existing file from a "save" tape.

The "account-restore" command can create and load an entire account into the file system from a "save" tape.

The "restore-accounts" command can create and load any account on tape which doesn't already exist in the file system from a "save" tape.

The ":files" command can create and rebuild the entire file system from a "save" tape.

Choosing the "(F)iles" option from the "(A)bs, (F)iles (X)ecute" question after booting from the Pick System #1 diskette can create and rebuild the entire file system from a "save" tape.

restricted system access

how to deny users access to TCL.

Users can be denied access to TCL by placing the character "r" in the options attribute (Attribute 9: Attribute type) of their item in the users file. When this restriction is active, users are returned to the system logon program when they exit a process by pressing the <break> key, if it is enabled.

retrieval locks / update locks

overview of restricting access to accounts or files.

If any process, such as a TCL command or a FlashBASIC program is executed from an account that does not have retrieval and/or update privileges for a file, and then attempts to read and/or write to that file, the process terminates with an error message indicating that the file is "access protected".

To read from or write to a retrieval/update protected file, the retrieval and update lock codes must match those of the file that is opened. This is a system function and cannot be accomplished from FlashBASIC. If the file is retrieval protected, the file cannot be opened and the program will terminate.

Example

```
list payroll
At "tcl", the following message occurs if "payroll" is access
protected.
[210] File 'payroll' is access protected.
file payroll
If a file called "payroll" has an access retrieval lock which does
not match the lock codes (keys) for the current user, this statement
aborts the program with the following message:
[210] in program 'enter-payroll', File 'payroll' is access protected.
open 'payroll' to payroll.file else stop 201,'payroll'
If the file is access protected, the "else" clause is taken and the
following message prints:
[201] 'payroll' is not a file name.
```

s

used in asynchronous communications to suspend the flow of data to a device. Its counterpart is <ctrl>+q ("x-on").

Syntax

<ctrl>+s

secondary list

created using the "s" option with the "select" or "sselect" command. Its sole purpose and only real use is during compares and copies.

An example use of a "secondary" list is as follows:

1) Create the primary list.

2) Create the secondary list (using the "s" option).

3) Invoke the "copy" or "compare" command.

4) When the system prompts "to:", enter the filename, (preceded by a left parenthesis). The secondary list takes over and is used for output and comparison ids.

A secondary list may be created to use as the "target" for a list of item-id's in the "source" or "active" list.

The "end" command may be used to "deactivate" an active or secondary list. See "end" for more information on the "double whammy" effect.

Example

```
The following commands compare two files containing FlashBASIC
program source.
sselect bp by mod.date
[404] 75 items selected out of 75 items.
sselect old.bp with mod.date (s
[404] 75 items selected out of 75 items.
compare bp
with:(oldbp
This program copies items in the entity file to a work file using the
"item-id"s kept in the list, "targets".
sselect entity by zip
[404] 1600 items selected out of 1600 items.
gl targets (s
[404] 1600 items selected.
copy entity
to:(work
```

security

provides an overview of D3 security.

D3 provides several levels of security, including:

- File retrieval and update lock codes.

- Separation of user and account, each with their own unique passwords.

- Privilege levels.

- Port logon control. After three unsuccessful logon attempts, the port is disabled to prevent further attempts. To "re-enable" the port, press any key, other than <return>, followed by two <returns>. This is primarily in cases where someone is using a dial-up line and gets line noise, which could be interpreted as invalid logons.

segment mark

refers to the specific metacharacter used within D3 items. The segment mark is not actually part of data. Rather, it is used as an "item delimiter", and is automatically appended to the "end" of an item when written to disk.

The "segment mark" is a "reserved" character. This means that it may be not used for any other purpose.

The following table shows the different representations for the "segment mark":

Ascii value: "_"

Hexadecimal value: ff

Decimal value: 255

selection processor

responsible for presenting items to the LIST Output processor based on processing the selection criteria.

seq

alters the normal sort sequence of characters used by the "ms (mask alter sort)" processing code.

The "seq" item must be manually added to the "messages" file using the following suggested format for the item:

seq (item-id)

001 !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

Software Change Request

whenever requesting a software change, please provide as much of the following information to Pick Systems as you can. This form can be mailed or FAXed to Pick Systems. Software Change Requests can also be phoned to Pick Systems.

Name:

Company:

Address:

Phone #:

Fax #:

Severity: Low Medium

High Critical

Type: Error Report

Enhancement Request

Request for Information

Attachments: Media Documentation

Other:

D3 Release:

D3 Serial #:

If this is a problem report, please provide a complete description of the computer hardware on the system(s) having this problem. Please specify brand, model, size, speed, firmware level, address, IRQ, and configuration information about the device and any controller. Please include information about the following:

CPU:

Memory:

Tape(s):

Printers:

Disk(s):

Serial Ports:

Please describe how the system is currently operating or operating incorrectly:

Please describe how the system should operate:

start buffer mark

used to pad tape records to the length specified during the most recently executed tape attachment command.

The following table shows the different representations for the "start buffer mark":

Displayed form: "["

Hexadecimal value: fb

Decimal value: 251

subvalue

a field used to store data within a value of an attribute within an item.

Subvalues represent the lowest level data component within an item in a file. An item can contain any number of subvalues within a value.

subvalue mark

refers to the specific meta character used within D3 items to indicate the end of a subvalue, and the beginning of another.

The "subvalue mark" is a "reserved" character. This means that it may be not used for any other purpose.

The following table shows the different representations for the "subvalue mark":

Ascii value: "\"

Hexadecimal value: fc

Decimal value: 252

A subvalue mark can be entered in the Update processor by typing <ctrl>+<shift>\ (backslash).

Super-Q-Pointer

extended q-pointer for accessing non-D3 and/or non-local files.

A Super-Q-Pointer is a q-pointer to a file which is not in the local D3 file system. The first attribute is a "Q", the second attribute is blank, and the third attribute contains a host name (from the "hosts" file) followed by a colon followed by the file name on the remote file system.

The first attribute may also be "QS" in which case the file-save process saves the entire remote file as if it existed on the local machine provided the save is used with the "e" option. This data may then be restored into that same remote file or into some other file.

synonym-defining items

also known as "q-pointers," are used in account master dictionaries to point to other files.

The files may be dictionaries or data files within the current, or in other accounts. In general, only the first four attributes are used (attributes 0, 1, 2, and 3).

Attribute 0 (zero) is the item-id of the synonym.

Attribute 1 (one) is called the "d/code" attribute. It contains the character "q" to indicate that the current item is a synonym-defining item.

Attribute 2 is the name of the account (md) where the item-id of the file named in attribute 3 resides. If the account name is not specified, the current account md where the synonym-defining item resides is used.

Attribute 3 is the name of the file to which the item points. If the file name is not specified, the item-id (synonym name) is used for the destination file. If no such file name exists, the current account md is used for the destination file.

Essentially, a "q-pointer" defines a pathway to a file which typically resides in another account. A "q-pointer" may reference any file in the system, including those within the same account.

D3 also provides dynamic file "paths", which prevent having to create a "q-pointer" to simply look at a file elsewhere in the system.

Example

```
  cust
     001 q
     002 production
     003 cust
This points to the "cust" file in the production account.
  ch
     001 q
     002 production
     003 cust,history
This points to the "history" data section of the "cust" file in the production
account.
  prod
     001 q
     002 production
This points to the "prod" file in the "production" account, if it exists,
otherwise it will point to them "md" level file in the "production" account.
  cust
     001 q
     002
     003 customer-file
This points to the "customer-file" in the current account. Note that attribute
2 is null, which indicates the current md.
```

sysbase

refers to the "first" frame where data is stored on on the disk.

The D3 Filesystem begins at fid 1. The following frames contain system information, the lock tables, the D3 Boot ABS, and pib process control blocks.

The sysbase is the address of the system dictionary, or MDS.

The "what" command outputs the sysbase fid.

system delimiters

segment marks, ASCII char 255

attribute marks, ASCII char 254

value marks, ASCII char 253

subvalue marks, ASCII char 252

system privileges

determines which processes the user has access to or can invoke.

There are three privilege levels, "sys0", "sys1", and "sys2". The privilege level is placed into attribute 8 (the "correlative" attribute) of the user item in the "users" file.

"sys0", the lowest level, has the following features:

- Can not use peripheral storage devices.

- Can not start or stop printers.

- Can not update dictionaries, md or "mds" level files.

- Can not change or display anything from the system debugger or the FlashBASIC debugger. Commands are limited to "g", "end" and "off".

"sys1" has the features:

- Can use peripheral storage devices.

- Can not start or stop printers.

- Can use some verbs, such as "create-file", "delete-file", "clear-file", "create-index", "clear-index" and "delete-index", but can not access other file management verbs.

- Can not change or display anything from the system debugger or the FlashBASIC debugger. Commands are limited to "g", "end" and "off".

- Can update dictionaries and md files, but not the "mds" file.

"sys2", the highest level, has no restrictions. It provides access to all processors, provided the operator has access to the appropriate verbs. Some of the verbs which require "sys2" privileges are: "dump", "logoff", "converse", "tandem", "mirror", "init-ovf", "stopptr", "startptr", "clear-locks", "set-baud", "set-port", "set-time", "set-date", and "sp-kill".

tape handling verbs

all processes which handle magnetic media are classified as "tape handling verbs".

All references to "media" and "tape" mean the same thing.

tape socket

defines a tape system across a network. This section is an introduction to the functionality of the tape over a network sub-system.

Documentation Structure

The tape system over a network documentation is divided into the following entries in the Pick Systems Reference Manual:

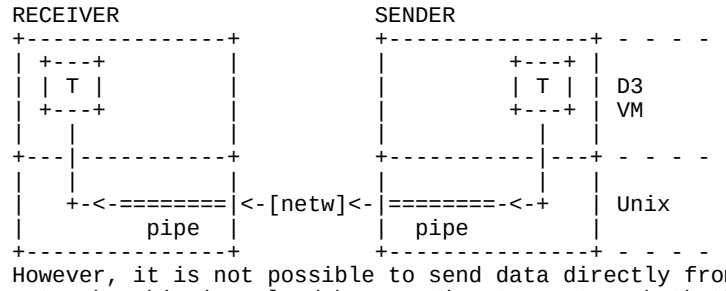
- "tape socket, General": (this section) General introduction about the tape over a network, its capabilities and the fundamental notions.

- "network save/restore, General": How to use the tape over a network utility to do simultaneously a full save on one system, and a full restore on another.

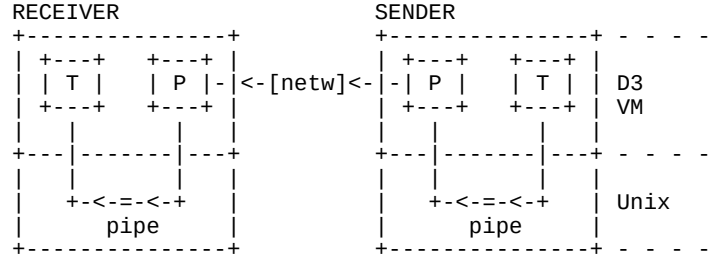
- "tape-socket, TCL": Detailed reference of the TCL command "tape-socket".

Overview :

The objective is to establish a 'pipe' between two systems through which tape blocks can be transferred: On the sender's side, the tape process (eg T-DUMP) writes the tape blocks into a 'pipe', and, on the receiver's side, the associated process (eg T-LOAD), reads the blocks and processes them, as shown in the following figure:



However, it is not possible to send data directly from a pipe across a network. This is solved by creating SERVERS on both the sender and receiver. These servers work closely together to establish a communication path and maintain it. This principle is represented by the following figure:



The principle is to have phantom processes on both the sending and the receiving systems establishing a network communication, and exchanging data with the tape processors through a Unix pipe. On the receiver's side, 'T' is the process reading from tape, doing a T-LOAD, for example, and 'P' is the phantom process handling the network reception.

On the sender's side, 'T' is the process writing to tape, doing the T-DUMP, and 'P' is the phantom process handling the network emission.

On each machine, a Unix pipe is set up to allow data exchange between the phantom process and the tape process. This pipe must be set up in the D^3 configuration file of each machine as a 'network' device (type 'c').

Optionally, the system can be linked to the transaction log system, to automatically create a 'hot backup' system where all updates to a main machines are sent over the network to a backup

system, which is thus a mirror image of the main system, ready to take over the application in case of failure of the main system.

The network is accessed through the BSD 'socket' library.

Servers :

The phantom processes on each system act as a server for the other system. In the following, the phantom process will be called "input server" on the receiver's side and "output server" on the sender's side. On each system, the servers are identified by a "server name". The System Administrator defines all working parameters at install time, and whether the servers should be linked automatically to the transaction logger. On most configurations, there will be only one server. Therefore, the System Administrator would use the 'default' server name, which is used in the menus.

Features :

- Automatic startup of Servers.

- Automatic reconnection of Servers in case of network failures.

- Synchronization of clocks when Servers are started. The sender system executes a remote "set-date" and "set-time" on the remote system with its own time and date, to ensure they are in sync. This can be prevented using the (S) option of the "tape-socket" TCL command.

- Optional tracing of all network traffic.

- Optional automatic 'link' to the Transaction Log sub-system to have an automatic startup of both systems.

- Optional interface to the transaction logger to regularly test that the transaction logger is operating properly.

- Optional notification to a designated D^3 user of network and transaction logger incidents.

- Remote commands can be sent from the Output Server to the remote Input Server. This allows some degree of remote system administration of the Input Server.

- Automatic installation of the software the first time the command is used.

- Simple menu interface for management of the default Server and TCL interface for use in macros.

- Short, rotating, log file, for the most recent error messages, and separate permanent log for long term monitoring.

Automatic Startup :

Once a Server has been set up (see the section "Server Setup" below), a simple TCL command starts it. This command can be inserted in the 'user-coldstart' macro. If the Server has been linked with the Transaction Logger (see below), and if the Server is the Output Server, the Transaction Logger is also started, or re-started, automatically as a phantom process. If the Input Server is linked to the Transaction Logger, then a transaction restore is started automatically on the terminal doing the 'start' command, in which case this command, if inserted in the 'user-coldstart' macro, should be the very last command.

Automatic Connection of Servers :

By default, the Servers are trying to communicate with each other, so that the connection is established automatically. In case of network problem, the Output Server tries every 5 seconds

to re-establish the communication. When the communication is broken, and cannot be re-established in less than 5 seconds, a message is sent to the System Administrator. After this, another message is sent when the communication is re-established. There is normally never a need to stop the Input Server. The Output Server can then be stopped and restarted.

Linking to the Transaction Logger Sub-System :

This feature allows to completely control the Transaction Logger (sender) and the Transaction Restore (receiver) by the tape-socket Servers, in a 'hot backup' configuration. Once the Input Server has been started on the backup system, all operations can be done from the main system: starting the Output Server starts the Transaction logger; stopping the Output Server detaches the tape from the transaction logger, and sends a message to the input server, so that the Transaction restore can be restarted automatically. The Transaction Log Test Polling feature allows detecting transaction logger incidents, by notifying the System Administrator that the test item has been received in time by the backup system (see more on this below). Before starting or stopping the Transaction Logger, the Output Server makes several controls, to be sure the Transaction Logger is ready. If not, the process will be cleaned up, and, if required, a new Unix process will be spawned.

It is important, once the Servers have been linked to the Transaction Log Sub-System, to start/stop the transaction logger and/or restore EXCLUSIVELY through the tape-socket server start/stop commands (or through the menus). Failure to do so, will generate error messages to the System Administrator who will have to do some manual intervention to correct the situation. The 'txlog' Transaction log menu can still be used to obtain the status of the transaction logger, but NOT to start, stop or detach the tape. If it necessary to release the transaction log queue, and, to do so, use the txlog menu, stop the Output Server FIRST, else, not only data will be lost, but the transaction restore will de-synchronized and will have to be restarted manually.

Transaction Log Test Polling :

Periodically, for example every 10 minutes, a test item 'test' is written in a test file "dm,ts.log,test', created automatically. This item contains unique information, and a time/date stamp. If the Transaction Logger and Restore are working properly, this test item is transferred into the same file, on the backup system where the arrival time and data are recorded in the item. Before updating the item, however, the Output Server sends a message to the Input Server to check whether the PREVIOUS test item arrived correctly. If so, everything is fine, and the Output Server is able to calculate an approximate transfer time (which includes not only the actual transfer time, but also the time the test item stayed in the transaction log queue). If the item did not arrive, a message is sent to the System Administrator. Note that, especially if the polling period is short, it is possible for the test item to still be in the queue, and, therefore, make it appear that it was not transferred. If the queue is really large, it is possible that it may take a long time before the test time progresses to the top of the queue before being transmitted.

TCL Command :

The "tape-socket" TCL command is used to create the input or output server and control their activity. See the section "tape-socket, TCL" for more information on the command.

System Setup :

The details of the system setup may vary depending on the application (hot backup, network save/restore, ...). The following are just general guide lines. See the appropriate Pick Systems Reference Manual documentation.

- Establish a network between the two systems. This network must support TCP/IP (eg Ethernet, Token Ring, etc...). The System Administrator must set the network names of both systems, even though only the receiver's host name is used.

- Determine a free TCP/IP port number. Use the "netstat -a" Unix command to see what is currently in use. A value like 2000 or 3000 is usually safe.

- Create Unix pipes on both system, using "mknod", and make sure they have appropriate permissions. Put these pipes as 'pseudo-tapes' in the D3 configuration files of both systems.

- Setup the Servers on both systems (see the section "Server Setup" below).

- Start the Input and Output Servers.

Server Setup :

The "tape-socket" menu has a "Server Setup" option which asks for the necessary running parameters. Some basic elements have to be defined by the System Administrator for the system to operate. The "Server Setup" option asks the following elements (if values are already defined, they are displayed between brackets, and typing <return> leave them unchanged):

Server Name

Using the "other server" submenu, it is possible to manipulate more than one server. A server name is any alphanumeric string ('.' are not allowed), of up to 8 characters. The default server name is 'tserver'.

Server Type

Enter 'in' for the Input Server and 'out' for the Output Server. This value is required.

Remote HOST name

Enter the network name of the system on which the Input Server will reside. This parameter is required for the Output Server setup, to know on what piece of hardware the Input Server will be started. The host name must be declared in the Unix file "/etc/hosts".

TCP/IP port number

Select a valid (>1024, <32767), unused TCP port number. Use the "netstat -a" Unix command to find out which ports are currently used on the system. Both Servers MUST agree on this value, else they will not be able to communicate. This value is required.

Protocol

Currently, only "inet" (Internet Protocol) is supported. This value is required.

Unix pipe name

Select a valid Unix pipe name (eg /dev/tape). If it does not exist, it must be created, and added in the D3 configuration file, if not already there, using the TCL command "config tape", for example. This value is required. See the example in the "tape-socket, TCL" documentation.

Number of trace messages to keep

Log entries are kept in small circular buffer that can be displayed with the "Status" command. Select a number from 4 to 16 for example. All messages (except periodic 'OK' messages) are also kept in the Permanent Log.

D3 user to notify in case of errors.

Enter one or more D3 user ids, separated by commas, the system will send a message to, in case of error. If left empty, then 'dm' or 'sysprog' will be notified. In case of repetitive errors, these messages can become a nuisance, and suppressed later. However, it is STRONGLY advised to select at least one user who would always be logged on to the system so that errors will not go unnoticed. The whole system normally operates without much human intervention, and if notification is disabled, or if the user to notify is not logged on, communication can be interrupted for a very long time before an error is corrected. Errors are also logged into a file. To select all users on the system, enter an asterisk (*). To send a message to a line number, whether it is logged on or not, use an exclamation mark followed by the line number in decimal (eg !0). To disable the notification, enter 'off'.

Transaction Log test polling period

Enter the value in seconds, or in HH:MM:SS of the period with which the Output Server will test the operation of the Transaction Logger. If the Transaction Logger is not used, enter 0 to disable the polling. Refer to the section "Transaction Log Test Polling" above for a detailed discussion of this mechanism. An appropriate value would be between 00:01:00 (1 mn) to 01:00:00 (1 hr). A smaller period could create false alarms by long delays due to the size of the queue. A longer period means that an incident would be unnoticed for a long time. The rule of thumb is that if the normal data update rate is small, then the test period should be small too. In case of mass transfers, like when doing a "touch" (see the TCL command "touch") of an entire data base, which would generate an enormous amount of data transfer, it is advised to temporarily increase the polling period to a large value (say 1 hr), so that the Output Server does not 'panic' at not seeing its updates of the test item go through the network.

Start Transaction logger

This parameter determines whether the Server is to be 'linked' to the Transaction Log Sub-System, to set a 'hot backup' configuration. Type 'on' to establish the link, and 'off' to disable it. When 'on', the Servers will attempt to control the Transaction logger and restore. When selected, be sure to understand the relationship of the two subsystems as discussed in the section "Linking to the Transaction Logger Sub-System" above. The most important rule is to control the transaction logger EXCLUSIVELY through the "tape-socket" menus or commands, to make sure the Servers are aware of the states of the Transaction Log Sub-System.

Log (DL) files or ALL

This parameter is required only for the Output Server if it is 'linked' to the Transaction Logger. It determines the options with which the Transaction Logger will be started when the Output Server is itself started. Enter 'DL' to log only the files for which the attribute 1 of the D-pointer has been changed to have a 'L' code (see the TCL command "set-dptr" for one way of doing this). Enter 'ALL' to log all updates to all files. Be careful with this second option, since updates to system files, like the 'abs', 'users', etc.. will also be logged.

Transaction Log queue period

This parameter is required only for the Output Server if it is 'linked' to the Transaction Logger. It determines, in seconds, the time after which an update is actually transmitted to the remote system. Enter 0 for the default value (currently 30 seconds). When dealing with a fast network, a typical value would be 2 or 3 seconds. This parameter means that an update to a file would stay

on the master system for 2 or 3 seconds, for example, before being sent to the remote system. In case of system crash, only these updates would be lost.

transaction logger

a subsystem designed to write file system updates to another device.

The changes are written to an internal queue. This queue is then written to a device, tape, or network by a dequeuing process run on a phantom line.

If the dequeuing process can keep up with the enqueuing of items by the users, then up to the second crash protection can be accomplished. Used with a file-save tape, all updates will have been recorded for restoration.

If used with another machine it can provide a "hot backup".

See the "tape-socket" menu to setup a Unix pipe on both machines for hot backups.

"Startlog" on the primary machine and "tlog-restore" on the secondary machine will provide the basic functionality for hot backups. See "tape-socket" to setup the proper pipes across the network.

"Txlog" is a menu available to help manage the transaction logger. It can start and stop the dequeuing process as well as filter data going into the queue. All files may be logged or only files with an "l" in attribute one.

"set-dptr" is a verb that can change a dptr or all dptrs in an account to add or remove the "l" from attribute one. If only "dl" type files are being logged then a normal create-file will not cause a create-file on the backup machine. So, changing the d-pointer to "dl" later will not cause a create-file either. This is part of the design and should be noted.

A device must be attached to the port that is starting the dequeuing process. The transaction logger will "steal" the device and begin dequeueing to it. The logger runs on a phantom process, usually the last one, right before the scheduler.

When a "stoplog" is entered, the current reel on the dequeuing device is terminated with a file mark and a "t-det" is executed by the dequeuing process. It will remember which tape device it was using, but now this device is free. The next "startlog" entered will re-attach the tape and increment the reel number.

Each reel in a transaction log dequeue session is independent and can be restored from tcl one after the other or separately.

Multi-user tape releases a list of devices is available to chose from. To cause the startlog command to display this list just detach all the tape devices prior to invoking startlog.

Example

```
set-dptr +l invoices (f
  invoices             :     D -> DL
    invoices           :     D -> DL
set-sct
Block size: 16384
[1709] Tape device is assigned to med density (150M) quarter inch
tape (SCT).
startlog
Activate transaction logger (y/n)?y
[607] Transaction logger started.
 The above sequence will allow all updates to the invoices file to be
enqueued for writing to the SCT drive.
```

```
stoplog
[602] The transaction logger is disabled.
This command will write a file mark on the tape and release it from the
transaction logger, t-det. Updates will still be enqueued automatically. This
is reel one.
startlog
[607] Transaction logger started.
This restarts the logger to the SCT drive on behalf of reel two.
```

Unix Files

accessing a Unix file from the D3 file system.

Through the OSFI, it is possible to access Unix files as if they were D3 items, using AQL, Pick/BASIC, FlashBASIC, etc... This section describes the format of the q-pointer, the file structure and the access rules.

Conventions:

Since the D3 file system structure is fundamentally different from the Unix file systems, a few conventions have to be made to map an object from one file system to the other:

- A D3 item is mapped onto a Unix file.

- By default, the D3 attribute marks are converted to newline characters (decimal 10). This conversion can be optionally disabled.

- Again by default, if a Unix file contains the usual D3 delimiters, they are converted in a sequence of two characters, DLE (decimal 16) followed by a displayable character:

```
SM        DLE       _
AM        DLE       ^
VM        DLE       ]
SVM       DLE       \
DLE       DLE      DLE
```

- Unix text files generally are terminated by a new-line, while D3 text items do not have a trailing attribute mark (The D3 equivalent of a new-line). By default, the terminating new-line (which would be converted to an attribute mark) is stripped when the item is read into D3, and re-appended when that item is again exported to Unix. This provides a comfortable interface for text items, but IT WILL ADD AN ADDITIONAL NEW-LINE WHEN WRITING BINARY ITEMS. Therefore, this default mechanism must be disabled with the "A" option when modifying binary text, and especially when saving a Unix directory.

- Optionally, a section of white-space preceding a block of alpha-numeric text which aligns to a tab-stop can be replaced by the appropriate number of tab characters. This process is reversed if the item is re-transferred to Unix.

- A D3 file is mapped onto one or more Unix directories. The main data level of the file is mapped to a directory which has the same name. The dictionary of the file, is a sub-directory called ".DICT". Other data levels are mapped onto sub-directories of the ".DICT" directory, prefixed with a period. The dictionary is optional, and required only if if data is actually stored in the dictionary. If the dictionary is missing, the file system will open it, but an error will be returned (no item, on a read, and file write protected on a write) if the application actually tries to access items in it. For example, consider the D3 file 'bp', with its directory and two data levels "bp":

```
bp
|
+------+------+------+
|      |      |      |
item1  item2  item3  .DICT
|
+-------+
|       |
ditem1  ditem2
```

This seemingly complex structure aims at making the most common case (flat file) as simple as possible, and to make the internal objects ('.DICT') invisible by default to a D3 TCL 'list' or Unix 'ls' command.

Q-Pointer Format :

The format of the Unix Q-pointer is:

file.name

001 Q

002

003 unix:directory{]options}

"directory" is the name of the Unix directory onto which the main data level of the file is mapped. This directory can be any valid directory name (local directory, mounted Unix removable medium, NFS directory). Special files (device, pipe, etc..) can also be specified with some restrictions.

"options" is an alphanumeric string which controls the behavior of the driver. Spaces can be inserted in the option string for readability. It follows the directory name, separated by a Value Mark:

t n Convert white space preceding text aligned to a tab-stop into a series of tabs. By default no conversion occurs. Note that this conversion option may modify the data (especially binary items) and is therefore only suggested for text files. The valid range of numbers for "n" is 2 to 9.

A Specifies that an extra attribute mark always be added when Unix files are moved into D3, and that that attribute mark always be removed when that item is placed back in Unix. This option is absolutely necessary when saving and/or copying between different files or to backup media. Without this, non-textual items may have an extra new-line appended to them when added to the final Unix destination.

c Specifies the target is a special character file. This option imposes some restrictions (see the section Special Files below).

n Suppress the conversion of Attribute Marks to New Lines. By default, when writing a D3 item, the AM are converted to make the text easy to edit with Unix editors. Note that a trailing attribute mark is added at the end of the Unix file when it is written with this option unless the "A" option is used as well.

R Use raw data. This option reads and writes data much faster than the default and/or using some of the other options. It is not compatible with the A or T options. Furthermore, if a character 255 (segment mark) is read from a Unix file, it is converted into an underscore character.

s Case insensitive item-id and file names. With this option, the filenames and item-ids are converted in lower case, to make them case insensitive. See the section about case sensitivity below.

Item Locks :

Item locks are not supported.

Unix Q-Pointers to Special Files :

It is possible to specify a special character file (pipe, device) as the Unix directory, specifying the 'c' option in the q-pointer. However, there are restrictions:

- Special files cannot have a dictionary or other data levels.

- Only OPEN, READ, WRITE and CLOSE operations are permitted. DELETE is ignored. Sequential access (eg LIST) returns 'no item present'.

- When writing, there is no guarantee that the data is written as one block. This is specially important on pipes for which the notion of atomic write is critical.

- When reading, the device must be able to report the size of the data using the Unix system call 'fstat()'. For example, a pipe may appear empty (size 0) at one point, and then contain data. The application must be prepared to handle empty items.

Case Sensitivity :

When the 'S' option is specified in the Unix q-pointer, the filenames and item-ids are converted to lower case. However, the driver does not detect files that may exist in the same directory with a different case. For example /bp/TEST and /bp/test are two different items. The user must be very careful when using Unix tools to access files otherwise used from D3 through a Unix q-pointer with the 'S' option.

The data is the file is never converted.

Example

```
Examples :
1. Create a Unix q-pointer to a FlashBASIC program file located on Unix:
pgm
    001 Q
    002
    003 unix:/home/dev/bp
        t4
Use the default conversion along with tab expansion.
3. Create a Unix q-pointer to a Unix directory to be saved as part of the
regular D3 file save:
bob
    001 QS
    002
    003 unix:/home/bob
        a
Use the "A" or append option to keep an additional attribute mark on the D3
data (which is stripped when written back to Unix).  This extra attribute mark
ensures that ALL data can be saved and restored without corruption due to
translation.
```

user exits

allows direct references to assembler routines known as "modes". These are generally not needed for new applications and are provided primarily for backwards compatibility and specialty utilities that must access internal structures.

The "unumber" syntax is the standard way to call pre-defined user exits. The number is a four-digit hexadecimal number. Leading zeros may be omitted. For available numbers, follow the "see also".

Users with assembly accounts may also enter their code via user exits. The standard way to do this is with the "u$mode.name" syntax which will jump directly into the mode called mode.name. Every user exit call of this type requires an abs lookup.

The user may initially use the "u?mode.name" syntax and store the result in a FlashBASIC variable. This will NOT execute the user exit, but will return a direct pointer to its location. After this, the mode can be repeatedly called by using that variables value as the conversion. This is much faster than the "u$mode.name" syntax because no further abs lookups are needed. If the "u?mode.name" call returns a null, then the mode cannot be located.

A numeric user exit which is not found in the user exit table is converted to a BASIC call to a subroutine named "U" followed by the number (like "u123"). If the system(0) function is executed at the beginning of this routine, it returns a 1 for an iconv() call or a 0 for an oconv() call.

Syntax

unumber

u$mode.name

u?mode.name

user-id

identifies a user. User-id's are found in the "dm,users," file.

The "admin.maint" menu provides a simple method of adding and changing user-id's.

value

field used to store data within an attribute of an item.

Values represent the data components within an attribute in an item within a file. An item can contain any number of values within an attribute which, in turn, can be composed of any number of subvalues.

value mark

refers to the specific meta character used within D3 items to indicate the end of a value, and the beginning of another.

The "value mark" is a "reserved" character. This means that it may be not used for any other purpose.

The following table shows the different representations for the "value mark":

Ascii value: "]"

Hexadecimal value: fd

Decimal value: 253

var

an OSFI driver providing access to TCL shell variables.

The var driver (most easily used via the "var" file in the "dm" account) provides access to TCL shell variables via a file interface.

A read using the variable name as the item id returns the shell variable's contents.

A read using a TCL command surrounded by back-quotes executes the TCL command and returns the results as if those results had been stored in an item.

A write using the variable name as the item id is identical to doing a TCL "set" on that variable.

A delete using the variable name as the item id is identical to doing a TCL "unset" on that variable.

A select/readnext loop is similar to the TCL "penv" command except that only the variable names are returned. Note that only user-defined variables are returned by the select. Preset and system variables can only be queried by doing a direct read.

Example

```
open "var"
*
* Read a shell variable
*
print @user
*
read xx from "user"
print xx
*
*
* Write a shell variable
*
execute "set myvar=abc"
*
write "abc" on "myvar"
*
* Delete a shell variable
*
execute "unset myvar"
*
delete "myvar"
*
* Display a list of user variables
*
execute "penv"
*
select
loop
  readnext id then print id else exit
repeat
*
*  Execute a tcl command
*
execute "display @`who`"
*
read xx from "`who`"
print xx
```

virtual machine

is a set of resources obtained from Unix at initialization time, which display the functions and characteristics of a D3 computer system, and is shared by several D3 processes.

In this case, a "real" machine (the Unix computer system) emulates a "virtual" machine (the D3 computer system), which then appears to have disk space, tape drives, printers, terminals, users, and the D3 filesystem and language.

Several virtual machines can coexist on any given single hardware platform.

A virtual machine is identified by a unique 'key' allocated by the System Administrator at install time.

A virtual machine must first be 'booted' (i.e. activated) before users can access it. Any user can be 'connected' to any number of virtual machines at any given time.

x

performs a variety of functions, depending upon where it is executed.

At the TCL prompt, <ctrl>+x cancels the current line.

In a multi-page AQL report directed to the terminal, <ctrl>+x terminates the report and returns control to the process which invoked it.

In the Update processor, <ctrl>+x is a preamble to one of the commands used when exiting or saving an item and is always followed by another character.

In the line editor (ed or edit), <ctrl>+x cancels the current line of input.

Syntax

<ctrl>+x

x-off

used in asynchronous communications to suspend the flow of data to a device. Its counterpart is <ctrl>+q ("x-on").

x-on

used in asynchronous communications to resume the flow of data to a device. Its counterpart is <ctrl>+s ("x-off").

AQL (Access)

a D3 facility to retrieve and output data. AQL is an original piece of the pick system and is made up of the List pre-processor (the oldest), the Select processor, and the Output processor.

AQL is a system-level information retrieval language that allows users to query their data base without writing complex programs.

AQL uses TCL commands as verbs and displays the results either on terminals or printers. An AQL verb operates on specified files and items based on various optional criteria, specifications, modifiers, limiters, and options.

Often described as an-ad hoc data query language, the greatly expanded dictionary capabilities of D3 offer the possibility of real nonprogrammer access to the data base.

AQL, used in conjunction with the Update Processor, makes D3 the most accessible data management system in existence.

Additional D3 features enhance the already comprehensive query language. FlashBASIC calls from dictionaries are used for complex data calculations and output formatting. The "ss" (spread sheet) connective allows printing out AQL reports in spread-sheet format. This is achieved by adding the "ss" connective to the sort sentence and defining the desired range parameters.

Use of b-tree indexes has increased the speed and performance of AQL. See "b-tree" in see.also.

AQL commands are entered at TCL and thus can be recalled, modified, or executed through utilization of the TCL-stack. AQL sentences may also be stored and invoked through macros, menus, procs, and FlashBASIC (using the "execute" statement).

An AQL statement has the following form: verb file.reference {item.list} {selection criteria} {sort criteria} {output specifications} {print limiters} {modifiers} {(options)}

The verb and file.reference are required as operator and operand respectively. The verb must be the first word of a TCL command.

All other elements are optional and are used to modify either the operator, operand, or output. Selection criteria, sort criteria, output specifications, print limiters, and modifiers follow the item.list and may be in any order. Options, if used, must be placed last and must be preceded by a left parenthesis. The right parenthesis is optional.

Relational operators may be used with any of the elements of AQL sentences to allow exact specification of the conditions to be met.

"file.reference" is the name of a file in the "md" to which the user is currently logged. It can also be a synonym file name, or "q-pointer". The file name can be preceded by the literal "dict" to access the dictionary of the file instead of the data portion of the file. The default is data. In some cases, data may be specified to indicate only the data portion of the file.

To reference a file in another account or md from TCL, pathnames or q-pointers may be used. The pathname may be used in place of the file.name in any TCL or AQL statement.

A pathname may be entered in one of the four following forms:
account.name,dict.name,file.name
account.name,file.name,
dict.name,file.name
file.name

"account.name,dict.name,file.name" is the fully qualified form, specifying all elements of the path.

"account.name,file.name," defaults the dict.name to the same as the file.name.

"dict.name,file.name" defaults the account.name to the current account.

"file.name" defaults the account.name to the current account and the dict.name to the same as the file.name.

The following form references a master dictionary, defaulting to: mds,account.name: account.name,,

Commands requesting a destination or source with a "to:" or "from:" prompt, such as "copy" and <ctrl>+cp (cut and paste from a specified item), expects an item-id (or itemlist), or a file.reference, or a file.reference and item-id. To name a path or file, the file.reference must be preceded by a parenthesis.

"itemlist" specifies one, some, or all item-ids in the file defined by the associated file.reference. The item.list may be: one or more explicit item-ids, a series of items separated by relational operators, an asterisk (*) to represent all the items in the file, or null. If a list is not active, a null item-id implies a new item for UP and all items for the other processors. To cause a processor to use the active list, the item.list must be null. An item-id with the same name as a language element in either the md or the dictionary of the file must be enclosed in single quotes.

An active list can be used by another "select" statement to narrow down the resulting list. A list can be passed to another list until a "save-list" or any other AQL, TCL2, or FlashBASIC verb is invoked. For example:

select entity with state = "ny"

n items selected.

select entity with city = "syracuse"

m items selected.

list entity name city state phone

The above series of statements first selects all entites in the state on new york, and out of these selects all entites in syracuse and then outputs a columnar report.

"selection criteria" limits the data by specifying criteria that must be met. Multiple criteria may be established to limit data selection to meeting a certain set of limitations. These sets are established by the logical relational connectives "and" or "or".

"sort criteria" are used to define the sort operation.

"output specifications" specifies the attributes to list. The selected attribute items or synonym labels are displayed in either a columnar or non-columnar format depending on the report width. The width of the report is the sum of the width of each attribute to be listed plus one blank separator between each attribute. If the width of the report does not exceed the page width as set by the "term" verb, a columnar format is generated. The attributes for each item are displayed one under the other. If the requested output exceeds the page width, the column headings are listed in a non-columnar format down the side of the output with their respective values immediately to the right. In the non-columnar format, the column headings are listed only if there is a corresponding value.

The item-id is always displayed unless it is suppressed using the "id-supp" connective or unless it is suppressed using the "i" option on the output-macro attribute of the associated file-defining item.

"print limiters" suppress the listing of attributes within an item that do not meet specified limits.

"modifiers" control listing parameters such as double-spacing ("dbl-spc"), control breaks ("roll-on" or "break-on"), column totals ("total"), and suppression of item-ids ("id-supp"), automatic headings ("hdr-supp" or "col-hdr-supp"), and default messages ("ni-supp").

"(options)" are used to tell the processor about special handling and tend to be processor specific. The options are single alpha characters and/or a numeric range specification as required by the specific processor.

Command results are reported to the terminal unless the "p" option is used to direct output to the Spooler. They are usually preceded by a left parenthesis, except in the case of spooler options. At this time, the right parenthesis is optional at the end of the option specification. When used, options must be the last element on the command line. See the "see.also" attribute for a listing of AQL verbs.

Tabbed output is possible by modifying the master dictionary item. See the "tabbed.output" token for more information.

AQL verbs

refer to operations which are performed by the "list" processor, and are one of the three distinct types of verbs in the D3 System.

AQL verbs require at minimum a verb and a file.reference, but they additionally support many qualifiers, such as selection and sorting criteria.

In D3, the "list-verbs" command produces an alphabetically organized list of all verbs in a given md, or the current md if one is not explicitly provided. Any verb with an "a" (or "A") in the "A2" column of the "list-verbs" output is considered to be an AQL verb.

AQL verbs essentially break down into the following categories:

1) columnar-output-producing: sort and list

2) list-producing: select, sselect and nselect

3) statistical: count, sum and stat

4) tape-handling: t-dump, s-dump and t-load

5) file-analysis: hash-test and istat

6) internal (or "copy") format: list-item and sort-item

7) label-producing: list-label and sort-label

8) file-updating: update, reformat and srefomat

Syntax

verb {only} file.reference {itemlist} {sellist} {modlist} {outlist} {(options)}

Example

```
list invoices
list dict invoices
list only dict invoices
sort dict invoices by a2 a1 a2
sort dict invoices with a1 "a]" s]" by ac
```

```
list-item md = 'list-]'
sort-item md = 'list-]' heading "'lc' List items from 'fl'" (p
```

collation order

refers to the sequence in which data is collated or sorted in an AQL process.

The collation order is typically triggered by one of the following AQL modifiers: "by", "by-dsnd" "by-exp", or "by-exp-dsnd".

The term "seqlist" (short for "sequence list") is used throughout the Pick Systems Reference Manual as a convention for representing the collation order.

Example

```
sort entity by zip
sselect invoices by date by customer
sort entity with name = "[mike]" by name
```

connectives

words in a master dictionary which are used to form the elements of AQL statements.

Relational operators are used to establish criteria based on the relationship of data to fixed values or other data.

Relational operators are used, for example, to select a range of zip code values within specified upper and lower limits. This can be done using "gt" and "lt" as well as the "and" relational operators to establish the selection criteria.

"#" or "ne"

test for a "not equal" condition.

"&" or "and"

used between two conditions to indicate they both must be true.

"<" or "lt" or "before"

tests for a less than condition.

"<=" or "le"

tests for a "less than or equal to" condition.

"=" or "eq"

tests for an "equal" condition.

">" or "gt" or "after"

tests for a "greater than" condition.

">=" or "ge"

tests for a "greater than or equal to" condition.

"not" or "no"

tests a negative condition, reversing a true relation.

"or"

is used between two conditions to indicate one or the other must be true.

default output specifications

selects the default attribute-defining items in an AQL sentence when none are explicitly provided.

In addition to explicitly listing attribute names in the AQL statement, there are three features that can be used to specify default output specifications. These specifications output the default attributes when attributes are not explicitly specified in the AQL statement:

- The attribute names can be listed as a macro in the file-defining item (see the description of file-defining items).

- Default attribute items can be created.

- Temporary attribute items can be created.

itemlist

indicates a specific list of item-ids to process.

Each item-id in the list may optionally be enclosed within single quotes. When an AQL sentence contains no specific list of item-id's then, subject to restrictions specified in the optional selection clause, all items in the file are eligible for processing. (See "selection criteria").

In D3, the single quotes around item-ids in an AQL sentence are optional, unless the resulting command is ambiguous.

Syntax

'item-id' 'item-id'

item-id item-id

Example

```
list md '*a0'
This lists the specific item '*a0', from the md of the current
account.
list md *a0
Since "*a0" is a valid attribute-defining item (ADI), this sentence
produces output showing attribute 0 (zero) of every item in the
current md.
list invoices 's1000' 's1010' 's1001'
This sentence requests a specific set of item-ids from the "invoices"
file, and in D3, would also be valid as:
list invoices s1000 s1010 s1001
```

list processor

refers to the facility for selecting, collating and outputting columnar and non-columnar AQL reports.

logical operators

a discussion of the relational and logical operators: =, eq, >, gt, <, lt, >=, ge, <=, le, #, no, not.

Logical operators are used in the selection criteria process. Each operator has an "english" equivalent which may be used interchangeably with the "symbol" form. When no operator is present, the default assumed is an "equal to" (=).

= or eq equal to

or no or not not equal to

> or gt or after greater than

< or lt or before less than

>= or ge greater than or equal to

<= or le less than or equal to

Syntax

with {not} attr.name operator "valuestring"

Example

```
sort entity with state eq "ca"
list invoices with amount.due not "0"
list employees with birthday before "1/2/62"
list employees with age gt "21"
select entity with no zip
sort invoices with every amount > "0"
```

modifiers

list of the AQL modifiers.

Syntax

modifier {modifier...}

modlist

used as a documentation convention to indicate a list of AQL "modifiers".

options:

alter the behavior of AQL sentences.

For example, the "list", "sort", "list-label" and "sort-label" verbs all use a standard output format which by default includes a standard page heading (time, date and page number), and the item-id of each item processed.

"list" and "sort" additonally produce column headings above each output column and generate a "number of items listed" message at the end of the report.

Options are always presented at the very end of an AQL sentence, and must be preceded by a "(" (left parenthesis). The ")" (right parenthesis) is optional. Multiple options may be specified. No delimiter is required between them.

Some combinations of options are not valid. For example, it is unlikely that there would be a need for combining the "d", "i", and "c" options in the same sentence.

Nearly every one of the standard AQL options has an equivalent "modifier" or "connective". The modifier performs exactly the same function as the option, but requires a different syntax.

All macros that contain AQL sentences allow the full complement of AQL options to be passed into the macro for execution.

Syntax

(options

Options

b	Suppresses the "number of items listed" message at the end of most AQL reports. This is the same as the "j" option. See also "ni-supp" modifier.
c	Suppresses column headings above output columns as well as the default page heading on AQL reports. See also "col-hdr-supp" modifier.
d	Suppresses the detail lines normally generated by the associated and encompassing list processor command. See also "det-supp" modifier.
e	Extended output. Forces columnar output irrespective of terminal width. This is useful for software parsing the output results of a query.
h	Suppresses the default page heading (system time/date and page number). See also "hdr-supp" or "supp" modifiers.
i	Suppresses the display of item-ids in the leftmost column of an AQL report. See also "id-supp" modifier.
j	Suppresses the "number of items listed" message at the end of most AQL reports. This is the same as the "b" option. See also "ni-supp" modifier.
k	Suppresses the output of the legend message for list and sort operations. See also "leg-supp" and "legend-supp" modifier.
n	Affects output to the terminal only, preventing the AQL output from pausing at the end of each (terminal) page of output. See also "nopage" modifier.
p	Directs output of the AQL sentence to the system printer, via the Spooler. See also "lptr" modifier.
q	Bypass utilization of indexes.
r	Forces listings into an "across the page" format, if the listing would otherwise be non-columnar. See also "fill" modifier.
s	Suppresses the normal action of attribute-defining items with an attribute-type of "w", thus preventing any embedded Output processor commands from being applied to the data being processed (list and sort only).
t{number}	Displays a periodic total to the screen indicating the current progress of a select, sselect, or count. The approximate number of items selected followed by the approximate number of items scanned is displayed periodically. The period is a default of 100 items selected, but may be changed by appending a number after the "t" option. Note that a select which utilizes an index will NOT display a running total. (select, sselect, and count only). Also note that if a simple "count" or "select" is done, the total counter may not be displayed at the rate of the specified period.

w{-}{number} Displays the output of 'list' or 'sort' verb in a full screen window, allowing the user to scroll backward, forward, left and right through the report. If a minus sign "-" appears just after the w option, then a wide virtual display is used to display the output of the AQL statement. The user can then scroll left and right a screen at a time, as well as up and down a screen at a time. If a number is entered, that number of lines from the top of the first page are stored away as the header, and are displayed on each page. This may cause confusion for reports that display the page number within the header. The h option (hdr-supp) is automatically appended to any reports that do not include a heading statement. A short help screen is available while displaying the output to elaborate on the keystrokes.

y The "y" option displays the AQL "compiled" string to the terminal. This string contains information about the attributes to display on the report in an internal representation for the list processor. If any attribute-defining items being output contain "A" (algebraic) processing codes, each one is translated into an "F" (function) processing code and the "F" code is displayed.

z The "z" option disables the AQL pre-processor.

options: footing

designates a text string composed of literals and special options to output at the bottom of each page. These are the standard "footing" options available to AQL, FlashBASIC, OP and Runoff.

Many options are available to add additional information to the footing line(s).

Options

" (Two single quotes). Outputs (one) single quote.

'aattr.name' Inserts the value of "attr.name". Processes any conversion codes, but not correlatives in the attribute-defining item.

b Inserts the attribute name causing the break if the 'b' option has been specified with "roll-on".

c Centers the output line.

d Retrieves the current date (format dd mmm yyyy).

f{n} Inserts the file name; {in a field of "n" blanks}.

j Right-justifies the rest of the line.

l Issues a carriage return and line feed.

n Prevents the pause at the end of each page (like NOPAGE modifier).

p Retrieves the current page number, right-justified in a field of four blanks.

p{n} Retrieves the page number, right-justified in a field of "n" blanks. (default "n"=4).

r Retrieves the Roman numeral page number.

s Toggles on italics mode.

t Retrieves the current time and date (format hh:mm:ss dd mmm yyyy).

u Toggles on underline mode.

v Toggles on boldface mode.

xc Used with 'c' to center a segment. The 'xc' delimits the end of the segment.

xs Toggles off italics mode.

xu Toggles off underline mode.

xv Toggles off boldface mode.

options: heading

designates a text string composed of literals and special options to output at the top of each page. These are the standard "heading" options available to AQL, FlashBASIC, OP and Runoff.

Options

" (Two single quotes). Outputs (one) single quote.

'aattr.name' Inserts the value of "attr.name". Processes any conversion codes, but not correlatives in the attribute-defining item.

b Inserts the attribute name causing the break if the 'b' option has been specified with "roll-on".

c Centers the output line.

d Retrieves the current date (format dd mmm yyyy).

f{n} Inserts the file name; {in a field of "n" blanks}.

j Right-justifies the rest of the line.

l Issues a carriage return and line feed.

n Prevents the pause at the end of each page (like NOPAGE modifier).

p Retrieves the current page number, right-justified in a field of four blanks.

p{n} Retrieves the page number, right-justified in a field of "n" blanks. (default "n"=4).

r Retrieves the Roman numeral page number.

s Toggles on italics mode.

t Retrieves the current time and date (format hh:mm:ss dd mmm yyyy).

u Toggles on underline mode.

v Toggles on boldface mode.

xc Used with 'c' to center a segment. The 'xc' delimits the end of the segment.

xs Toggles off italics mode.

xu Toggles off underline mode.

xv Toggles off boldface mode.

outlist

list of attribute-defining items to output on the report.

Each adi displays in the order of appearance in the sentence.

Example

```
list entity name contact phone
In this example, the "outlist" is the "name", "contact" and "phone".
```

phrases

Phrases must be defined in the dictionary of the file for which the phrase will be used (or in the account's master dictionary). Line one of the phrase definition must be an H. Line two of the phrase definition contains the words which make up the phrase.

Phrases may contain any legal AQL words and constructions except verbs and file names. This includes output attributes, headings, footings, selection criteria, modifiers (ID-SUPP, DBL-SPC, etc.), even explicit item-id's. Phrases may contain other phrases. Circular phrase references are not permitted and result in an error message.

It is possible to store a collection of phrases in a file and reference them by means of a "remote" phrase definition. A remote phrase definition contains an R on line 1, the phrase file name on line 2, and the phrase item-id on line 3. The item in the remote phrase file must be a phrase definition (H on line 1), and the file and item must exist.

A special phrase may exist in the file dictionary (or in the account's master dictionary if the DICT modifier is specified) which defines the default output attribute list when no explicit attribute names have been included for output. The item-id of this special phrase is the at-sign followed by the file name (@filename) or, optionally, just the at-sign (@). This phrase defines the default output attribute list in the case no output attributes are specified in the sentence. This is used in place of the conventional numbered output attributes (1,2,3...). If the @filename phrase is missing, then the @ phrase is assumed to define the default output attributes. Using this method, dictionaries which define several data files may use different output attribute list phrases.

The @ phrase may have an H or R on line 1, in which case numbered attribute names are illegal. If line 1 is an H or R, all un-quoted words which consist of numeric digits are treated as values. That is, if a user's master dictionary contained the @ phrase with an H or R on line 1, the statement "LIST MD 1 2 3" would list only items '1' '2' and '3' using the default output attribute list (from the @ phrase).

Alternatively, if line 1 of the @ phrase contained an HZ or RZ, numbered attribute names are allowed and will only be considered as values if they do not exist as attribute definitions. For example, if a user's master dictionary contained the @ phrase with an HZ or RZ on line 1, the statement "LIST MD 1 2 3" would list the entire master dictionary displaying attributes 1, 2, and 3 (provided that 1, 2, and 3 were attribute definitions in the master dictionary).

Example

```
By definition a phrase may be any syntactically complete portion of an AQL
statment except verbs and file names.  For example, a phrase named
ADDRESS.LIST in the file CLIENT could be used to display NAME and ADDRESS:
ADDRESS.LIST
001 H
002 NAME STREET CITY STATE ZIP
Once defined in the DICT of the file CLIENT this phrase may be used in an AQL
query such as:
SORT CLIENT WITH STATUS = ACTIVE BY ZIP ADDRESS.LIST LPTR
This statement would produce a report of clients defined as ACTIVE in the data
file and using the phrase definition ADDRESS.LIST as the output
specifications.  Please note that the selection value ACTIVE is not enclosed
in quote marks; another feature of the AQL pre-processor.
A HEADING statement could have been included in the phrase
ADDRESS.LIST, however, a more effective method may be the use of a "remote
phrase."  For example a remote phrase file called STAFF could be created
containing HEADING phrases for each staff member. To use remote phrases an 'R'
type phrase is required in the DICT of the file CLIENT, as well as, a phrase
in the data file STAFF.
DICT of CLIENT
JIM
001 R
002 STAFF
```

```
003 JIM.JONES
DATA of STAFF
JIM.JONES
001 H
002 HEADING "Page 'P'   PREPARED FOR JIM JONES   'CTLL'"
Now the AQL statement:
SORT CLIENT WITH STATUS = ACTIVE BY ZIP JIM ADDRESS.LIST LPTR
Would produce the same report as in the previous example, however, the 'R'
type phrase 'JIM' in the DICT of CLIENT will capture the HEADING from the
STAFF file. Multiple remote phrase files may be used i.e. FROM could be a file
of desired FOOTINGs).
 NOTE  !!!  :  If a code of 'A' was stored in the data file of CLIENT as the
method of indicating a client as ACTIVE the above example would require the
statement be entered as:
SORT CLIENT WITH STATUS = "A" BY ZIP JIM ADDRESS.LIST LPTR
This is due to the fact that "A" is defined in the MASTER DICTIONARY as a
throw away CONNECTIVE and the quote marks are needed to resolve any ambiguity.
This would also apply to other words which are defined in the MD.
 Another type of phrase which may be used is the @filename phrase. The
@filename phrase is used to define the default output attributes when no
output specifications are included in the AQL statement. For example, the
@filename phrase for the file CLIENT would be created in the DICT of CLIENT
under the name @CLIENT:
@CLIENT
001 H
002 NAME CITY PHONE CONTACT
Now if the LIST or SORT verbs are used to query the file CLIENT the attributes
NAME, CITY, PHONE and CONTACT will be the output for statements such as:
LIST CLIENT HEADING "Page 'P'    CLIENT DEFAULT LIST 'CTL'"
or
SORT CLIENT BY NAME WITH TYPE = VAR
The above statements contain no output specifications, therefore, the @CLIENT
phrase is used to determine the output. If the statement was entered as:
SORT CLIENT BY NAME NAME STREET CITY STATE ZIP
Then the default output attributes defined in @CLIENT would be ignored and the
output list requested would be used.
NOTE: If line 001 is defined as HZ or RZ (remote) then attribute numbers (i.e.
1 2 5 7 8) may be used on line 002 to extract the data in those attribute AMC
id's, provided that those AMC numbers are defined in the file dictionary.
 The @filename phrase may also be just the @ sign and the filename will be
assumed.  Also if Q-pointer(s) exist for the file the @filename for the actual
file will be used to determine the output attributes. For example, if CUST is
a Q-pointer to the file CLIENT and the AQL statement is entered:
LIST CUST
Then the @CLIENT default attribute phrase would be used. In other words, @CUST
is not required and is not valid!
A special use of the @filename phrase occurs in the case when multiple data
sections are defined in a dictionary. For example, if in the DICT of our
CLIENT file a data section for ACTIVE and another for INACTIVE are defined
then a separate @filename phrase may also be defined for each data section.
 DICT of CLIENT       DICT of CLIENT
 @ACTIVE              @INACTIVE
 001 H                001 H
 002 NAME CITY PHONE  002 NAME CITY STATE ZIP
Now if the CLIENT file was querried:
LIST CLIENT,ACTIVE
the pre-processor would use the @ACTIVE filename phrase to determine the
default output attributes. And if the file was querried:
LIST CLIENT,INACTIVE
the pre-processor would use the @INACTIVE filename phrase to
determine the default output attributes.
```

pre-processor

the AQL Pre-Processor is an enhancement to the AQL querry language used on D3. The pre-processor provides two major functions. First, it adds the ability to use and define phrases which contain parts of AQL statements. Second, it eliminates the requirement that explicit values be enclosed in quotation marks, except to resolve ambiguities.

The AQL Pre-Processor is enabled by changing the verb definition in the M/DICT for any AQL verbs for which pre-processing is desired. The pre-processor scans the input sentence, expands phrases and determines which words are values, then passes the sentence to the standard AQL compiler.

In normal operation, the AQL Pre-Processor processes the input statement before the AQL compiler.

Quotation Marks

With the AQL Pre-Processor, quotation marks are not required to distinguish values from other parts of an AQL sentence. Any word which is not in the file dictionary (or M/DICT), is assumed to be a value, provided that it makes sense to use a value in that part of the sentence. If it does not make sense, then an error message is printed and the sentence is aborted.

It is still necessary to use quotation marks when there is an ambiguity in the meaning of a word. For example,

LIST MD D/CODE D/CODE

could mean to list the item 'D/CODE' showing the attribute D/CODE, or it could mean list all items showing the attribute D/CODE in two columns. In this case, quotation marks are required to distinguish between the first and second meaning. Also, if a value contains blanks, quotation marks are required to distinguish between a single value with embedded blanks, and multiple values separated by blanks.

Headings, footings, break-on options and grand-total headings must still be enclosed in quotation marks.

Options

y causes a message to be displayed whenever the Pre-Processor assumes an undefined word is a value. The message shows the undefined word and the context it is assumed to be used in. The processed sentence is then displayed on the screen before entering the AQL compiler. The terminal will wait for a key to be pressed before proceeding. If a control-X is entered, then the sentence will be aborted.

z used to override the AQL Pre-Processor. If the Z option is present, the pre-processor is bypassed, and the AQL compiler is entered directly.

Example

```
The following examples show typical AQL statements in which quote marks are
not required. Each of these examples will utilize the 'Y' option which will
display a message defining how the pre-processor interprets the statement.
SORT SALES BY DATE WITH DATE >= 1/1/87 AND <= 12/31/87 (Y
[2393] Assumed '1/1/87' is a value used as selection criteria on
attribute 'DATE'.
[2393] Assumed '12/31/87' is a value used as selection criteria on
attribute 'DATE'
SORT SALES BY DATE WITH DATE >= "1/1/87" AND <= "12/31/87"
In this example the AQL pre-processor identified one selection
clause with two values and then showed the AQL statement complete with quote
```

```
marks.
LIST ORDERS 1234 1235 (Y
[2391] Assumed '1234' is an item-id, or modifies the item-id.
[2391] Assumed '1235' is an item-id, or modifies the item-id.
LIST ORDERS '1234' '1235'
In this example the pre-processor correctly identified two item-ids and then
showed the AQL statement including quote marks.
SORT G/L BY-EXP BALANCE > 1000.00 (Y
[2395] '1000.00' is a value used to modify the sort clause
       on attribute 'BALANCE'.
SORT G/L BY-EXP BALANCE > "1000.00"
This example shows that the pre-processor is able to identify that
1000.00 is a value used to modify the BY-EXP sort clause.
LIST G/L BALANCE > 1000.00 (Y
[2392] '1000.00' is a print limitter which modifies 'BALANCE'.
LIST G/L BALANCE > "1000.00"
This example shows the AQL pre-processor identify that the value
'1000.00' is a print limitter modifying the output attribute
'BALANCE'. The statement will list all items but only values greater than
1000.00 will be printed for the column BALANCE. NOTE that BALANCE is an output
attribute. If the word 'WITH' was included in this statement then 'BALANCE >
1000.00' would be a selection criterion.
```

quotes

discussion on the rules pertaining to string delimiters imposed by AQL sentences.

The use of single or double quote marks depends on the nature of the AQL sentence.

For referencing values (data elements), the string should be enclosed in double quotes:

list customers with name = "fred]"

For referencing item-ids, the (item-id) string should be enclosed in single quotes:

list customers '100"101"102'

Spaces between item-id's are optional.

Both item-id strings and value strings may be included in an AQL statement:

list customers '100"101"102' with contact = "[barney]"

The above case shows when D^3 enforces the rules about single quotes surrounding item-ids and double quotes surrounding value strings.

The backslash (\) character may also be used as a string delimiter. This is useful when the string being searched for contains both single and double quotes.

The previous narrative explains the "traditional" rules of enforcement of the use of quotes. In D^3, certain exceptions to the rules are allowed.

For instance, in the sentence:

list entity 88971677 99171776

The single quotes that would ordinarily be required around the specific item-id's are no longer required, provided that there is no ambiguities in the AQL sentence. AQL attempts to locate the unresolved items in the AQL sentence as specific item-id's in the target file, if they could not be resolved as attribute-defining items.

This new set of rules simplifies the syntactical requirements for AQL sentences, yet introduces new variables into the interpretation of AQL sentences. For example, in the sentence:

list md 'description'

This requests the specific item-id 'description' from the md of the current account, and only shows that item. By contrast, the sentence:

list md description

displays the "description" attribute for every item in the current md, since "description" happens to be a valid attribute-defining item.

Syntax

'item-id' {'item-id' ...}

with adi operator "valuestring" {and|or with...}

'item-id' {'item-id' ...} with adi operator "valuestring" {and|or with...}

retrieval scheme

how words in a sentence are extracted either from a specified dictionary or indirectly from the md.

The AQL processor uses both the master dictionary and the file dictionary to determine the definition of the elements in the AQL sentence.

The file pointer to the file dictionary and the connectives used in the sentence, for example, are found in the master dictionary.

The file pointer to the data file and file-specific attribute-defining items are found in the file dictionary.

If an element is defined in both the master dictionary and the file dictionary, the definition in the file dictionary is used.

If the element is not found in either the master dictionary or the file dictionary, the AQL processor creates a new element by concatenating the unknown element to a blank and the next element in the string. The processor first attempts to look up this new element in the file dictionary. If it is not found in the dictionary, it looks in the master dictionary. If the new item-id is not found, an error message displays.

The AQL processor does not look up terms in the string that are enclosed in quotes, single quotes, or backslashes. These are assumed to be literals.

Example

```
list entity name *a9999
In this example, "name" is found in the dictionary of "entity", and
"*a9999" is found in the md, making this a valid AQL command.
```

selection criteria

determines which items are to be processed by AQL and is designated by the "with" or "if" modifiers.

Without any "selection clauses" in an AQL sentence, every item in the file is eligible for processing.

Compound selection clauses are formed with the "and" or "or" relational operators. When two clauses are connected by an "and", both conditions must evaluate to "true" for an item to be selected for processing. When two selection clauses are connected with an "or", or no relational operator is specified between selection clauses, either condition may evaluate to true for the item to be selected for processing.

The value specified in the "valuestring" clause must match exactly with the stored value for selection to occur. For example, in the sentence:

list entity with city = "irv" city

Only those items which contain "irv" in the "city" attribute are eligible for processing.

String searching:

AQL permits the use of "string searching" (sometimes called "wild cards") which allow introducing variables into the selection criteria. For example, in the sentence:

list entity with city = "irv]" city

The "]" character specifies that any character(s) may follow the literal string, "irv". This means that both "irvine" and "irving" are selected. In the sentence:

list entity with name "[inc."

Any item whose "name" attribute ended with "inc." is eligible for processing.

The "bracket" ("string searching") characters may also state an "including" or "containing" condition, as in the form:

list entity with name = "[pick]"

Which includes in the report any item whose "name" attribute contains the string "pick". This includes "Pick Systems", "Movenpick Restauraunts" and "The First Pick".

Example

```
list entity with name "ar]" name
In this selection criteria, only those items whose "name" field
begins with "ar" are selected.
list entity with name "mi]" and with phone "804]"
This example shows two selection clauses connected with an "and",
which means that both conditions must be evaluated as true in order
for the items to be selected for processing.
list entity with name "re]" or with city "irv]" name city
This example shows a mutually exclusive set of selection crtieria
connected with an "or". Either condition evaluating to true will
accept the item for processing. That is, if the "name" attribute
begins with "ar", or the "city" attribute begins with "irv", the item
will be selected for output.
list entity with name "ar]" and with city "irv]" or with city "san]"
and with contact "[joe]"
There is virtually no limit to compound selection criteria in an
AQL sentence, with the exception of the fact that AQL limits a
sentence to a maximum of 9 "and" clauses.
```

sellist

represents the set of selection criteria clauses that determine which items should be selected for processing by AQL.

See the "with" modifier for additional information on selection criteria clauses.

Example

```
list entity with name "ar]" name
In this example, only those items whose "name" field begins with "ar"
are selected. (See "selection criteria" for more information).
list entity with name "mi]" and with phone "804]"
This example shows two selection clauses connected with an "and",
which means that both conditions must be evaluated as true for items
to be selected for processing.
list entity with name "re]" or with city "irv]" name city
This example shows a mutually exclusive set of selection crtieria
connected with an "or". Either condition evaluating to true accepts
the item for processing, that is, if the "name" attribute begins with
"ar", or the "city" attribute begins with "irv", the item is selected
for output.
list entity with name "ar]" and with city "irv]" or with city "san]"
and with contact "[joe]"
This example lists those items which meet one of two criteria: a name
starting with "ar" and with a city starting with "irv", or, a city
starting with "san" and a contact containing the string "joe".
```

seqlist

defines AQL collation sequence.

The sort(s) invoked by the "by", "by-dsnd", "by-exp", or "by-exp-dsnd" modifiers a left-to-right order determine the order in which items are processed by AQL.

There is no limit to the number of sort keys within an AQL report.

The "by" modifier automatically sorts the specified attribute in ascending sequence, and only applies to the first value in the designated attribute, even if it contains multiple values.

The "by-dsnd" modifier sorts in descending sequence, and only applies to the first value in the designated attribute, even if it contains multiple values.

The "by-exp" modifier performs an "exploded" sort and treats every value in a multi-valued attribute as a separate item for sorting purposes. "by-exp" sorts in ascending sequence. "by-exp-dsnd" also treats each every value as a separate item for sorting purposes, but sorts in descending sequence.

Example

```
sort entity by name name
This example has one sort key. It produces a listing of the entity
item-id and name attribute sorted by the value of the first name in each
entity item.
sort entity by state by city by name
This example has three sort keys. First, the data in "entity" is
sorted by ascending "state" sequence, and within state by ascending "city"
sequence, and within city by ascending "name". The attributes listed in the
file's macro or output-macro are displayed.
sort invoices by-dsnd amount.due
This examples reverses the normal sort order and sorts the highest
values to the top of the list, which is useful for circumstances such as
determining who owes you the most money.
```

string searching

uses the "[" and "]" characters to search for strings beginning with, ending with, or containing a given substring.

When used in conjunction with the "with" modifier, data elements may be searched for the occurrence of a string of characters, as illustrated in the examples.

Example

```
list invoices with name "[PICK]"
lists all invoices with the string "PICK" anywere in the name
attribute. Any other string may proceed or follow the letters "PICK".
Select invoices by name with name # "[PICK"
selects a sorted list of invoice items whose name ends with the
string "PICK". Any string of characters may precede "PICK", but the
element must end with "PICK".
list invoices with name # "value]"
List those items whose name being searched "begins" with "value". Any
string of characters may follow "value", but the element must begin
with "value".
"Wildcard" search. Used for searching for any characters within
strings:
select invoices with name = "[sm^th]"
Retrieves any item containing at least five characters, the first two
of which begin with "sm", followed by any character, and ending with
"th". This retrieves "smith" as well as "smyth".
list entity with name = "i^m"
lists entity file items whose "name" attribute contains a string
beginning with "i", followed by any character, and ending with "m".
list entity with name = "a^^"
list entity items whose Searches attribute for values containing a
string beginning with "a" and followed by two characters. any two
characters following the "a" will meet the selection criteria.
Multiple selection criteria clauses:
When applying multiple selections against a single attribute value,
it is not necessary to build a separate selection criteria phase for
each, as in the following examples:
list customers with state = "ca" or with state = "az" or with state =
"va"
This could also be stated as:
list customers with state "ca""az""va"
```

Tabbed output

AQL allows another form of output referred to as tab delimited output for use when transferring data from D^3 to PC applications. Optionally, this can be selected by modifying the AQL verb that will be used in the AQL statement. This is done by adding an additional modeid (53) to the third value of attribute 2 of the master dictionary definition of the AQL verb that is being used. AQL will produce output delimited by tabs and terminated with a CR for transfers to applications running on PC's.

temporary attribute items

a brief discussion on the use of temporary attribute-defining items in AQL sentences.

Attribute items using special attribute names can be specified in an AQL sentence without actually existing in either the file dictionary or the master dictionary.

The attribute name is of the form "Aac", where "A" is the literal "A" and "ac" is the attribute number.

Temporary attribute items are created with a justification code (attribute 9) of lx (left justify and expand display field to fill report). For example, even if neither the master dictionary nor the file dictionary for entity has an attribute-defining item "a14", a statement such as "list entity a14" lists attribute 14 in the entity file where "a14" is the temporary attribute name.

The two reserved numbers 9998 and 9999 may also be specified as part of temporary attribute-defining items. See "*a9998" and "*a9999" for descriptions.

Example

```
list md with a15 a15
```

user exits, Access

allows direct access to system functions. The following is a list of user exits commonly used from within AQL.

wildcards

are used for "string searching". This allows searching for strings beginning with, ending with, or containing a given string.

Wildcard characters may be used to select item-ids and attributes based on common characters. Wildcards can be used in the selection criteria or in complex item.lists.

[(left bracket). Matches characters following the bracket. Ignores characters in the string to the left of the bracket.

] (right bracket). Matches characters from the beginning of the string to the bracket. Ignores the characters in the string to the right of the bracket.

^ (caret). Matches any character in the position occupied by the caret. If both left and right brackets are used, the character string may appear anywhere in the attribute value. To select, sort, or list item-ids, a relational operator precedes the item-id containing the wild cards. Other attributes are identified by either their attribute number or their name as specified in the file dictionary attribute-defining item.

Example

```
list entity with name = "sam]"
This lists any item whose "name" field starts with "sam".
list entity with name = "[son"
This lists any item whose "name" field ends with "son".
list entity with name = "[sam]"
This lists any item whose "name" field contains "sam".
list entity with name = "[p^c]"
This lists any item whose "name" field contains at least three
characters, beginning with "p", and ending with "c". This includes
"Pick Systems", "Able Packing Co." and "Starving Movers, plc".
```

b

Suppresses "number of items listed" message.

c

The equivalent modifier is "col-hdr-supp".

d

Suppresses detail on AQL reports.

h

Suppresses default page heading.

i

Suppresses item-ids.

j

Suppresses "number of items listed" message.

m

Allows all values to be processed by the output conversion -- even nulls.

Normally, null values are not processed by the output conversion so that they do not necessarily take up space on a report. However, some output conversion, such as those that call basic subroutines, may need to see each and every value, even if it's null.

The equivalent modifier is "converting-nulls".

n

Outputs report without pausing at the bottom of each page.

p

Directs output to printer.

r

Displays report in "across-the-page" mode.

y

displays the AQL "compiled" string to the terminal. This string contains information about the attributes to display on the report in an internal representation for the list processor. If any attribute-defining items being output contain "A" (algebraic) processing codes, each one is translated into an "F" (function) processing code and the "F" code is displayed.

The first line of the display contains the actual name of the D-pointer from which the report was generated (prefixed by a "D") along with a list of all internal processing codes (indexes, audit stamps, etc.) obtained from the D-pointer.

Example

```
list ap.doc (y
```

z

prints the AQL "compiled" string to the printer. (Unlike the "y" option, which directs output to the terminal.) This string contains information about the attributes to display on the report in an internal representation for the list processor. If any attribute-defining items being output contain "A" (algebraic) processing codes, each one is translated into an "F" (function) processing code and the "F" code is displayed.

The first line of the display contains the actual name of the D-pointer from which the report was generated (prefixed by a "D") along with a list of all internal processing codes (indexes, audit stamps, etc.) obtained from the D-pointer.

Example

```
list ap.doc (z
```

#

"not equal to" operator.

&

"and" operator.

<

"less than" operator.

<=

"less than or equal to" operator.

=

"equal to" operator.

>

"greater than" operator.

>=

"greater than or equal to" operator.

a

Throwaway connective.

after

is a relational operator which represents a "greater than" condition used in constructing selection clauses.

The "gt" and ">" operators perform the same function.

Syntax

with {not} attr.name after "valuestring"

Example

```
list customers with name after "c"
list customers with amount.due > "900"
sort customers with last.purchase gt "1/1/93"
```

and

used between two selection clauses to indicate that both clauses must evaluate to "true" to select the item for processing.

The "&" connective performs the same function as "and".

Syntax

with clause and with clause

Example

```
list entity with name "ar]" and with city "irv]" or with city "san]"
and with contact "[joe]"
This example lists those items which meet one of two criteria: a name
starting with "ar" and with a city starting with "irv", or, a city
starting with "san" and a contact containing the string "joe".
```

any

search entire item for a match, all attributes and all values.

Each value is compared to the selection criteria and if the value passes the criteria, then the item is selected for further processing. This feature is useful for searching non-attribute oriented files for specific words or phrases, such as word processor documents, source programs, etc.

The word ANY may be used to mean ANY attribute, value, or subvalue in the item. ANY may be used in selection criteria (WITH, IF), or specified for output. If ANY is specified for output, it must also be specified in selection criteria. The output of the ANY attribute definition will be any subvalue, value or attribute which passes the selection criteria.

Syntax

with {not} any attr.name {operator} {"value"} {any}

Example

```
select entity with any name "[john doe]"
This selects all entity items which contain the name "john doe"
list entity with any "sally" any
Page 1     entity
entity.... ........................
332        004 sally
1234       001 sally
This indicates that the "any" found two items which had an attribute, value,
or subvalue which exactly matched the string "sally".  The output of the
second "any" displays the attribute, value, or subvalue number on which the
match was found as well as the actual data in that field.
```

are

throwaway connective.

before

is a relational operator which represents a "less than" condition used in constructing selection clauses.

The "<" and "lt" operators perform the same function as "before".

Syntax

with {not} attr.name before "valuestring"

Example

```
list employees with birthday before "1/1/62"
list journal with amount.owed and with last.payment < "1/1/93"
list journal with not amount.owed lt "900"
```

break-on

creates a visual "break" in output data when the value in the current output field is different from the previous value.

The "break-on" modifier groups items in a listing according to the value of the break-on attr.name(s). attr.name indicates the attribute on which a break will occur.

During the list or sort operation, a control-break occurs whenever there is a change in the value of the specified attribute. Value comparison is made on a left-to-right, character-by-character basis, with a maximum of the first 24 characters being used in the comparison only.

When a control-break occurs, it prints a preceding blank line, a line with three asterisks (***) displayed in the break-on attribute column (i.e., the attribute whose value has changed, thus causing the break), and a following blank line. If the optional text string is specified, the processed text string will be substituted for the asterisks.

Up to 15 control-breaks may be specified, the hierarchy of the breaks being specified by the sequence of the break-on modifiers in the AQL sentence, the first being the highest level.

For multiple control-breaks, output proceeds from the lowest level break to the highest level. The data associated with the lowest level control-break are printed on the current page (even if the end of the page has been reached). If multiple breaks occur, normal pagination proceeds on the second and subsequent data lines, unless an option prevents this.

Headings and output control options may be specified for control-breaks. A user-generated heading can be specified to be printed in place of the default control break heading ("***") by following the break-on attr.name with the desired heading, enclosed in double quote marks ("). Within the heading, output control options may be specified, enclosed in single quote marks ('). The text, if specified, replaces the default asterisk field ("***") in the attr.name column when the control-break printout line occurs.

If the "det-supp" modifier is specified in the sentence, then the attribute value will display on the break line, unless a text string was specified.

Options are used to modify some of the actions taken at control-break time, and are specified as one or more characters. If options are used without accompanying text, they must be enclosed in single quotes within double quotes (e.g., "'v'").

If the "total" modifier is specified in the sentence, subtotals are also output at each control break, as well as columnar totals.

The data associated with the break-on attribute may be suppressed in the detail lines by using a max length of zero in the attribute-defining item (line 10 in the attribute-defining item). If suppression of the data associated with the control break is desired at total time, it must be done with an f;"" in attribute 7 (the output-conversion attribute) of the attribute-defining item associated with the control break.

In generating the value for comparison, correlatives in the attribute-defining item are processed but conversions are not.

Syntax

break-on attr.name {"{text}{'options'}"}

Options

b Outputs the value causing the break in either the heading or footing output field where the 'b' option is found in the heading or footing option string. It is not meaningful to specify this option within more than one break-on specification.

d Suppresses the break data line if only one detail line has been output since the last control break.

l Suppresses the blank line preceding the break data line. This option is ignored when the 'u' option is used.

n Resets the page counter to one on each break.

p Issues a form feed (page eject) after the data associated with this break has been output.

r Inhibits page "rollover". Outputs any occurrence of one or more control break lines at the end of the page, rather than at the top of the next page, thus forcing all the data associated with this break to be current on the same page.

u Underlines all total fields.

v Outputs the value causing the control break in the break-on label.

" (two single quotes) Inserts a single quote in text.

Example

```
sort invoices by date break-on date
sort sales by salesman break-on salesman "'bp'"
```

by

designates a sort key, in ascending order. It must be is followed by an "attr.name".

In D3, "by" may be used with any verb that lists or selects items, such as "list" or "select".

Syntax

by attr.name {by attr.name ...}

Example

```
sort entity by zip
sselect invoices by date by customer
sort entity with name = "[mike]" by name
```

by-dsnd

designates a sort key, in descending order. It must be is followed by an "attr.name".

In D3, "by-dsnd" may be used with any verb that lists or selects items, such as "list" or "select".

Syntax

by-dsnd attr.name {by-dsnd attr.name...}

Example

```
sort invoices by-dsnd amount.due
```

by-exp

designates a sort key, in ascending, "exploded" multi-value order. It must be is followed by an "attr.name".

The "by-exp" modifier is used to sort attributes that may contain more than one value. When the "by" modifier is used on a multi-valued attribute, it only sorts on the first value in the attribute. Further, each item-id will only appear one time, assuming that its display is not suppressed.

When the "by-exp" modifier is used, it "explodes" each value in the attribute and treats each value as though it were a separate item. Each value sorts into its proper position, and the item-id will appear with each value. In other words, the number of times and item-id will appear on a report coincides to the number of values in the exploded attribute.

In D3, "by-exp" may be used with any verb that lists or selects items, such as "list" or "select".

The "explosion limiter" specifies that output only occurs on data elements meeting the print limiting criteria.

Syntax

by-exp attr.name {"explosion limiter"}

Example

```
sort customers by-exp invoice.pointers
sort invoices by-exp product = "widget"
select fn by-exp pay-date >= "01/01/93" and by-exp pay-date <=
"12/31/93".
Note that in this example, the second "by-exp" is required to return
the correct data.
```

by-exp-dsnd

followed by an attribute name which references a multi-valued attribute to be used individually as sort keys in descending order.

The "by-exp-dsnd" modifier is used to sort attributes that may contain more than one value. When the "by" modifier is used on a multi-valued attribute, it only sorts on the first value in the attribute. Further, each item-id will only appear one time, assuming that its display is not suppressed.

When the "by-exp-dsnd" modifier is used, it "explodes" each value in the attribute and treats each value as though it were a separate item. Each value sorts into its proper position, and the item-id will appear with each value. In other words, the number of times and item-id will appear on a report coincides to the number of values in the exploded attribute.

The "explosion limiter" specifies that output only occurs on data elements meeting the print limiting criteria.

This can be used with any verb that lists or selects items, such as "list", "sort", or "select".

Syntax

by-exp-dsnd attr.name {"explosion limiter"}

Example

```
If one invoice 0001 exists with quantity ordered attribute as
follows:
005 1
    6
    3
sort invoices by-exp-dsnd quantity.ordered
The above statement would result in the following.
0001   6
0001   3
0001   1
```

c/total

displays the total number of frames that make up a binary item. The frame count is divided into 2 parts, the number frames of D3 object code and the number frames of FlashBASIC object code.

It is used by AQL and list-obj to add together attributes 3 and 12 of compiled date-time stamps.

Example

```
list dict bp c/total
Page   1     DICT bp
DICT bp........ frames
test                1
test2               4*
Notice the "*" indicates this program has been compiled with
FlashBASIC.
```

check-sum

calculates a hexadecimal "checksum" for a specified attribute, or the entire item if no attribute is specified, and produces result as a list.

The checksum statistics change whenever a file is updated, which provides a means of determining if a file has been changed.

When an "itemlist" is specified, statistics are calculated only for the selected item-ids. If not specified, the entire file is used.

If an "attr.name"is specified, statistics are generated only for the attribute name or number specified. If "attr.name" is not used, or if the specified attribute number is 9999, the entire item is used.

The attribute mark trailing the specified attribute (or item) is included in the statistics.

To use checksum statistics, "check-sum" all files, or portions of files to be verified and record the output statistics. At a later time, repeat the check-sum process. To verify that the files have not changed, manually compare the two checksums.

Syntax

check-sum file.reference {itemlist} {attr.name} {sellist} {modlist} {(options)}

Options

see "Options: AQL"

Example

```
check-sum bp
Byte statistics for : bp
               Total Average Maximum  Min
Bytes      =   39546  919.67    8117   13
Values     =    1614   37.53     320    1
Attributes =    1614   37.53     320    1
#Items     =      43
Chksum     =EBF3CE55
Bits       =  121447
```

col-hdr-supp

suppresses column headings above output columns and the default page heading on AQL reports.

The equivalent option is "c".

converting-nulls

allows all values to be processed by the output conversion -- even nulls.

Normally, null values are not processed by the output conversion so that they do not unnecessarily take up space on a report. However, some output conversion, such as those that call basic subroutines, may need to see each and every value, even if it's null.

The equivalent option is "m".

count

produces a total of the number of items that meet the selection criteria as determined by the "itemlist" and "sellist" specifications.

Syntax
count file.reference {itemlist} {sellist} {modlist} {(options)}

Options
see "options: AQL"

data

throwaway connective.

dbl-spc

double spaces output by inserting a blank line between items.

det-supp

suppresses the detail lines in a report.

"det-supp" provides the ability to easily fold detailed information into summary reports.

The "det-supp" modifier is typically used in conjunction with the "total", "break-on", and "roll-on" AQL modifiers.

The equivalent option is "d".

dict

used immediately preceding a file name affects the dictionary level of the specified file rather than the data section.

Syntax
dict file.reference

each

specifies that each value in a multi-valued attribute must meet the specified selection criteria to be eligible for processing.

The "every" modifier performs the same function as "each".

Syntax
with {not} each attr.name {operator} {"value"}

el

retrieves a previously saved list and enters the Update processor (UP).

This command uses UP to edit a list created by the "save-list" verb. Each element in the list is treated as a line by UP. All UP functions are valid.

If no file.reference is indicated, the list is retrieved from "pointer-file".

If no itemlist* is indicated, the default name, "%user.name", is used.

Syntax
el {{file.reference} itemlist} {(options)}

Options
z Suppresses "top" and "eoi" messages.

Example
```
el invoices.past.due
el dict customers cust.list
el
pointer-file %jmr NEW ITEM
01
In this form, a "default" list name is used, derived from the current
user-id.
```

eq

"equal to" operator.

equal

is a relational operator which represents an "equal to" condition used in constructing selection clauses.

When a "valuestring" is included, the data found in each item must match it exactly in order for the data to be selected for processing. See "string searching" for searching for partial matches.

The "=", "is" and "eq" operators perform the same function as "equal".

Syntax
with {not} attr.name equal "valuestring"

Example
```
list jobs with user equal "dm"
```

every

Forces comparison on every value, rather than any value.

file

Throwaway connective.

fill

forces listings into an "across the page" format if the listing would otherwise be non-columnar.

The headings and values are printed one after another across the page. This has no effect on columnar listings. The equivalent option is "r".

Example
```
list entity name address fill
```

footing

designates a text string composed of literals and special options to output at the bottom of each page.

Syntax

footing "{{text} {'options'}...}"

Options

* see "options: footing".

Example

```
list entity name phone footing "'lc'page 'p' printed at 'tlc'this end
into shredder first"
```

for

Throwaway connective.

ge

a relational operator which represents a "greater than or equal to" condition used in constructing selection clauses.

The ">=" operator performs the same function as "ge".

Syntax

with {not} attr.name ge "valuestring"

Example

```
list invoices with amount ge "0"
```

grand-total

substitutes a string of text in place of the standard "***" literal string normally output on the grand total line.

This produces a grand total only in statements that include the "total" modifier. Multiple options may be enclosed in the same set of single quotes.

Syntax

grand-total "{{text} {'options'}...}"

Options

'l' Suppresses the blank line preceding the grand total.

'p' Issues a form feed (page eject) before the grand total.

'u' Underlines all totaled fields with a row of equal signs (=).

" (Two single quotes). Inserts a single quote in text.

Example

```
sort journal by company by salesman with age > "120" amount.due
salesman company total amount.due grand-total "'u'Total Due" roll-on
company roll-on salesman det-supp
```

gt

"greater than" operator.

hash-test

produces a "histogram", or graphic representation of item-id distribution within a given file, according to a "test" modulo provided. The command is useful when performing file reallocation providing insight into how the item-ids hash into the file, as well as providing a suggested modulo.

Syntax

hash-test file.reference {itemlist} {sellist} {(options)}

test modulo:modulo <return>

Options

* see "options: AQL".

s Displays summary statistics only.

Example

```
hash-test testfile<return>
test modulo: 3<return>
file= testfile     modulo= 3        10:07:24 03 Mar 1992
frames  bytes items
1    444    1 *>
1    772    2 *>>
1   1742    2 *>>
3
Hash-Test/Istat Statistics
total item count =     5        byte count =  2958
avg. bytes/item  = 591.6  avg. items/group =   1.6
std. deviation   =    .5  avg. bytes/group = 986.0
suggested modulo =     3
This file has three groups, a total of five items. Each ">" ("hash
mark") indicates an item.
```

hdr-supp

suppresses the default page heading (system time/date and page number) on output. The equivalent option is "h" and a synonym modifier is "supp".

Example

```
list entity hdr-supp phone (p
list entity phone (hp
The above examples produce the same output.
```

header

see "heading".

heading

designates a text string composed of literals and special options to output at the top of each page.

The "header" AQL Modifier is a synonym for "heading" AQL Modifier.

Syntax

heading "{{text} {'options'}...}"

Options

* see "options: heading".

Example

```
list customers heading "'lc'doolittle enterprises'lc'customer
report'l'"
This heading first issues a linefeed, then centers "doolittle
enterprises" on the output device. Then it issues another linefeed
and centers "customer report" on the next line. The last "l" forces a
blank line between the heading text and the output on the report.
```

id-supp

suppresses the automatic display of item-ids in the leftmost column of an AQL report.

The equivalent option is (i.

Example

```
list entity phone (i
list entity id-supp phone
The above examples produce the same output.
```

if

see “with”

ifno

Syntax

ifno attr.name

in

throwaway connective.

is

throwaway connective.

istat

produces a graphic representation of item-id distribution within a file and suggests a new modulo.

Its counterpart is the "hash-test" command.

Syntax

istat file.reference {itemlist} {modlist} {(options)}

Options

* see "options: AQL".

r Provides additional information about indirect (pointer) items, which are otherwise not acurately reflected by the output.

s Suppresses display of histogram; displays summary statistics only.

Example

```
istat entity (s
file = entity  modulo= 11              16:08:40 12 Oct 1993
frames bytes items
          11
            Hash-Test/Istat Statistics
total item count =     8        byte count =       2226
 avg. bytes/item = 270.2  avg. items/group =          3
  std. deviation =   1.0  avg. bytes/group =      202.3
suggested modulo =   3.0
```

istat

produces a graphic representation of item-id distribution with a file and suggests a new modulo.

Syntax

istat file.reference {itemlist} {modlist} {(options)}

Options

* see "options: AQL".

r Provides additional information about indirect (pointer) items, which are otherwise not acurately reflected by the output.

s Suppresses display of histogram; displays summary statistics only.

items

throwaway connective.

k

suppresses the legend.

l

synonym for "list".

le

selects items whose specified attr.name contains a value less than or equal to the specified valuestring. The "<=" operator performs the same function as "le".

Syntax

with {not} attr.name le "valuestring"

Example

```
select invoices with amount.due le “0”
```

leg-supp

see legend suppress

legend-supp

suppresses the output of the "legend" on list and sort operations directed to the spooler.

The equivalent modifier is "leg-supp" and operator "k".

Example

```
list-item dm,bp, 'term-type' (kp
list-item dm,bp, leg-supp 'term-type' lptr
list-item dm,bp, 'term-type' legend-supp (p
The above examples are produce the same output.
```

list

generates formatted output based upon the specified arguments.

If an "itemlist" is specified, items are listed in order of the item-ids in the "itemlist". If an "itemlist" is not specified, items are listed by the groups into which they hash, and within each group, each item is listed in the order in which it was entered or most recently updated.

The report is either produced in a "columnar" or a "non-columnar" format. This is determined by the total width of all the attribute-defining items requested, plus the width of the item-id column. If the total of all these adi's (plus one space between each column) is less than the current device width (defined by the "term" command), the report is produced in columns. Otherwise, the report goes to the non-columnar output format.

Syntax

list file.reference {itemlist} {sellist} {modlist} {outlist} {(options)}

Options

* see "options: AQL".

Example

```
list entity
This lists the entity file displaying the attributes specified in the file's
macro or output-macro.
list entity with surname "[Brown]" name phone heading "'c'Brown
People In Entity'lcdxcjpll'" footing "'ll'Company Confidential"
ni-supp (pk
Produces the following output:
list entity with surname "[Brown]" name phone heading "'c'Brown
People In Entity'lcdxcjpll'" footing "'ll'Company Confidential"
ni-supp (pk
                   Brown People In Entity
                         <date>
<page>
<blank line>
<blank line>
entity.... name......................... phone.......
99999      Browning Joe                  818-555-1212
                                         213-555-1212
12345      Brown Sally                   619-555-1212
                                         714-555-1212
88822      Albrown Mike                  909-555-1212
:
<blank line>
<blank line>
Company Confidential
This list the surname and phone attributes of entity items which
contain the text "Brown" in the surname. The report has headings and footings
and is output to the printer without legends or the number of items message.
```

list-item

displays items in their internally stored format.

"list-item" combines the action of the "copy" verb with the selection criteria and format capabilities of the "list" verb.

The items are copied to the user's terminal or to the printer just as the "copy" verb would copy them, with the line numbers on the left margin.

Syntax

list-item file.reference {itemlist} {sellist} {seqlist} {modlist} {(options)}

Options

* see AQL options: h, i, k, n, and p.

f Starts display of each item on a new page.

s Suppresses listing of attribute (line) numbers.

Example

```
:list-item dm,bp, 'term-type' (p
```

list-label

produces one or more columns of output.

After pressing <return> to submit the AQL sentence for processing, a "?" prompt appears. This prompt is to supply the comma separated label output parameters:

columns,rows,skip,indent,size,space,c

columns the number of labels across the page.

rows the number of print lines on each label.

skip the number of lines (vertically) between labels.

indent the number of spaces to indent from left margin.

size the number of print positions per label.

space the number of spaces (horizontally) between labels.

c compresses null values.

If the "indent" parameter is not zero, this command prompts for entry of the "header" text for each print line requested. If any text is entered at any of these prompts, that text will appear before the corresponding print line on the left-most (1st) column only. Pressing <return> at any of these prompts indicates a null header.

The labels are printed in the same order as the items appear in the file. To change the ordering use the "sort-label" command.

Syntax

list-label file.reference {itemlist} {sellist} {outlist} {modlist} {(options)}

Example

```
:list-label entity contact company address csz (cip
?2,4,1,0,30,2,c
The label parameters shown define the output as having 2 labels
across, 4 lines each (contact, company, address & csz), skip 1
line, indent 0 (zero - assumes labels are aligned at the first print
position), 30 print positions each (longer strings are truncated to
30, in this case), space over 2 to the right between each label, and
"c" for "compress" nulls (don't allow blank lines within the label).
```

logical connectives

a brief discussion of the boolean operators "and" and "or" as used in AQL selection clauses.

Logical connectives are boolean operators used to connect two or more selection criteria clauses in an AQL sentence.

When two clauses are connected with an "or", either clause may be true for data to be selected for processing:

... with city "richmond" or with state "ca"

When an "and" clause connects two clauses, both clauses have to be true for data to be affected:

... with city "richmond" and with state "ca"

When the connective between two clauses is omitted, the default connective is an "or":

... with city "richmond" with state "ca"

The "and" clauses take precedence over "or" clauses, and a maximum of nine "and" clauses may be specified.

lptr

directs output of AQL sentences to the system printer, via the Spooler.

The output is directed according to the options most recently established with an "sp-assign" command.

The AQL sentence that created the output is automatically listed at the top of the first page of output, unless otherwise suppressed with the "tcl-supp" modifier.

The equivalent option is "p".

Example

```
list-item dm,bp, 'term-type' lptr a10
list entity "12345" phone (p
```

lt

"less than" operator.

ne

"Not equal to" operator.

ni-supp

suppresses the "number of items listed" message at the end of most AQL reports.

The equivalent options are "b" or "j".

Example

```
list entity phone ni-supp name
list entity phone name (j
list entity phone name (b
```

no

Reverses selection condition.

nopage

prevents terminal output from pausing at the end of each page.

"<ctrl>x", which terminates a report when the report is paused at a page break, serves no purpose when a report is being produced in "nopage" mode. It will not stop the report. The only way to stop it is to press the <break> key and "end"ing the process.

The equivalent option is "n".

Example

```
list md nopage
list md a1 (n
```

not

represents a "not equal to" condition. This reverses the "true" sense of the condition in a selection criteria. The "#", "no" and "ne" " operators perform the same function as "not".

Syntax

if not {each} attr.name {operator} {"value"}

Example

```
list jobs if not stat = "c"
list jobs if stat not "c"
```

nselect

selects items which would fail the selection criteria of a "select".

When an "output" list appears in a select-class statement, the list is built from the contents of the specified attribute(s).

Syntax

nselect file.reference {itemlist} {sellist} {outlist} {modlist}

Example

```
nselect md = "m]"
which is the same as:
select md # "m]"
The following example finds the FlashBASIC programs which did not
compile in the "bp" file:
select bp
[404] 78 items selected out of 78 items.
nselect dict bp
[404] 3 items selected out of 78 items.
Assume that there are two files with different number of items in each. Some
item-ids exist in both files, but not all of them. First, we select the first
file, then we select the items not in the second file:
select file1
[404] 767 items selected out of 767 items.
nselect file2
[404] 323 items selected.
The list contains the items in "file1" that are NOT in "file2".
```

of

throwaway connective.

only

ignores default attribute-defining items and displays only the item-id's from the given file.reference.

The "only" modifier MUST precede the file.reference.

Syntax

only file.reference

Example

```
list only entity
```

or

used between selection criteria clauses to indicate that either condition being evaluated as true will select the item for processing.

Syntax

with clause or with clause

Example

```
list entity with name "ar]" and with city "irv]" or with city "san]"
and with contact "[joe]"
This example lists those items which meet one of two criteria: a name
starting with "ar" and with a city starting with "irv", or, a city
starting with "san" and a contact containing the string "joe".
```

reformat

takes values from the attributes specified in the command and places them either in another file or to magnetic media in the order in which they are presented.

The specific items to be selected are included in the "itemlist".

If no "itemlist" is specified, all items in the file are selected in order by the group into which they hash, and within groups, in the order they were added to the file. The items are written to the new file or tape in the order in which they are selected.

The output specifications indicate those attributes that are to be selected.

The source item-id is selected first and becomes the destination item-id unless id-supp is specified. In that case, the first attribute value becomes the new item-id. Subsequent attributes are placed in the new item in the order they are specified.

"reformat" requests the destination file.reference. To send the output to a file, enter the file.reference. To send the output to magnetic tape, enter the word "tape". To write the output back onto the source file, press <return> or <enter>.

If the items are to be written back to the source file, an itemlist must be specified or a list must be active, otherwise, an infinite loop condition may result, in which items are continually added to the file!

When a file is reformatted to tape, the values are concatenated and either truncated or padded at the end with nulls (hexadecimal 00) to the record length specified by the most recently executed "t-att" verb. One tape record is written for each item.

item-ids are printed as items are dumped, unless the "id-supp" connective, or "i" option, is specified.

A tape label, which contains the file.reference, tape record length in hexadecimal, time, and date, is written on the tape first, unless the "hdr-supp" or "col-hdr-supp" connectives ("h" or "c" options) are specified. Two end-of-file (EOF) marks terminate the tape.

Any processing code that refers to a selected attribute is applied before the item is written to the new file or tape.

Syntax

reformat file.reference {sellist} {itemlist} outlist {modlist}

filename:destination-file or tape<return>

Options

see "options: AQL (AQL: Verbs)"

Example

```
reformat entity a0 name phone
File Name   :? entity-new
This takes all items from the "entity" file and writes them to the
"entity-new" file, using the temporary attribute-defining item
reference of "a0" to use the same item-id's in both files. The data
in "entity-new" will contain the "name" in attribute one and the
"phone" number in attribute two.
```

relational operators

used in the selection criteria process.

Each operator (sometimes called "logical") has an "English" equivalent which may be used interchangeably with the "symbol" form. When no operator is present, the default is an "equal to" (=).

Symbol Meaning

= or eq Equal to

#, ne,no or not Not equal to

> or gt or after Greater than

< or lt or before Less than

>= or ge Greater than or equal to

<= or le Less than or equal to

Syntax

with {not} attr.name operator "valuestring"

Example

```
list entity with state eq "ca"
list entity with amount.due not "0"
list entity with birthday before "1/2/62"
list entity with age gt "21"
list entity with no zip
list invoices with every amount > "0"
```

roll-on

creates a visual "break" in output data when the value of a specified attribute changes, an improvement over the "break-on" command.

"roll-on" is similar to the "break-on" connective, but has several "smarter" features.

The attribute values specified by the "roll-on" connective are "rolled" into the left adjacent column if data in that column has not been modified by a connective (excluding "break-on"), or is not a total column. This provides the ability to generate a roll-on value without occupying a column of output.

By comparison, to accomplish the same results using "break-on" requires building a "hidden" attribute-defining item -- one that allows "0" column positions on output. Even then, the break-on using the "hidden" field still occupies one column position on the report.

If the left adjacent column is a "total" column or has been modified, the right adjacent column is used if it satisfies the same criteria. If both the left and right adjacent columns have been modified or are "total" columns, a new column is created.

The new column heading is that of the lowest level "roll-on" connective and all attribute values specified in sequential "roll-on" connectives are "rolled" into this column.

If not specified, the number of "*'s" corresponding to the "roll-on" level, concatenated to the value rolled on is printed. A line is skipped only with the highest level "roll-on".

The "attr.name" is the name of the attribute on which to break. The "attr.name" designated by "roll-on" should also be specified as a sort key with "by" or "by-dsnd".

"text" is the text to be printed on the "roll-on" line.

Unlike the break-on connective, the roll-on value automatically appears at the detail break lines. By contrast, the break-on connective requires the use of the "'v'" option to accomplish the same result.

Syntax

roll-on attr.name {"{text} { 'options' } {text}"}

Options

b Outputs the value causing the break in either the "heading" or "footing" output field where the 'b' option is found in the heading or footing option string. It is not meaningful to specify this option within more than one "roll-on" specification.

d Suppresses the break data line if only one detail line has been output since the last control break.

n Resets the page counter to one on each break.

p Issues a form feed (page eject) after the data associated with this break has been output.

r Inhibits page "rollover". Outputs any occurrence of one or more control break lines at the end of the page, rather than at the top of the next page, thus forcing all the data associated with this break to be current on the same page.

u Underlines all total fields.

v Outputs the value causing the control break in the "roll-on" label. (see the "b" option in "break-on").

" Inserts a single quote in text.

Example

```
list orders.hist roll-on so# " 'p'" inv.line id-supp hdr-supp ni-supp
col-hdr-supp tcl-supp (kp
sort cust by territory by state by city entity.num ename roll-on territory
"'bp'" entity.address entity.zip phone# cgroup s.pcontact id-supp heading "* *
*   'b' Territory Dealer List * * *'cldcll'" tcl-supp (p
```

s-dump

writes items from a given file to the attached peripheral storage device in sorted order.

The "heading" or "header" modifier allows specifying the contents of the tape label written to the tape prior to the dump.

The tape must already be at load point in order to write the tape label.

If no selection criteria are specified, the entire file is written.

When saving to multiple reels, at the "insert" prompt (insert next reel and type "C" to continue), entering a "Q" will stop the process.

If the dump is being done to streaming cartridge tape (SCT), an explicit end-of-data sequence must be written at the end, since SCT cannot back up. The "t-weof" (TCL) command and the FlashBASIC "weof" statement both write the second eod needed to indicate end-of-file. Otherwise, "account-restore" or "sel-restore" fail to see the end of the tape.

Syntax

s-dump file.reference {itemlist} {seqlist} {sellist} {modlist} {heading "text"} {(options)}

Options

c Allows compatibility with R83 systems when dumping binary data.

h Suppresses the tape label.

i Suppresses the display of item-id's as they are dumped.

p Directs output to printer.

r{frame.size} This option is used to create dumps that are to be used on a system with a data frame size different from the system which created the tape. "frame.size" designates the "target" system frame size. See the table under "warnings" for various frame sizes

Example

```
s-dump entity by name
s-dump invoices by date by customer with date <= "1/1/92" heading
"Pre-1992 archived invoices"
```

sampling

limits the number of items processed by the AQL sentence to a specific number.

If selection criteria are included within the sentence, processing will continue until the specified number of items are found which match the selection criteria.

Syntax

sampling sample.number

Example

```
select entity sampling 10
[4042] 10 items selected out of 10 sampling items.
select entity with name "[mary]" sampling 10
[4042] 10 items selected out of 1513 sampling items.
In this example, 1513 items had to be sampled to return the 10 items
requested.
```

select

retrieves all items from the specified file.reference that meet the optional selection criteria.

The items are selected in the order of their appearance in the file.reference. The selected item may be saved for future use or acted upon immediately.

When an "output" list appears in a select-class statement, the list is built from the contents of the specified attribute(s).

If an itemlist is specified, items are selected in the order of the item-ids in the itemlist.

If no itemlist is specified, all items in the file are considered. In this case they appear in order by the group into which they hash, and within groups, in the order they were added to the file.

If output specifications are included, the values of the specified attributes are used to form the list. If omitted, the list is formed from the item-ids.

The list is a temporary list that is used by the next verb as its implicit itemlist. The list can be permanently saved by using the "save-list" or "sl" verb. The list can also be passed to an OP command or to a FlashBASIC program if the next TCL command executes that program.

The elements of the list may be used as item-ids to reference data in any file, not just the file referenced in the select command. For example, if a select on one file is followed by a list on a different file, the list of item-ids generated by the select are used as an itemlist in the list command.

The "outlist" parameter specifies the attribute-defining item{s} from which values are to be extracted for creation of the list. (See example 2.) This is somewhat similar to using "qselect", with the additional provision for being able to specifiy collation order and selection criteria.

Syntax

select file.reference {itemlist} {sellist} {outlist} {modlist}

Options

j Suppresses all messages.

q Bypass index selection.

s Activates a "secondary" list.

Example

```
select invoices with amount.due > "0"
This builds a list of the item-ids which match the given selection
criteria.
select customers with last.order.amount > "1000" order.pointers
This creates a non-sorted list of the contents of the
"order.pointers" attribute.
sselect invoices = 'd]' with invoice.date ge "1/1/94" by date
This creates a sorted list of all invoices whose item-id begins with
the letter "d" and with an "invoice.date" on or after January 1,
1994.
```

si

Invokes "sort-item" command.

sort

generates sorted and formatted output of selected items and attributes from a specified file.

The output produced by sort is identical to the output produced by list when the "by" modifier is specified.

Syntax

sort file.reference {itemlist} {seqlist} {sellist} {modlist} {outlist} {print limiters} {(options)}

Options

* see "options: AQL".

Example

```
sort entity by company by name with name "PICK]" company name id-supp (p
Prints a list of entity items sorted by company and name, whose name starts
with "PICK". The company and name are printed, and the entity item-id is not
printed.
```

sort-item

displays items in their internally stored format, in order of the given sort-key(s).

"sort-item" combines the action of the "copy" verb with the selection criteria and format capabilities of the "sort" verb.

The items are copied to the user's terminal or to the printer just as the "copy" verb would copy them, with the line numbers on the left margin.

Syntax

sort-item file.reference {itemlist} {seqlist} {sellist} {modlist} {(options)}

Options

* see AQL options: h, i, k, n, p.

f Starts display of each item on a new page.

s Suppresses listing of attribute numbers.

Example

```
sort-item dict entity by ac
```

sort-label

produces one or more columns of output in order of the specified sort-key(s).

Outputs formatted labels or reports in columnar format from the specified file.reference, sequenced by one or more sort-key(s).

After pressing <return> to submit the AQL sentence for processing, a "?" prompt appears. This prompt is to supply the label output parameters:

columns The number of labels across page.

rows Number of print lines on each label.

skip Number of lines (vertically) between labels.

indent The number of spaces to indent from left the margin.

size The number of print positions per label.

space The number of spaces (horizontally) between labels.

c Compresses null values.

Each parameter must be separated by a comma.

If the "indent" parameter is not zero, this command prompts for entry of the "header" text for each print line requested. If any text is entered at any of these prompts, that text will appear before the corresponding print line on the left-most (1st) column only. Pressing <return> at any of these prompts indicates a null header.

Syntax

sort-label file.reference {itemlist} {sellist} {seqlist} {outlist} {modlist} {(options)}

Options

* see "options: AQL".

Example

```
:sort-label entity by zip contact company address csz (cip
?2,4,1,0,30,2,c
The label parameters shown define the output as having 2 labels
across, 4 lines each (contact, company, address & csz), skip 1
line, indent 0 (zero - assumes labels are mounted at first print
position), 30 print positions each (longer strings are truncated to
30, in this case), space over 2 to the right between each label, and
"c" for "compress" nulls (don't allow blank lines within the label).
```

sreformat

moves the sorted values from the attributes specified in the AQL sentence and places them either in another file or to tape in the order of the specified sort key(s) (specified in "seqlist").

Each entry in the outlist becomes an attribute in the destination item, in the same order as in the outlist. The source item-id is selected first and becomes the destination item-id, unless "id-supp" (or an "i" option) is specified. In this case, the first attribute value becomes the new item-id.

Upon execution of this command, the prompt, "file name:", appears on the screen. Either a file.reference may be entered, which directs the output to the specified file.reference, or the literal, "tape", which directs the output to the attached magnetic tape.

The output specifications indicate those attributes that are to be selected.

The attributes are written to the new file or tape in the order they are selected.

To write the output back onto the source file, press <return> or <enter>. If the items are to be written back to the source file, an item list must be specified or a list must be active. Otherwise, an infinite loop condition may result, in which items are continually added to the file!

When a file is reformatted to tape, the values are concatenated and either truncated or padded at the end with nulls (hexadecimal 00) to the record length specified by the most recently executed "t-att" verb. One tape record is written for each item.

The item-ids are printed as items are not printed as the items are transferred.

A tape label, containing the file name, tape record length in hexadecimal, time, and date, is written first, unless the "hdr-supp" or "col-hdr-supp" connectives ("h" or "c" options) are specified. Two end-of-file (eof) marks terminate the tape.

Any processing code (or correlative) that refers to a selected attribute is applied before the item is written to the destination file or tape.

Syntax

sreformat file.reference {sellist} {itemlist} {outlist} {seqlist} {modlist} {options}

filename:{destination-file | tape }

Options

* see "options: AQL"

Example

```
:sreformat sls by region region total amount (d
filename:temp-sales
```

ss

produces columnar and cross totals on rows of designated attributes within a given range of dates.

attr.name1 is the attribute-defining item containing a date which limits the tabulation. The output form of this attribute determines the column headings. The output form of "attr.name1" does not have to be the same as the values included in "beg.date" and "end.date". For example, "attr.name1" may display the month, while "beg.date" and "end.date" must be in the form "mm/dd/yy" enclosed in double quotes (").

beg.date is the beginning date range to be included. If not specified, "beg.date" is determined by the number of columns that will physically fit on the display or printout.

end.date is the ending date range to be included. If not specified, the current system date is used as the default.

attr.name2 is the attribute that contains the values to be columnar and cross (row) totaled.

If the width of the report exceeds the width of the output device, the extra columns are truncated.

If two or more "ss" connectives are used in the same AQL sentence, a single report is generated, with columns for each subsequent "ss" connective following the totals column for the previous connective.

Column headings are created for each possible value produced by "attr.name1" within the "beg.date" and "end.date" range. Output-conversions are processed before producing the heading. Correlatives are not processed. The format of the column is determined by "attr.name2".

Each cell in the listing contains the total value of "attr.name2" for the date specified in that column heading.

"roll-on" can be used in conjunction with "ss" to produce subtotals by specified categories. Each rolled attribute value produces a row in the output. If no "roll-on" is specified, only a total line is produced.

The granuality of the date display is determined by the output-conversion of the "attr.name1". The date granuality is the value of the last multiply in the output-conversion. If the following output-conversion is used to provide week ending dates, the report will have a granuality of 7 days and the dates will be the Saturday after the date in the item. In both of the following examples, the date is stored in attribute 4.

a4/'7'*'7'+'6'

d2/

Whereas the following output-correlative makes a granuality of 3 days:

a4/'3'*'3'

d2/

The number of dates to display is based upon the display width of the "attr.name2" attribute, but the width of the columns on the report is based upon the greater of the "attr.name1" display width and the "attr.name2" display width.

Syntax

ss attr.name1 {{beg.date} {end.date}} attr.name2 {(g}

Options

g Suppresses the row and column totals.

Example

```
sort invoices with code "c" "d" and with amount ne "" by code by
category ss quarter "01/01/92" "12/31/92" amount roll-on code roll-on
category "'d'" det-supp id-supp
The previous example produces a report with subtotals for each "code"
and totals to each category within the code for the quarters "1-92",
"2-92", "3-92" and "4-92".
sort invoices by customer ss quarter "06/30/92" amount roll-on
customer det-supp id-supp
This produces a report by customer for three quarters, "4-91",
"1-92", and "2-92". The beginning date range is calculated based on
the attribute's width, the output device's width, and the ending date
of "06/30/92".
```

sselect

creates a sorted list of items from the specified file.reference that meet selection criteria.

The list is a temporary list that is used by the next verb as its implicit item.list. The list can be permanently saved by the "save-list" verb. The list can also be passed to an OP command or to a FlashBASIC program if the next TCL command executes that program.

When an "outlist" appears in a select-class statement, the list is built from the contents of the specified attribute(s).

The elements of the list may be used as item-ids to reference data in any file, not just the file referenced in the sselect command. For example, if a "sselect" on one file is followed by a "list" on a different file, the list of item-ids generated by the sselect are used as an item.list in the "list".

In the special case where the "sselect" verb is used with a "by-exp" or "by-dsnd-exp" modifer, the elements of the list are multi-valued, where the first value is the selected data, and the second value is the three-digit value count. In FlashBASIC, the two values are retrieved by the use of the "readnext id,vc" form of the "readnext" statement.

Syntax

sselect file.reference {itemlist} {sellist} {seqlist} {outlist} {modlist}

Options

j Suppresses all messages. ("n" items selected, etc.)

s Activates a "secondary" list.

w Bypasses index selection.

Example

```
sselect invoices with amount.due > "0" and with no payment
Creates a list of invoice item0ids which have an amount.due > o,
and which have not made a payment. The list is sorted by invoice
item-id.
sselect entity by name with name = "PICK]" name
Creates a list of entity names which start with the string "PICK"
sorted in name order.
```

stat

outputs the total number of items processed and an average value of all items that meet the specified selection criteria.

Syntax

stat file.reference {itemlist} {sellist} {modlist} {attr.name} {(options)}

Options

* see "options: AQL".

Example

```
:stat invoices with date > "1/1/92" amount
Total = 73970 average =  1946.57 count = 38
```

sum

produces a numeric total of a given attribute for all items meeting the optional selection criteria.

If an "attr.name" (attribute name or number) is not specified, the total number of characters in the selected items are calculated.

If only the file.reference is specified, the total number of characters in the entire file is calculated, including attribute and value marks.

Syntax

sum file.reference {itemlist} attr.name {sellist} {modlist} {(options)}

Options

* see "options: AQL".

Example

```
sum invoices amount (p
```

supp

suppresses default page heading.

t-dump

copies items from a given file to the attached peripheral storage device.

The "heading" modifier allows specifying the contents of the tape label written to the tape prior to the dump.

The tape must already be at load point in order to write the tape label.

If no selection criteria are specified, the entire file is written.

In all multi-reel operations, if the process detects end of media before completing the dump, it prompts for the next reel with the message:

Load volume #X and type 'C'

label 08:00:00 16 Jan 1991 ... #

When the next "reel" is inserted or mounted, "c" continues the process, or "q" stops it and returns control to TCL.

If the dump is being done to streaming cartridge tape (SCT), an explicit end-of-data sequence must be written at the end since SCT cannot back up. The "t-weof" (TCL) command or the FlashBASIC "weof" statement may be used to write the second eod needed to indicate end-of-file. Otherwise, "account-restore" or "sel-restore" may fail to see the end of the tape.

Syntax

t-dump file.reference {itemlist} {sellist} {modlist} {heading "text"/options} {(options)}

Options

a Make pre-D3 compatible tapes.

c Allows compatibility with R83 systems when dumping binary data

h Suppresses the tape label.

i Suppresses the display of item-id's as they are dumped.

p Directs output to the printer, via the Spooler.

r{frame.size} This option is used to create dumps that are to be used on a system with a data frame size different from the system which created the tape. "frame.size" designates the "target" system frame size. See the chart listed under warnings for the current D3 frame sizes.

Example

```
t-dump entity
t-dump invoices with date <= "1/1/92" heading "Pre-1992 archived
invoices"
```

t-load

restores files from a tape that was generated by "t-dump".

If a selection criteria is not specified, the entire file is loaded.

If multiple "t-dumps" were performed, then multiple "t-load" are required to restore the data.

When saving to multiple reels, at the "insert" prompt (insert next reel and type "C" to continue) you may also type "Q" to quit.

"t-load" reads the tape label in order to get the tape block size. If the tape is unlabeled, "t-load" reads the first tape block, determines that the tape is unlabeled, and backspaces the tape by one block before continuing. In this case, the appropriate tape block size must have been specified using the "t-att" verb.

Syntax

t-load file.reference {itemlist} {sellist} {modlist} {(options)}

Options

a Bypasses tape label handling and assumes that media is positioned at the beginning of the desired data area.

i Suppresses display of item-ids.

m Suppresses display of error message 223 when the item is already on file and the "o" option is not used.

o Overwrites existing items, when duplicate item-id{s} exist.

p Directs output to printer, via the Spooler.

r{n} Loads binary items (which require contiguous frames) from "t-dump"'s made on a system with "n" data bytes per frame.

v Reads items but does not write them into the file system. When used with the "o" option, it will write the data to the file.

Example

```
t-load dict entity
t-load entity (o
```

tape

retrieves data from the currently attached magnetic media, rather than the actual data level of the file.

The tape must have been created in a "t-dump" format and must be positioned to the beginning of the appropriate tape file. The attribute-defining items specified in the AQL sentence are retrieved from the dictionary level of the specified filename.

Example

```
list invoices with date >= "1/1/91" and <= "12/31/91" tape
This example assumes that the appropriate ("t-dump") media has been
attached and previously positioned at the proper tape file.
```

tcl-supp

suppressses the display of the AQL (TCL) sentence used to generate the report on printed output only.

Example

```
list entity with phone "714]" tcl-supp lptr
```

the

throwaway connective.

throwaway connectives

class of AQL modifiers which have no effect on command execution.

These modifiers allow use of a syntax closer to a natural, human, language.

These words reside in the md. The normal set includes: "a", "data", "file", "of", "an", "for", "the", "any", "are", "in" and "items".

The list of available throwaway connectives may easily be expanded by copying any one of the above items in the md to new items. This makes tailoring vocabularies to different languages easier.

Any item in the md with "cz" in attribute 1 is a "throwaway" connective.

Example

```
The following example creates new throwaway connectives,
"displaying", "showing", and "field".
copy md a a a<return>
to:displaying showing field<return>
After adding the new items, the following sentence is now possible:
sort the customer file showing the name field<return>
Where, "the", "file", "showing", and "field" are all ignored, but
would provide for a more grammatically correct sentence and is the
same as:
sort customer name
```

total

accumulates a columnar total for a specified attribute-defining item.

The optional "total-limiter" functions like the "value string" portion of the selection criteria process, limiting totals to values that pass the specified selection criteria.

"attr.name" is the name of the attribute whose values are to be totaled.

"operator" is any legal relational operator, such as "=". If no operator is specified, "=" (equal) is assumed.

"value.list" is a list of values to match, enclosed in (") (double quotes) or "\" (backslashes).

If the "total" modifier is used in the same sentence as a "roll-on", subtotal values are generated whenever a break occurs.

The subtotal is displayed in the appropriate column for each attribute specified in a "total" modifier. At the end of the listing, a total is generated for each column.

In computing the value for accumulation, correlatives (attribute 8) are processed, but conversions (attribute 7) are applied to the totaled value.

Syntax

total attr.name {"total-limiter"}

total attr.name {{operator} "value.list"

Example

```
list invoices total amount total amount (p
list invoices total status "overdue"
list invoices total amount le "300" total amount gt "1000"
```

total-on

see "roll-on".

u1070

returns a checksum for the current item.

u201e

returns the number of attributes in an item.

u2070

returns the checksum value for the specified frame.

u3070

returns the PCB fid for the current process as 4 hexadecimal digits.

u5070

converts a string to the form:

string[1,3]: "-000"

It strips all but the first three characters, and then appends the literal '-000' to the end.

If the string is less than three characters long, then it is returned untouched.

Syntax

u5070,string

Example

```
string = "abcdef"
u5070,string
Returns "abc-00".
```

u508e

calls the "copy" processor from AQL.

Headings and footings may be included.

This is the same as "list-item" or "sort-item".

u6070

inputs up to 100 characters.

A carriage return is output immediately after the ":" (prompt character), which displays as a request for input. The resulting input is typed in underneath the column specified by the User Exit.

A carriage return is also output immediately after the data is entered.

u7070

inputs up to 100 characters.

A carriage return is output immediately after the user completes their data entry.

u8070

returns a checksum for the current item

u9070

toggles the Page/NoPage format.

This is the same as activating the AQL "n" option.

ua070

returns the specified value from a multi-valued set.

The "value.count" (the "n"th value to return) follows the attribute-defining item reference in the AQL sentence.

Example
```
list file.name attribute.definition.item "value.count"
```

uc070

returns the specified value from a multi-valued set.

"n" is the value count to return.

Syntax
uc070;n

Example
```
uc070;3
Returns the 3rd multi-value.
```

using

allows AQL to report on the data in a given file using attribute-defining items from a different file.

Only one "using" connective is allowed in a TCL statement.

Syntax
using {dict} file.reference

Example
```
list customers using dict suppliers name phone
In this example, the attribute-defining items "name" and "phone" are
extracted from the dictionary of the "suppliers" file, and applied to
items in the "customers" file.
```

with

marks the beginning of a selection processor conditional, which is processed against the associated file to select items to pass on to the LIST processor.

The "not" (or "no") modifier negates the normal effect of selection -- only those item that do NOT meet the criteria are selected.

"value" is a specific value to match, enclosed in double quotes (") or backslashes (\). If a selection criterion does not include a value, all items that have at least one value for the specified attribute name are selected.

"every" (or "each") specifies that in a multi-valued attribute, every value for the attribute must meet the specified condition.

"attr.name" is the name of an attribute as specified by the attribute-defining item in the dictionary.

"operator" is any legal relational operator, such as "=".

Syntax
with {not} {each} attr.name {operator} {"value"}

Example
```
list entity with contact
list entity with no contact and with phone
list invoices if amount.due > "100"
list entity if no phone.number
list entity with no contact and no phone.number
list orders if every quantity.ordered > "10"
```

within

retrieves a list of all the items which are sub-items of a specified item.

This is used to explode one attribute containing one or more multi-values that are item-ids within the same file, like in bill-of-material explosions. The file-defining item ("d-pointer") must contain a "v" code in attribute 8 in the form: v;;n where "n" indicates the attribute count to explode. The explosion may proceed up to 20 sub-levels.

Syntax

list within file.reference 'item-id' {(options)}

count within file.reference 'item-id' {(options)}

Example

```
Suppose that in the dictionary of the hypothetical file, "parts",
that the "correlative" attribute of the "d-pointer" to "parts"
contains the processing code, "v;;1", which means to explode on
attribute 1.
The sentence:
list within parts 'car'
Produces the following output:
level    Parts......
1        car
2        engine
2        wheel
3        tire
3        rim
2        hood
[405] 6 items listed out of 1 items.
list parts
parts......
wheel
tire
rim
engine
hood
car
[405] 6 items listed out of 6 items.
The hypothetical data could be examined with the command:
ct parts car
Which produces the following output:
car
001 engine]wheel]hood
ct parts engine
engine
001 flywheel]piston
"flywheel" and "piston" items don't exist in "parts", so they don't get
exploded.
ct parts wheel
wheel
001 tire]rim
ct parts tire rim hood
tire          rim           hood
001           001           001
002 $35.00    002 $60.00    002 $70.00
```

without

negates the normal rules for selection criteria.

"ifno", "with no" or "with not" are identical to "without".

Syntax

without attr.name

Example

```
list entity without phone
This is essentially the same as saying "with no phone". It could also
be stated as:
list entity with phone = ""
```

Attribute-defining Items

describe, format, and manipulate data contained within files.

Attribute-defining items (ADI's) are located in a file's dictionary or in the account's master dictionary. ADIs are used by nearly all D3 Processors to create data interrelationships, format data, or select data.

Each ADI in the dictionary has its name as its item-id. There may be several attribute-defining items that refer to the same attribute in an associated data file item. This typically is done when an expanded name is needed for clarity and also when different views are needed for the same data element. There is no limit to the number of attribute-defining items that can point to a specific attribute. The definition of the attributes and the attribute's master dictionary ADI are as follows:

Attribute 0: Item-Id

Attribute 0 is the item-id (or name) of the ADI item in the dictionary. For example, the item "a/amc" usually found in an account's master dictionary defines the second attribute of an ADI.

Attribute 1: Dictionary Code (d/code)

Attribute 1 contains an "a", "s", or "x" to indicate that the item is an attribute-defining item. "a" is used to indicate the "base" attribute definition, "s" is used to define surrogate or redefinitions of an attribute, and "x" is used to define "protected" attribute definitions. By convention, there is usually only one "a" ADI per attribute used to define the each attribute, all other ADIs for the same attribute is given an "s" or "x" designation. In D3, an optional description or text string may follow the "a", "x" or "s" without affecting any other normal functions.

Default attribute items have numeric item-ids starting with 1. These item-ids are used by AQL verbs as output specifications if no other output specifications are given. The numeric item-ids must be consecutive; that is, in order to have the third attribute list by default, attribute items 1 and 2 must exist, even if they are not needed for the listing. The attribute items for attributes that are not needed for listings can be given a d/code of "x".

Synonym-defining items (d/code = "s") behave just like attribute-defining items (d/code = "a"). They are used when there are multiple attribute-defining items for the same attribute. (Multiple attribute-defining items for the same attribute allow applications to process the same data in different ways.) By convention, the primary attribute-defining item is created with a d/code = "a". Any other attribute-defining items for the same attribute are created with a d/code = "s".

Attribute 2: Attribute Mark Count (a/amc)

The number stored in this attribute is the attribute number where the defined attribute values are stored in each item. The first attribute in an item has an AMC of 1, the nth attribute has a AMC of n. It is used to physically locate the attribute values within the item of the associated file.

Attribute 3: Substitute Header (sub.header)

Attribute 3 is used to specify report column headings in the associated file. If a subheader is not specified, the attribute name is used as the column heading. If the column width specified in attribute 10 is not wide enough to display the heading the column is expanded to allow space for the column heading. A "\" can be placed in attribute 3 to print all periods in the column header. The subheader may be multi-valued to output a multi-line heading.

Attribute 4: Structure (structure)

Attribute 4 is used to establish internal relationships within an item. The relationship between controlling and dependent attributes is such that a value must be entered into a controlling attribute before the Update processor will allow entry of values into dependent attributes. Each multi-value for the controlling attribute controls the corresponding values in the dependent attribute. The codes "c" or "d" are used to specify controlling or dependent attributes respectively. The format for usage of these codes is:

c;attr#;attr#;...

d;attr#

Any attribute specified by a "c" as controlled by that attribute must also have an attribute-defining item with attribute 4 specified by a "d;attr#", where "attr#" is the attribute number of the controlling attribute.

The attributes dependent upon the controlling attribute must contain a "d" (dependent) code in attribute 4.

In an AQL report, to see dependent attributes, controlling attributes must be specified.

The controlling form "c;" (controlling with no dependents) is valid and can be used to force the Update processor to display the substitute-header above the field value.

In the Update processor, to see dependent attributes, controlling attributes must be specified.

Attribute 5: Reserved and Unavailable

Attribute 6: Reserved and Unavailable

Attribute 7: Output-Conversion

Attribute 7 contains processing codes which prior to output, modify non-null data contained within the referenced attribute. Processing codes are described in detail as individual tokens. The Processing Codes are used to perform data manipulations which often remove the need for a program.

Attribute 8: Correlative (correlative)

Attribute 8 contains processing codes similar to the output-conversion attribute, except the codes specified within this attribute are used to preprocess data prior to sorting or selecting in addition to display.

Attribute 9: Attribute type (justification)

Attribute 9 contains codes used to indicate the type of justification used on the attribute. See the subjects, "attribute-type" and "master dictionary" for details about this field.

Attribute 10: Column Width (column-width)

The column width attribute is used to define the minimum number of character spaces to be allocated for displaying the data within the attribute on AQL reports. The actual column width used is dependent upon the column width attribute, the subsitute column heading attrbute, the item-id, the justification attribute, and the width of the display.

Attribute 11: Reserved and Unavailable

Attribute 12: Reserved and Unavailable

Attribute 13: Reserved and Unavailable

Attribute 14: Input-Conversion

Attribute 14 contains processing codes applied to data immediately after entry. These codes can be used for data validation as well as transformation. These codes are executed even if there is a null entry.

Attribute 15: Macro (macro)

The attribute names contained in this attribute are passed as an attribute name list to the UP when zooming to another item. The values named in this attribute are separated by spaces, not punctuation.

Attribute 16: Reserved and Unavailable

Attribute 17: Description (description)

This attribute is reserved for comments and descriptions concerning the function of the current attribute-defining item. The text in this attribute is displayed as the UP help message when a "?" is entered as the only data to a UP prompt.

Attribute 18: Reserved and Unavailable

Attribute 19: Reserved and Unavailable

Attributes 20 to 30: Hot Keys (hotkey.all, hotkey0, ... hotkey9)

Eleven hot keys are available to provide quick access to FlashBASIC programs from within the Update processor. The Hot Keys are defined in a file-defining item (FDI) and attribute-defining item (ADI). The value of these keys is specified in attributes 20 to 30 in the FDI or ADI. Attribute 20 is hotkey.all, attribute 21 is hotkey1, and attribute 30 is hotkey0. A user selects which Hot Key they wish to use by typing <control>+x#, where # is a Hot Key number (0-9). Whenever a Hot Key is selected one Hot Key entry is used based upon the following search:

If the ADI has a hotkey.all entry, it is used.

If the ADI has a Hot Key # defined, where # is the number entered by the user, that Hot Key entry is used.

If the FDI has a hotkey.all entry, it is used.

If the FDI has a Hot Key # defined, where # is the number entered by the user, that Hot Key entry is used.

Only the first Hot Key entry found in the above search sequence is used.

No parameters can be passed to the FlashBASIC subroutine called from a Hot Key.

Example

```
The next two processing codes have a controlling dependent
relationship.
dictionary-code        A
attribute-count        2
substitute-header
structure              C;3
output-conversion
correlative
attribute-type         L
column-width           10
input-conversion
macro
output-macro
description            This attribute controls attribute 3.

dictionary-code        A
```

```
attribute-count        3
substitute-header
structure              D;2
output-conversion
correlative
attribute-type         lw
column-width           10
input-conversion
macro
output-macro
description            Dependent upon attribute 2.

The following ADI can be placed in a master dictionary and used for general
purpose counting when you want to know the number of items matching a specific
criteria.
item id:               tally
dictionary-code        A
attribute-count        999
substitute-header
structure
output-conversion      mrz
correlative            a"1"
attribute-type         R
column-width           4
input-conversion
macro
output-macro
description       Provide a constant value of 1 for commands
                   "total tally det-supp" which shows the number of
                   items from each break-on.
hotkey.all
```

default attribute items

attribute-defining items that may reside in the dictionary of any file. If no attribute-defining items are explicitly requested in an AQL report, the "default" fields display automatically.

Default attribute items have numeric item-ids starting with 1. These item-ids are used by AQL verbs as output specifications if no other output specifications are given, including a defined output-macro on the file-defining item.

The numeric item-ids must be consecutive; that is, in order to have the third attribute list by default, attribute items 1 and 2 must exist, even if they are not needed for the listing. The attribute items for attributes that are not needed for listings can be given a d/code of "x".

*a0-13

default attribute-defining items provided automatically with each account.

There are 14 of these items: *a0, *a1, *a2, *a3 ... *a13. The number in the item-id corresponds to the attribute mark to which it points.

The "*" is optional in D3, see 'temporary attribute items' for more information.

Syntax

*anumber {*anumber...}

Example

```
list md with *a1 = "d]"
list mds,, with a1 = "d]" and with a8 = "sys]"
```

***a9998**

a special attribute identifier used to produce a sequential counter for items being processed by the AQL sentence.

Example

```
list entity *a9998 name (i
```

***A9999**

a special attribute-defining item provided with each account. Its function is to return the size, in bytes, of the item being processed by AQL.

A synonym item called "size" performs the same function as the "*a9999" item.

Also see the "a" processing code operand "ni".

Example

```
list entity total *a9999
```

a/amc

references the attribute-count, usually of an MD or a DICT item.

Example

```
list md with a/amc "d]"
```

a9998

See "temporary attribute items".

a9999

See "temporary attribute items".

ac

references the attribute-count (attribute 2) of the dictionary item being processed by AQL.

account

displays the account of the file being pointed to if the item currently being processed by AQL is a q-pointer.

Example

```
list mds account
```

an

references the item-id of the item being processed by AQL.

at

displays the justification specification of the dictionary item being processed by AQL. See "attribute-type".

attr.code

displays the attribute code of the dictionary item being processed by AQL:

a - attribute-defining item

s - synonym attribute-defining item

attr.count

used to reference the attribute-count of the dictionary item being processed by AQL.

Example

```
list md attr.count
```

attr.name

is a specific reference to an attribute-defining item in the dictionary of a given "file.reference".

Example

```
list entity company.name contact
In this example, "company.name" and "contact" are "attr.names".
```

attr.type

displays the justification specification of the dictionary item being processed by AQL. See "attribute-type".

attribute-count

used to reference the attribute-count of the dictionary item being processed by AQL.

attribute-name

used to reference the item-id of the item being processed by AQL.

attribute-type

displays the justification specification of the dictionary item being processed by AQL:

l{x} Left justification. Text breaks at width of column.

r{x} Right justification. This will overlay the column(s) to the left if the data is longer than the column width.

t{x} Text justification. This behaves like left justification, but will break on a space if the data is longer than the column width.

u "Unconditional" Left justification. This will overlay the column(s) to the right if the data is longer than the column width.

w Processes through OP before listing. Any valid OP command may follow this code or be used in the attribute. This only works in non-columnar formats. The "w" type can also be used by the Update processor. See the "access" function number 22.

ww When used from an AQL report, this attribute-type processes all attributes from this attribute through the end of the item through OP before listing. This only works in non-columnar format. This also forces a form-feed after processing the last attribute of the item. When used from the Update processor, this attribute-type indicates that any attributes to follow will appear in free-text mode.

x Used with l, r, or t to expand the display field to fill the report display.

base

displays the beginning frame-id (fid) of a series of logically contiguous frames containing the primary file space.

c/acct

displays attribute 8 which is the account on which a binary item was last compiled.

This is used by "list-obj" and AQL.

Example

```
list dict bp c/acct
Page   1     DICT bp
DICT bp........ account.............
test           dm
test2          personnel
```

c/bytes

displays attribute 11 of compiled binary items through the list-obj TCL command. This is the number of bytes of D^3 object code. It is also available to AQL.

This attribute holds the number of bytes of the actual compiled object code within the frame(s) that hold the binary item. This information is useful to estimate the number of frames that will be necessary to FlashBASIC compile standard D^3 object code. This is typically 4 times the size in bytes, but is dependent on implementation, optimization level and related options, like "f".

This byte count does not include the symbol table, FlashBASIC code or the size of the compile date-time stamp.

Example

```
list dict bp c/bytes
Page   1     DICT bp
DICT bp........ bytes..
test           234
test2          439
```

c/code

displays attribute one of a binary item which is the "cc" code. It is available to AQL and list-obj.

Example

```
list dict bp c/code
Page   1     DICT bp
DICT bp........ code
bp             D
test           CC
test2          CC
sort dict bp with c/code = "cc" c/code
Page   1     DICT bp
DICT bp........ code
test           CC
test2          CC
Displays only the items in the file that are binary.
```

c/date

displays attribute 4 which is the date a binary item was compiled.

The date is stored in internal format and displayed in external format.

Example

```
list dict bp c/date
Page   1     DICT bp
DICT bp........ date.....
test           09/03/93
test2          09/04/93
```

c/fid

displays attribute 2 which is the starting frame of binary items.

Example

```
list dict bp c/fid
Page   1     DICT bp
DICT bp........ start fid
test           447810
test2          447816
```

c/flash

displays the number of frames of FlashBASIC code of binary items.

Example

```
list dict bp c/flash
Page   1     DICT bp
DICT bp........ flash.sz
test                0
test2              12
Notice "test" was not FlashBASIC compiled but "test2" was.
```

c/options

displays the compile options used in to compile a program.

This attribute is stored alphabetized.

Example

```
list dict bp c/options
Page   1     DICT bp
DICT bp........ c/options
bp             16
test           D
test2          OY
Notice that "test" was compiled with the "d" option which inhibits break.
"test2" was FlashBASIC compiled with the "o" option and the "y" allows
multiple concurrent FlashBASIC copies.
```

c/port

displays the port which last compile the program.

This is used by "list-obj" and AQL.

Example

```
list dict bp c/port
Page   1     DICT bp
DICT bp........ port
test              2
test2            41
```

c/release

displays the release when a binary item was compiled.

Example

```
list dict bp c/release
Page   1     DICT bp
DICT bp........ release..............
test              6.1.0.M10.A5.D4.F1
test2             6.1.0.M10.A5.D4.F1
M10 is the monitor version 10, A5 is the ABS version 5, D4 is the Datafiles
version 4, and F1 is the FlashBasic version 1.
```

c/size

displays the number of D3 frames of object code.

This number does not include the number of FlashBASIC object code frames. See "c/total".

Example

```
list dict bp c/size c/flash c/total
Page   1     DICT bp
DICT bp........ pick.sz flash.sz frames
test                   1        0     1
test2                  1        4     5*
Notice the "*" indicates "test2" has been compiled with FlashBASIC.
```

c/time

displays the time when a binary item was compiled.

This is stored in internal format and displayed in external format.

Example

```
list dict bp c/time
Page   1     DICT bp
DICT bp........ time.....
test            11:43:01
test2           15:01:32
```

c/user

displays the user that last compiled the binary item.

Example

```
list dict bp c/user
Page   1     DICT bp
DICT bp........ user......
test            bob
test2           sally
```

col.width

determines number of column positions.

column-width

references the initial width for the display column allocated by AQL for displaying the associated data.

Its initial value can be adjusted by AQL depending on the type-values used in 'attribute-type'. If null, the default column-width used is based upon the length of the attribute's id, which is superceded still by the length of a 'substitute-header', if present.

cor

references attribute 8 of the dictionary item being processed by AQL.

cw

references the number of columns allocated for the dictionary item being processed by AQL.

d/code

references attribute 1 of the dictionary item being processed by AQL and describes the type of dictionary item.

"d/codes" of "d", "dx", "dy", "dl" and "dp" are used by file-defining items, while d/codes of "a", "s", or "x" are used by attribute-defining items.

d/code1

references the first letter of attribute 1 of the dictionary item being processed by AQL.

d/size

returns the size (in bytes) of each item processed by AQL.

dc

references attribute 1 of the dictionary item being processed by AQL and describes the type of dictionary item.

description

references the nature and usage of the attribute-defining item being processed by AQL, attribute 17.

dict.code

references attribute 1 of the dictionary item being processed by AQL and describes the type of dictionary item.

dictionary-code

references attribute 1 of the dictionary item being processed by AQL and describes the type of dictionary item.

D/codes of D, DX, DY, DL and DP are used by file-defining items, while d/codes of A, S and X are used by attribute-defining items.

dsc

references the nature and usage of the attribute-defining item being processed by AQL.

f/base

displays the beginning frame-id (fid) of a series of logically contiguous frames containing the primary file space.

f/mod

references attribute 3 of the file-defining item being processed by AQL and describes the modulo of that file.

f/realloc

references attribute 13 of the file-defining item being processed by AQL and describes the reallocation modulo for that file, if any.

f/sep

references attribute 4 of the file-defining item being processed by AQL and describes the separation of that file, if any.

f/syn

references attribute 12 of the file-defining item being processed by AQL and describes synonyms for that file.

file.code

references attribute 1 of the dictionary item being processed by AQL and describes the type of dictionary item.

File-defining items (in an account) use the following d/codes:
D{L P S X Y} (data file)
DC (Pick/BASIC program file - obsolete)
M,N (macro)
ME (menu)
P (UP prestore item)
PQ (PROC)
Q{S} (Q-pointer)
V (verb)
C (connective)
Attribute-defining items (in a data file) use the following d/codes:
A (attribute)
S (substitute attribute)
X (do-not-display attribute)

fileflgs

returns the internal file flags for file-defining items. Otherwise, a null ("") is returned.

filename

displays the filename of the file being pointed to if the type is a "q".

hotkey0

references attribute 30 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey0", the subroutine will be executed when the command "<ctrl>+x0" is executed. See "hotkeys".

hotkey1

references attribute 21 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey1", the subroutine will be executed when the command "<ctrl>+x1" is executed. See "hotkeys".

hotkey2

references attribute 22 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey2", the subroutine will be executed when the command "<ctrl>+x2" is executed. See "hotkeys".

hotkey3

references attribute 23 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey3", the subroutine will be executed when the command "<ctrl>+x3" is executed. See "hotkeys".

hotkey4

references attribute 24 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey4", the subroutine will be executed when the command "<ctrl>+x4" is executed. See "hotkeys".

hotkey5

references attribute 25 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey5", the subroutine will be executed when the command "<ctrl>+x5" is executed. See "hotkeys".

hotkey6

references attribute 26 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey6", the subroutine will be executed when the command "<ctrl>+x6" is executed. See "hotkeys".

hotkey7

references attribute 27 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey7", the subroutine will be executed when the command "<ctrl>+x7" is executed. See "hotkeys".

hotkey8

references attribute 28 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey8", the subroutine will be executed when the command "<ctrl>+x8" is executed. See "hotkeys".

hotkey9

references attribute 29 of a dictionary item.

If there is a FlashBASIC subroutine called on "hotkey9", the subroutine will be executed when the command "<ctrl>+x9" is executed. See "hotkeys".

icv

references the processing codes applied at input time, stored in attribute 14. These codes may be used for data validation as well as data transformation.

in.conv

references the processing codes applied at input time, stored in attribute 14. See "input-conversion".

input-conversion

references attribute 14 of a dictionary. This dictionary attribute is used exclusively by the Update processor.

There are many processing codes which can be called from "input-conversion" on both file-defining items and attribute-defining items.

item-id

references attribute 0 of the current item being processed by AQL.

itmflgs

returns the internal item flags for file-defining items.

Returns null ("") if no internal flags are set.

justification

displays the justification specification of the dictionary item being processed by AQL. See "attribute-type".

l/ret

references attribute 5 of a dict item and describes retrieval-lock codes specified for that file-defining item, if any.

l/upd

"l/upd" references attribute 6 of a dict item and describes update-lock codes specified for that file-defining item.

m/dict

an alternate means of referencing the "md" of the current account. It is simply a "q-pointer" which "points" to the current md.

macro

references attribute 15 of a file or attribute-defining item.

For a file-defining item, the text in this field describes the default attribute name list to include in the update screen. For an attribute-defining item the text describes the default attribute name list when zooming to another item.

This attribute is used for input exclusively by the Update processor and has no effect on any other processor, except when there is not default attribute list on the output-macro attribute of the file-defining item. In this case, the "macro" attribute dictates the default attributes to be used by the "list" processor.

Options

see "options: Update Processor".

Example

```
The following example is a file-defining item with the default attributes to
be used by the Update processor:

:ud filename filename
dictionary-code   a
modulo            67
structure
retrieval-lock
update-lock
output-conversion
correlative       id100
attribute-type    l
input-conversion
macro             name address zip phone (i
```

```
output-macro
output-conversion
description

In this example, since there are no attribute-defining items
specified on the output-conversion field, the default attributes used by the
LIST processor is determined by the 'macro' attribute.
The 'i' option displays the item-id of the file along with the specified
attributes.
It is possible to have 'multiple-views' of the data when UP is invoked. This
is accomplished by having a multi-valued list of attribute-defining items
defined on the macro dictionary.  To select which 'view' of the data to use, a
FlashBASIC subroutine must be called from input-conversion of the file-
defining item. In this subroutine, the "access" function is used to pass the
position value to UP.

The following example shows a file-defining item with a multi-value list of
attribute-defining items. A sample FlashBASIC subroutine is included, showing
exactly how to select the proper 'view' for UP.

:ud filename filename
dictionary-code   a
modulo            67
structure
retrieval-lock
update-lock
output-conversion
correlative       id100
attribute-type    l
input-conversion  call select.view
macro             name address zip phone (i
                  name phone comments
                  name phone fax address zip (i
output-macro
output-conversion
description
:u bp select.view
01 subroutine select.view(item)
02 item = access(3)
03 execute 'who' capturing who
04 user = field(who,' ',2)
05 if user = 'joe' then
06   access(18) = 2
07 end else
08   access(18) = 1
09 end
In this example, if the user is 'joe', the view displayed is : name phone
comments
```

md.name

references attribute 2 of MD items. For q-pointers, "md.name" describes the name of the file being pointed to.

modulo

references attribute 3 of the file-defining item being processed by AQL and describes the size of that file.

ocv

references attribute 14 of file or attribute-defining items and describes the output-conversion codes which modify the value prior to display.

out.conv

references attribute 14 of file or attribute-defining items and describes the output-conversion codes which modify the value prior to display.

output-conversion

references attribute 7 of a dictionary.

There are many processing codes which can be called from "output-conversion" on both file-defining and attribute-defining items.

output-macro

references attribute 16 of file-defining items. For file-defining items, the text in "output-macro" describes the default attribute list to be used in an AQL report.

For a "file-defining item", the text in this field is substituted into any TCL statement which outputs this file. For an attribute-defining item, this attribute has no relevance.

If the output-macro attribute is null on a file-defining item, the text from the "macro" attribute will be used for output as well as input.

If the attributes to be displayed are explicitly defined in the TCL statement, the default output-macro (or macro) attributes are ignored.

The TCL command "list filename" displays the file using the attributes defined in the output-macro attribute of the file-defining item.

However, the TCL command "list filename name address zip" displays the file showing the 'name' 'address' and 'zip' attributes.

r0

references the item-id, right justified, for sorting purposes.

Example

```
list md r0 (i
```

reallocation

used to resize a file when it is restored.

Before the file is saved, the new modulo is placed into attribute 13 of a file-defining item. During an account-restore or full file restore, (see :files) the file will be restored at the new size. If a file is too small, updating attribute 13 and account-save, delete-account then account-restore will cause the system to allocate more frames in the primary space for the file.

Parentheses must enclose the reallocation number.

Example

```
ct dict employees employees
    employees
001 D
002 6165
003 31
004
```

```
005
006
007
008
009 L
010
011
012
013 (97)
```

Notice that the file has a modulo of 31 frames and during the next account-restore the modulo will be reallocated to 97 frames.

ret.lock

references attribute 5 of a dict item and describes retrieval-lock codes specified for that file-defining item, if any. The retrieval lock controls access to the file for retrieval.

retrieval-lock

references attribute 5 of a dict item and describes the retrieval-lock code specified for that file-defining item.

rlk

references attribute 5 of a dict item and describes retrieval-lock codes specified for that file-defining item, if any.

s/amc

references attribute 4 of attribute-defining items and describes the controlling/dependent code specifications.

s/name

references attribute 3 of attribute-defining items and describes the substitute header for that attribute.

shd

references attribute 3 of attribute-defining items and describes the substitute header for that attribute.

size

returns the size (in bytes) of each item processed by AQL.

str

references attribute 4 of attribute-defining items and describes the controlling/dependent code specifications.

struc

references attribute 4 of attribute-defining items and describes the controlling/dependent code specifications.

structure

references attribute 4 of attribute-defining items and describes the controlling/dependent code specifications.

sub.header

references attribute 3 of attribute-defining items and describes the substitute header for that attribute.

substitute-header

references attribute 3 of attribute-defining items and describes the substitute header for that attribute.

summary

displays master dictionary items in a "generic" format.

Example

```
list md summary
```

syn

references attribute 12 of the file-defining item being processed by AQL and describes synonyms for that file, if any.

synonym

references attribute 12 of the file-defining item being processed by AQL and describes any synonyms for that file.

ty

displays the justification specification of the dictionary item being processed by AQL:

l Left justification.

r Right justification.

t Left with word wrap.

u Unconditional left justification.

w Process through OP before listing. Any valid OP command may follow this code or be used in the attribute. Only works in non-columnar formats.

ww Process all attributes from this attribute through the end of the item through OP before listing. only works in non-columnar format.

x Used with l, r, or t to expand display field to fill report.

upd.lock

references attribute 6 of a dict item and describes update-lock codes specified for that file-defining item, if any.

update-lock

references attribute 6 of a dict item and describes the update-lock code specified for that file-defining item.

v/cat

displays the file and program name for cataloged FlashBASIC programs.

Example

```
list md with v/cat v/cat
```

v/conv

references attribute 14 of file or attribute-defining items and describes the output-conversion codes which modify the value prior to display.

v/corr

references attribute 8 of the dictionary item being processed by AQL.

v/desc

references the nature and usage of the attribute-defining item being processed by AQL.

Example

```
list md v/desc
```

v/flags

describes flags which define options for the TCL processor.

The list of the flags kept in the third attribute of TCL verbs is shown below:

c Copies the item into local workspace.

e Saves the filename for the Update processor.

f Retrieves the filename only and ignores item-id's.

n Allows new items.

o Suppresses the line edit and "first-time" flag.

p Prints item-ids.

s Preserves the "active" (select) list.

u Allows update to items.

z Requires final pass after the last item.

Example

```
list md with v/flags v/flags
```

v/max

references the number of columns allocated for the dictionary item being processed by AQL.

v/mode

defines the virtual jump table defintions for verbs, or explanations for macros and PROCs.

v/name

describes the item-id of a TCL command item.

v/struc

references attribute 4 of attribute-defining items and describes the controlling/dependent code specifications.

v/tag

references attribute 3 of attribute-defining items and describes the substitute header for that attribute.

v/typ

displays the justification specification of the dictionary item being processed by AQL. See "attribute-type".

v/type

determines the type of TCL command.

Background/Phantom Process

background process which runs without a terminal.

D3 has a phantom process scheduler which is itself a background process.

The scheduler process sleeps until a running job completes or until a new job enters the queue.

Phantom jobs are spawned using the "z" verb from TCL. (See the "z" command).

A phantom job runs if there is an available phantom port. If there is no phantom port available, the phantom job is enqueued until one becomes available.

When a phantom job completes, it wakes the scheduler and logs itself off. The process is now available for other phantom tasks.

When a phantom job is executed, four things happen:

1 A pcb frame is fetched from overflow for the phantom job.

2 An item is written to the "dm,jobs," file which indicates that the request is waiting in the queue.

3 The item-id of the jobs file item is attached to the end of the queue.

4 The scheduler process is awakened if it is asleep.

When the scheduler wakes up, it checks the queue and reads the item from the jobs file that corresponds to the next item-id in the queue.

If a port is available, the scheduler does three things:

1 The scheduler updates the status of the item in the jobs file to 'running'.

2 It then adds an item to the jobs file using the "port.number" as the item-id.

3 Finally, the phantom job is logged on to the port.

When a phantom process has finished, the status in the jobs file is updated to 'logged off'. Next, the phantom is awakened and workspace is returned to overflow.

When the scheduler releases a phantom job back to overflow, the item whose "port.number" is the item-id of the job, is deleted from the jobs file.

The Spooler is a phantom job in D3.

See the "end" command for terminating phantom jobs.

The "p" option in the "users" file terminates phantom processes when a gfe is encountered, rather than just going to sleep. See "users".

To increase the number of phantom pibs on a Unix-based implementation of D3, edit the "pick0" file and decrease the value of "npibs" by the same number as that added to the value of "nphts". For example, if "npibs" is 10 and "nphts" is 12, change "npibs" to 8 and "nphts" to 14 to add 2 more phantoms.

background processing

See "phantom".

killing phantom jobs

see the (TCL) "end" command.

phantom process scheduler

background process which handles the phantom jobs submitted using the TCL command "z".

This process is usually started in the "system-coldstart" macro by the "start.ss" program.

scheduler

see phantom process scheduler

:background-start

starts the background processor. The background command will not run unless the background processor has been started with this command. This command is automatically run by the coldstart procedure.

Syntax

:background-start

:background-stop

stops the background processor. The background scheduler will no longer be able to run. Jobs can still be queued for execution when the scheduler is not running, they simply will not start. This verb has no effect on backgrounds that are already running. To stop those jobs, they must be logged off.

Syntax

:background-stop

background

starts a "phantom" job with rerun and delay facilities.

If the verb is executed with a TCL command attached, it behaves just like the 'zs' command, except the output is captured in the backgrounds.output file. If the verb is typed alone at tcl, then the user is asked a number of questions pertaining to the job. If the first parameter after the verb is a question mark, a list of already-created background jobs is presented for the user to modify.

Background jobs differ from regular phantom jobs in their ability to be scheduled for a later time and date, and their ability to rerun based on certain rules. A job can be configured to run once per minute, every thursday at 10:pm, or monday through friday between the hours of 8:00am and 5:00pm (for example).

All output from background jobs is captured in a D3 file called backgrounds.output. Information on the status of a job after each run is captured in a file called backgrounds.report. To stop background processing (but not phantom processing) at any time, use the :BACKGROUND-STOP command. To re-start them, use the :BACKGROUND-START command.

Syntax

background {TCL command { | TCL command or data { | ... }}}

background {?}

background

Example

```
<example 1>
:background sselect md | sl stuff<cr>
:
11:30:16 07 Mar 1995  from user jl line 130
```

```
jl*9*9928*41415 completed.

<example 2>
:background ?<cr>
<screen clears>
Seq Background          Command         User    Prt Act date Time
> 1 1PERMIN             POKE 11 TIME    jl      9   03/07/95
11:00
  2 WHERE               WHERE           jl      9   03/07/95 12:18

Select and press <RETURN>
<press ctrl-n to select sequence 2>
Background job WHERE was created at 10:17 on 03-07-95.
it will execute the following commands:
   WHERE<
It is automatically re-executed every 60 minutes.
It has already been executed 2 times.
It will run at 12:18 on 03-07-95.
Is this the background job you want to change (Y/N) ?y
Do you want to disable this job: (Y/N) ?n
Default is message will not be sent to this port upon completion
Return a message to this line when done (Y/N) ?
Commands already entered are:
     WHERE<
Do you want to re-enter (Y/N) ?
Next activation is on 03-07-95
Enter beginning date ?
Next activation time is 12:18
Enter starting time ?
Job will re-activate every 0 days
Rerun job after how many days  ?
background job will re-execute every 60 minutes
Rerun after how many hours, minutes (HH:MM) ?
If you wish to exlcude days from activation, enter codes for the days as
SUN,MON,TUE,WED,THU,FRI,SAT separated with commas (,) e.g. 'SAT,SUN'.
Excluding days ?
Exclude hours (hh:mm-hh:mm) ?
Background job WHERE was created at 10:17 on 03-07-95.
it will execute the following commands:
   WHERE<
It is automatically re-executed every 60 minutes.
It has already been executed 2 times.
It will run at 12:18 on 03-07-95.
Okay to start this background job (Y/N) ?y

<example 3>
:background
Enter background job's item-id ?jlmail
Creating new background job
Default is message will not be sent to this port upon completion
Return a message to this line when done (Y/N) ?n
Enter commands.  Use the "|" key to split data from the verb.
For example, if you wanted to make a copy of the "time" verb in the md giving
it the new name of "time.backup", the command would be: copy md time
(o|time.backup
Enter command?msg !10 You have @`field 2 @`select mail with
usr="jl"`` msgs
Enter command?
execute immediately (y/n) ?y
Is this background job to recur (Y/N) ?y
Job will re-activate every 0 days
Rerun job after how many days  ?
```

```
Rerun after how many hours, minutes (HH:MM) ?5:00
If you wish to exlcude days from activation, enter codes for the days as
SUN,MON,TUE,WED,THU,FRI,SAT separated with commas (,) e.g. 'SAT,SUN'.
Excluding days ?sat,sun
Exclude hours (hh:mm-hh:mm) ?
Background job jlmail was created at 12:16 on 03-07-95.
it will execute the following commands:
   msg !10 You have @`field 2 @`select mail with usr="jl"`` msgs <
It is automatically re-executed every 300 minutes.
It will not activate on SUN SAT .
It will run as soon as possible.
Okay to start this background job (Y/N) ?y
:
```

startsched

starts the "phantom" scheduler.

The scheduler is normally started on the very last phantom port of the system.

The phantom scheduler itself is a phantom process. See "scheduler", "phantoms" for more information.

Syntax

startsched {(options}

Options

port.number Designates the port.number on which to start the phantom scheduler. If not specified, the scheduler is run from the last phantom port.

i Initializes the "%r" item in the dictionary of the "jobs" file. This has the effect of NOT releasing any phantom processes that need wrapping up.

z

starts a "phantom" jobs.

"phantom" jobs run in the background, meaning that they do not necessarily need a terminal present.

A phantom process is a process that is initiated at a terminal and detaches itself from that terminal for execution independent of that terminal. It is processed as a background task and the results are displayed on the initiating terminal when the process is complete.

The number of phantom processes that can be run is determined when the system is installed. In general, there is one phantom process available for every eight users plus one additional process.

After a phantom process is initiated, the user can use the terminal originating the phantom process for other jobs. The "dm,jobs," file tracks phantom processes. Phantom processes can be monitored with the "list-jobs" command.

"zh" and "zhs" hold all terminal output in a hold file.

"zs" and "zhs" suppress error messages. Otherwise, error messages are displayed on the terminal that initiated the phantom process.

"TCL command" specifies the TCL statement to be executed as the phantom process. If a TCL command is specified, no further prompting appears, and the job is submitted using all the default values from the following questions.

If a TCL command is not specified, it prompts for the following:

user id:

If specified, the user password is requested. Otherwise, it defaults to the current user-id. Passwords are case-sensitive.

md id:

If specified, the account password is requested. Otherwise, it defaults to the current master dictionary.

TCL command:

Specifies the (initial) TCL statement to execute by the phantom process.

line:

Specifies the port number on which the process is to run. If not specified, a "phantom" line is used.

input data:

Designates input data to be stacked for use by the phantom process. Each separate string must be terminated by a <return>. A "null" response may be "stacked" by pressing "<ctrl>+n" followed by <return>. Pressing <return> at the "input data:" prompt indicates that all commands have been entered and the process is then released to the phantom scheduler.

Syntax

z {TCL command}

zh {TCL command}

zhs {TCL command}

zs {TCL command}

Example

```
:z
user id (<cr>=bob):<return>
md id (<cr>=ref):<return>
command: compile bp process.see.also
(type ctrl-n<cr> to enter null data)
input data:<return>
[438] Job #882632715 Submitted.

This illustrates the "prompted" sequence. After the job completes, a message
is transmitted to the sender, indicating that the job has completed:

09:05:18  29 Feb 1992 from user bob line 129
[241] Successful compile! 1 frame(s) used.
:z count entity
[438] Job #892243266 submitted.
This is the "unprompted" form.
```

Pick/BASIC—FlashBASIC

a high-level programming language; is the principal programming language bundled with the Pick System and was designed for the implementation of application requirements not handled by standard Pick functionality. Pick/BASIC is an extension of the Dartmouth BASIC (Beginners All-purpose Symbolic Instruction Code) language.

Pick/BASIC source can contain any number of statements in any order and can be stored as programs or subroutines called by other programs or processing codes. Pick/BASIC programs are created in the user's editor of choice and reside as items within a file.

Prior to execution, the programs must be compiled to convert the program into object code. Compiled programs are stored in the dictionary of the file that contains the source code. Compiled programs or subroutines can be cataloged as verbs withing the users master dictionary (Refer to the TCL commands compile, catalog, decatalog, and run).

For D3 Unix, Pick/BASIC source code may be compiled in the traditional way, or it may be optimized or *flash*ed. When optimized, the Pick/BASIC source code is translated to C source code, and then compiled using the host machine's C compiler, producing machine-specific assembly language object code.

In D3/NT, Pick/BASIC source code may also be compiled in the traditional way, or it may be optimized or *flash*ed. In this sense, though, the Pick/BASIC source code is compiled to an optimized tokenized object code which is then interpreted when run. This object code runs considerably faster than the original tokenized object code, but not quite as fast as the assembly language code of the C compiler.

Program Statements

A FlashBASIC program is composed of statements made up of FlashBASIC commands, variables, constants, expressions, and functions. FlashBASIC statements may contain arithmetic, relational, and logical expressions. These expressions are formed by combining specific operators with variables, constants, or FlashBASIC intrinsic functions. The value of a variable may change dynamically throughout the execution of a program. A constant, as its name implies, has the same value throughout the execution of a program. An intrinsic function performs a pre-defined operation on a supplied parameter(s).

Normally, within a FlashBASIC program, the programmer writes only one statement per line. This is done for the ease of reading and tracking of logic. More than one statement can be placed on a program line by separating them with semicolons. Statements which end with "then" or "else" on a line create a block or multiline structure. See the "then/else" entry for details on the "then ...else" statements.

Labels may be placed at the beginning of any FlashBASIC statement. A statement label may be numeric or alphanumeric. If it is alphanumeric, it must begin with a letter and may be any combination of letters, numbers, periods, or dollar signs and can end with an optional colon. The colon is also optional with a numeric label. Refer to the entry "statement labels".

Except for situations explicitly called out in the following sections, blank spaces appearing in the program line which are not part of a literal are ignored. Blank spaces and lines may be used within the program for purposes of appearance.

Data Representation

There are two types of data used in FlashBASIC: numeric and string. Numeric data consists of a series of digits and represents an amount, such as 255. String data consists of a set of characters, such as Jan Smith. Refer to the entry data representation for more information.

Variables

Variables are values that can change. Values can be assigned with an assignment statement. Assignment statements assign the value of an expression to a variable. A variable can be a simple variable, an array element, a dynamic array element, or a substring. For example:

var=x+2

assigns the value of the expression x+2 to the variable var. Refer to the entry assignment for more information about assignment statements.

Arrays

An array is a table of values of any data type. Each value in an array is called an element. The elements in an array are ordered allowing each array element to be accessed by specifying its position within the array.

FlashBASIC supports two types of arrays: dynamic arrays and dimensioned arrays. Dimensioned arrays can be defined with the dim statement, which specifies the name of the array and the number of elements in the array. Or, dimensioned arrays can be defined with the file statement.

Dimensioned arrays are limited to two dimensions (rows and columns). Additional dimensions can be obtained by referencing dynamic arrays from elements in a dimensioned array.

A dynamic array is a string containing attribute marks, value marks, and/or subvalue marks which are used as field delimiters. All elements within dynamic arrays are separated by one of these delimiters. Dynamic arrays do not have a fixed size nor are they dimensioned. Subscripts in dynamic arrays are enclosed in angle brackets (<>). They are called dynamic because the number of elements can be increased or decreased automatically by the system without the programmer needing to recompile the program.

In most cases, accessing a dimensioned array is faster than accessing the same size dynamic array. A dynamic array may be an element in a dimensioned array, in which case the dynamic array subscripts are specified following the dimensioned array subscript. Refer to the entries array, dimensioned array, and dynamic array for more information about arrays.

Operators

Expressions are formed by combining operators with variables, constants, or functions. When an expression is encountered as part of a FlashBASIC program statement, it is evaluated by performing the operations specified by each of the operators on the adjacent operands.

Each operator has a precedence rating and, in any given expression, the highest precedence operation is performed first. (0 is the highest precedence and 8 is the lowest. If there are two or more operators with the same precedence or if an operator appears more than once, the leftmost operation is performed first. Precedence of the operators is as follows:

```
Operator       Operation               Precedence
( )            Express evaluation      0 -- highest
[ ]            Substring extraction    0
^  **          Exponent                1
-              Unary minus             2
*              Multiplication          3
```

```
/             Division                3
+            Addition                 4
-  s         Subtraction              4
expr         Format string            5
cat  (:)     Concatenate              6
=  <  >  #   Relational operators     7
<>  ><
<=  >=
match        Pattern matching         7
and  &       Logical operators        8 - lowest
or  !
```

Parentheses may be used anywhere to clarify evaluation, even if they do not change the order. The parenthesized expression as a whole has highest precedence and is evaluated first. Within the parentheses, the rules of precedence apply.

The function associated with the *=, +=, -=, /= assignment statements take place after all left side operations and before the assignment.

Refer to the individual entries for each operator and the entries logical operators, operators, pattern matching, precedence, relational operators, and special characters.

Arithmetic Operators

Arithmetic expressions are formed by using the arithmetic operators. The simplest arithmetic expression is a single numeric constant, variable, or function. A simple arithmetic expression may combine two operands using an arithmetic operator. More complicated arithmetic expressions are formed by combining simple expressions using arithmetic operators.

If a string value containing only numeric characters is used in an arithmetic expression, it is evaluated as a decimal number. If a string value containing non-numeric characters is used in an arithmetic expression, a warning message is printed and zero is assumed for the string value.

Format Strings

Numeric and non-numeric strings may be formatted using format strings, which consist of numeric masks and format masks. The numeric mask code controls justification, precision, scaling, and credit indication. The format mask code controls field length and fill characters.

The entire format string is enclosed in quotation marks. If a format mask is used, it should be enclosed in parentheses within the quotation marks. The entire format string may be used as a literal, or it may be assigned to a variable. A format string literal can immediately follow the string it is to format. A format string variable must be separated by at least one space from the string it is to format. The format string may also be used directly in conjunction with the print statement. For more information, refer to the entry masking.

String Expressions

A string expression can be a string constant, a variable with a string value, a substring, or a concatenation of string expressions. String expressions may be combined with arithmetic expressions.

Concatenation of strings is specified by a colon (:) or by the operator cat. The operator cat must be preceded and followed by blanks. Refer to the entry string expressions.

Substrings

A substring is a set of characters that makes up part of a whole string. The syntax to specify a substring is:

substring = string[m,n]

where "string" is a string variable, "m" is the starting character position, and "n" is the substring length. For example, if the current value of variable S is the string "ABCDEFG", then the current value of S[3,2] is the substring "CD" (the two-character substring starting at character position 3 of string S).

If the starting character specification is past the end of the string value, then an empty substring value is selected. If the starting character specification is negative or zero, then the substring is assumed to start at character one.

If the substring length specification exceeds the remaining number of characters in the string, then the remaining string is selected. If the substring length specification is negative or zero, then an empty substring is selected.

A segment of a character string may be changed or substituted without affecting the remainder of the string by assigning the new segment to the variable containing the original string. The new segment is assigned to the original string variable with the starting character and segment length specified with the variable. The string variable may be a single variable or an array element. For example, if the current value of the variable S above ("ABCDEFG") is to have characters 3 to 5 inclusive replaced by the string "123" the assignment S[3,3]="123" accomplishes this substitution so that the variable S now becomes "AB123FG". The general syntax for substring replacement is:

string[m,n] = substring

where "string" is a string variable, "substring" is the substring to be inserted, "m" is the starting character position, and "n" is the number of characters in the string which are replaced. See the Pick Systems Reference Manual "substring assignment" token for more details.

One or more fields or substrings within a character string variable which are delimited by a specific character can be changed by a single assignment.

string[delimiter,m,n] = substrings

where "string" is a string expression or variable, "substrings" is one or more substrings (separated by a delimiter) to be substituted, "delimiter" is the delimiter separating fields or substrings in the original string, "m" is the starting field position within the string for the substring to be placed, and "n" is the number of fields or substrings to be replaced.

If m is less than 1, 1 is assumed. If m is greater than the number of fields within string then the number of fields in string is increased by adding null fields separated by the delimiter so that string has the length of m fields. Multiple substrings separated by the specified delimiter can be substituted in the original string.

The expression n has different effects depending on whether it is positive, negative or zero. If it is positive, then n fields in string are replaced with n substrings. If m + n is greater than the number of fields in string, then the number of fields in the resulting string is increased to the value m + n, and the extra substrings are concatenated to the end of the original string. If m + n is less than the number of fields in string but n is greater than the number of substrings specified, only the specified substrings are substituted and the remainder of the n fields in string are nulled.

If n is zero, then no fields in string are deleted and the specified substring(s) are inserted into string before the mth field. If n is less than zero, then n fields greater than or equal to m are deleted from string and the entire expression containing the substring(s) is inserted at that point in the string.

Relational Expressions

A relational expression evaluates to 1 if the relation is true, and evaluates to 0 if the relation is false. Relational operators have lower precedence than all arithmetic and string operators. Therefore, relational operators are only evaluated after all arithmetic and string operations have been evaluated.

If one operand is numeric and one operand is string, both operands are treated as strings. To resolve a numeric relation, the values of the expression are compared. To resolve a string relation, the characters in each string are compared one at a time from leftmost characters to rightmost. If no unequal character pairs are found, the strings are considered to be equal.

If the two strings are not the same length, and the shorter string is otherwise identical to the beginning of the longer string, the longer string is considered greater than the shorter string.

The resolution of unequal character pairs depends on the characters. If the unequal pair of characters are non-alphabetic characters, the characters are ranked according to their numeric ASCII code equivalents. The string contributing the higher numeric ASCII code equivalent is considered to be greater than the other string. If the unequal pair of characters are letters of the alphabet, the pair is ranked in strict alphabetical order.

If one of the characters is a letter of the alphabet and the other character is non-alphabetic, the characters are ranked according to the numeric ASCII value of the uppercase letter, even if the letter is in lowercase, and the numeric ASCII value of the non-alphabetic character.

Arrays and Relational Expressions

Elements of dimensioned and dynamic arrays can be compared in relational expressions using the standard syntax. In addition, all elements of the array can be compared to a specified value by using an asterisk (*) to identify the element. In multi-dimensioned arrays, one or more dimensions can be specified using the example.

When the asterisk is used, the operators < > = <= >= # return a true value if at least one element of the array satisfies the relation. The asterisk may be specified only in relational expressions and only in one array expression in a statement.

Match - Pattern Matching Operator

string.variable match pattern

The match operator compares a string value to a predefined pattern and evaluates to 1 (true) or 0 (false). The pattern may consist of any combination of the following:

pattern operation

nn tests for n integers.

na tests for n alphabetic characters.

nx tests for n characters of any type.

'string' tests for literal string (double quotes are used when testing for a literal string in combination with the above patterns).

If n is 0, the relation evaluates to true only if all the characters in the string conform to the specified data type. The character n tests only for integers; +, -, and . are not considered to be numeric.

Logical Expressions

Logical expressions (also called Boolean expressions) are the result of applying logical (Boolean) operators to relational or arithmetic expressions.

Logical operators work on the true or false results of relational or arithmetic expressions. (Expressions are considered false when equal to zero, and are considered true when non-zero.)

Logical operators have the lowest precedence and are evaluated after all other operations have been evaluated. If two or more logical operators appear in an expression, the leftmost is performed first.

Logical operators act on their associated operands as follows:

a OR b is true (evaluates to 1) if a is true or b is true.

a ! b is false (evaluates to 0) if a and b are both false.

a AND b is true (evaluates to 1) only if both a and b are true.

a & b is false (evaluates to 0) if a is false or b is false or both are false.

Retrieval and Update Lock Codes

If a FlashBASIC program is executed by a user who does not have retrieval and/or update privileges for a file that does and is opened in the program, and then attempts to read and/or write to such a file, this will result in termination of the program with an error message. In order to read or write to such a file, users must make sure that the retrieval and update lock codes match those of the file that is opened. This is a system function and cannot be accomplished from FlashBASIC. If the file is retrieval protected, the file cannot be opened and the program will be terminated with an error message; see the Security Section.

FlashBASIC Statements

The following is a list of FlashBASIC statements. For more detailed information about these statements, refer to their individual entries in the body of the Pick Systems Reference Manual.

```
*                   !                   abort
assigned            aux                 begin               break
call                capturing           casing
chain               chap (7.0)          clear
clearfile           clearselect (7.0)   close
common              convert             crt
data                debug               del
delete              dim                 echo  off
echo on             else                end
end                 case                enter               equ{ate}
execute             exit                file
footing             for..next
for..next  until    for..next  while    get
gosub               go{to}              heading
hush (7.0)          if..then..else      in
{$}include          input               inputclear
inputparity (7.0)   inputnull (7.0)
inputtrap           ins                 insert
key                 let                 locate
```

```
lock                locked              log
loop                loop..until         loop..while
mat                 matbuild            matparse
matread{u}          matwrite{u}         next
null                on  gosub           on  goto
onerr               open                out
page                precision           print
print  on           printer  close      printer  off
printer  on         procread            procwrite
program             prompt              read{u}
readnext            readt               readtl
readtx              readv{u}            release
rem                 repeat              replace
return              returning           return  to
rewind              root                rqm
select              send                sleep
setting             shift               spoolq (7.0)        stop
subroutine          tcl                 tclread
then..else          unlock              waiting
weof                write{u}            writet
writev{u}
```

FlashBASIC Functions

The following is a list of FlashBASIC functions. For more detailed information about these functions, refer to their individual entries.

```
@functions          %functions          abs
access              alpha               ascii
assigned            change(7.0)         char
col1                col2                convert
cos                 count               date
dcount              delete              dquote(7.0)
dtx                 ebcdic              ereplace(7.0)
error               exchange(7.0)       exp
extract             field               fmt(7.0)
fold                iconv               index
insert              inmat(7.0)          int
len                 ln                  maximum(7.0)
mod                 not                 num
occurs              oconv               pwr
rem                 replace             rnd
scan(7.0)           sentence(7.0)       seq
sin                 sort                soundex
space               spooler(7.0)        squote(7.0)
sqrt                str                 summation(7.0)
sum                 swap(7.0)           system
tan                 time                timedate
trim                xtd
```

FlashBASIC Symbolic Debugger

The FlashBASIC Symbolic Debugger facilitates the debugging of new FlashBASIC programs and the maintenance of existing FlashBASIC programs. The FlashBASIC debugger requires sys2 privileges and has the following general capabilities:

- Step through execution of program in single or multiple steps.

- Transfer to a specified line number.

- Break execution at specified line numbers or when specified logical conditions have been satisfied.

- Display and/or change any variables, including dimensioned variables.

- Trace variables.
- Enter the system debugger.
- Direct output to either terminal or printer.
- Display and/or pop GOSUB stack.
- Display source code lines.

The FlashBASIC debugger may be entered at execution time under the following conditions:

- The <break> key is pressed.
- The d (debug) option is specified with the run verb.
- The debug statement is executed in the program.
- A runtime error is encountered (unless the a (abort) option of the run verb is selected).
- The abort statement is executed in the program.

When the FlashBASIC debugger is entered, it indicates the source code line number to be executed next and prompts for commands with an asterisk (*).

Symbol Table

When a FlashBASIC program is compiled, a symbol table is generated, unless the s (suppress table) option has been used. The symbol table is used by the FlashBASIC debugger to reference symbolic variables.

If a program calls an external subroutine, and the FlashBASIC debugger has been entered previously, a complete symbol table is set up for the external subroutine. Break points set up for a subroutine are independent from break points set up in the main program or other subroutines; however, the execution counters e and n are global. That is, the break point counters count both main program and subroutine break points in the order they are encountered. Multiple symbol tables allow programmers to set up different break points and/or variable traces for different subroutines.

FlashBASIC debugger commands are listed below. For more detailed information about each command, refer to it's entry.

b Establish a break point condition.

c Toggles source line display on and off.

d Display breakpoint and trace tables.

debug Escape to system debugger.

e Execute a specified number of lines.

end End program execution and return to TCL.

g Begin program execution at a specified line.

j Executes next line, displays next line if "c" option enabled.

k Remove a specified breakpoint.

l Display specified source lines.

l* Display all lines.

lp Toggle output device between terminal and printer.

n Execute through a specified number of breakpoints.

off Logs the user off the system.

p Inhibit display of program output.

pc Close output to spooler.

r Pop the top element off the gosub stack.

s Display the gosub stack.

t Set trace for a specified variable .

u Remove a specified trace.

$ Verify object code.

v Verify object code.

/var Print value of a specified variable.

/arr(m) Print value of a specified array element.

/* Print value of entire symbol table.

[] Display specified number of characters.

? Show the current program name, line number, and verify the program.

The following debugger commands are available from FlashBASIC:

down Move down the command stack.

edit Use the Update processor to edit the current program.

help Show the debugger commands.

up Move up the command stack.

?! show the current command stack and parameters.

$log

records compilation errors while compiling a FlashBASIC program. See "FlashBASIC error logging" for more information.

ac.expression

any numeric constant or any arithmetic, logical, or string expression which evaluates to a valid numeric value, used to reference an "attribute count" within statements, functions, and array or string variables which require or allow them. See "data representation" for valid numeric values.

Example

```
readv company.name from customer.file,cust.id,1 else...

In this example, "1" is the ac.expression.
apos = 2
readv company.address from customer.file,cust.id,apos else..
The variable "apos" must be numeric; in this case it is a 2.
customer.item<1> = "test string"
The numeric constant "1" indicates attribute 1 of the string in the
"customer.item" variable.
audit.item<bpos+i> = amount.list<bpos>
The arithmetic expression, "bpos+i", calculates the attribute
position in "audit.item".
```

arithmetic expressions

perform mathematical calculations on any set of operand expressions.

Symbol precedence operation

^ 1 exponentiation

* 2 multiplication

/ 2 division

\ 2 remainder

+ 3 addition

- 3 subtraction

Expressions are evaluated in order of precedence unless placed within parentheses.

Expressions at the same precedence evaluate left-to-right:

10+2*10 evaluates to 30.

Expressions within the innermost parentheses are evaluated first:

(10+2)*10 evaluates to 120.

(10+(2*10)) evaluates to 30.

arithmetic operators

add, subtract, multiply, and divide numeric operands in arithmetic expressions.

The simplest arithmetic expression is a single numeric constant, variable, or function. More complex arithmetic expressions may combine two or more expressions using an arithmetic operator.

If a string value is used in an arithmetic expression, the FlashBASIC runtime package attempts to convert it into a number. If it fails, "0" (zero) is used and a warning displays.

Example

```
total = (invoice.amount - discount) + tax
```

array

tables of values of any data type.

Each value in an array is called an "element" of the array and the elements are ordered. An array element is accessed by specifying its position in the array using a subscript variable. The variable must have one subscript for each dimension of the array. For example, in a two-dimensional array, the first subscript specifies the row, while the second specifies the column. If the second subscript is missing, it defaults to 1.

FlashBASIC has two types of arrays:

Dynamic arrays:

A string can be handled as a three-dimensional array based on the three main system delimiters; attribute marks, value marks, and subvalue marks. These are called "dynamic arrays". "Dynamic" for the ability to insert and delete array elements without having initially declared the array size. A dynamic array is nothing more than a string.

Dimensioned arrays:

Dimensioned arrays are a one or two-dimensional array of separate variables. These are called "dimensioned" arrays due to the need to declare the number of array elements to the compiler or runtime with the "dim" statement.

Dimensioned arrays are also defined by the "file" statement. In this case, the FlashBASIC compiler dimensions the array, using the file name as the array name. It determines the number of elements in the array from the program itself and the file dictionary. The array is dimensioned based on the maximum attribute number addressed in the program, plus 1.

Subscripts in dimensioned arrays are enclosed in parentheses.

Dimensioned arrays are limited to two dimensions (rows and columns). Additional dimensions can be obtained by referencing dynamic arrays from elements in a dimensioned array.

All elements within dynamic arrays are separated by delimiters. Dynamic arrays consist of fields (or sets of values) separated by attribute marks (char(254)). Each attribute or field in a dynamic array may consist of a number of values separated by value marks (char(253)). Each value may contain several subvalues separated by subvalue marks (char(252)).

Dynamic arrays do not have a fixed size, nor are they pre-dimensioned.

Subscripts in dynamic arrays are enclosed in angle brackets (<>). Dynamic arrays are limited to three dimensions (attribute, value, and subvalue). They are called dynamic because the number of elements can increase or decrease automatically.

In most cases, accessing a dimensioned array is significantly faster than accessing the same size dynamic array.

A dynamic array may be an element in a dimensioned array, in which case the dynamic array subscripts are specified following the dimensioned array subscript.

The following is a dynamic array element that has been referenced from the dimensioned array, "m":

m(a,b)<x,y,z>

where:

"a" is the row number of the dimensioned array

"b" is the column number of the dimensioned array

"x" is the attribute number of the dynamic array

"y" is the value number of the attribute

"z" is the subvalue number of the value

The element "m(a,b)" of the dimensioned array points to the beginning of the dynamic array.

Dimensioned arrays may be redimensioned as many times as desired. See the example.

Example

```
max = 100
n = 1
dim a(max)
execute "select md"
loop
  readnext id else exit
  a(n) = id
  if (n >= max) then
    max += 100
    dim a(max)
```

```
  end
  n += 1
repeat
This loads all the item-ids from the current md into the dimenisoned
array, "a".
```

array references

dimensioned and dynamic arrays may be referenced in FlashBASIC programs.

Dimensioned array references have the form:

array.variable(num.expression) or array.variable(num.expression, num.expression)

Dimensioned arrays have a maximum of two dimensions and must first be defined with a "dim" or "dimension" statement.

When used in conjunction with the "file" statement, dimensioned array elements may be referenced by the actual item-id of the attribute-defining item (adi) in the associated dictionary:

file entity

.

.

if fv.entity(name) = "" then

Dynamic (or "string") array references have the forms:

1) dynamic.array.variable<ac.expression>

This form references an entire attribute location of a dynamic array.

2) dynamic.array.variable<ac.expression, vc.expression>

This form references a value within an attribute.

3) dynamic.array.variable<ac.expression, vc.expression, sc.expression>

This form references a subvalue within a value.

Combining dimensioned and dynamic array elements:

A specific value or subvalue of a dimensioned array can be specified using the following forms:

array(ac.expression)<ac.expression, vc.expression>

This references the element of the dimensioned array derived from the current value of "ac.expression". Within this array element, the value count derived from "vc.expression" is referenced.

array(ac.expression)<ac.expression, vc.expression, sc.expression>

As in the previous case, with the additional reference to a subvalue location derived from "sc.expression".

D3 also allows every element within either type of array to be compared using an "*" as a wildcard character. The position of the "*" determines which element is searched.

Syntax

array.variable(ac.expression)

array.variable(row.number,col.number)

array.variable(adi)

dynamic.array.variable<ac.expression>

dynamic.array.variable<ac.expression, vc.expression>

dynamic.array.variable<ac.expression, vc.expression, sc.expression>

Example

```
customer.item(1) = name
This assigns the value of "name" to the first element of the
dimensioned array "customer.item".
customer.item<2,2> = address2
This assigns the value of "address2" to the second value in the
second attribute of dynamic array "customer.item".
if array(*) = "x" then...
This searches for any element in the dimensioned array "array" equal
to the letter "x".
if customer.item<2,*> = "" then...
This searches for any value of attribute 2 in the dynamic array
"customer.item" equal to null string ("").
if customer.item<*,*,*> = "" then...
This searches for any subvalue in any value in any attribute in the
dynamic array "customer.item" that is equal to a null string ("").
if array<*,1,*> = "xxx" then...
This searches for any subvalue in value 1 in any attribute in the
dynamic array "array" that is equal to "xxx".
```

array.variable

references a dimensioned array.

Example

```
When an "array.variable" is used to describe a dimensioned array, it
must be used with a subscript to indicate the specific variable
descriptor in the array:
a(10) or a(n)
Otherwise it must be used in a "mat" statement to indicate all
elements in the array:
mat a = ''
This sets every variable in the array "a" to null.
matread a from ......
This reads an item and parses the attributes to "a".
```

arrays, relational expressions

a brief discussion of referencing arrays in FlashBASIC and the use of the special "wildcard" array searching character.

Individual elements of dimensioned and dynamic arrays are compared in relational expressions using the standard syntax. All elements of the array can be compared to a specified value by using an "*" (asterisk) to identify the element.

In multi-dimensioned arrays, one or more dimensions can be specified using the example. When the asterisk is used, the operators "<", ">", "=", "<=", ">=" and "#" return a true value if at least one element of the array satisfies the relation. The asterisk may be specified only in relational expressions and only in one array expression in a statement.

Example

```
dim a(10)
if a(*) = "x" then ....
This compares all elements in the dimensioned array "a" to the value
"x".  If any element equals "x", the "then" statement is performed.
if s<*> >= "10" then....
Checks that any element in the dynmaic array "s" is greater-than or
equal to "10".
```

```
dim a(10,10)
if a(*,*)<*,*,*> = "x" then....
Checks every attribute, value, and subvalue in every string in the
dimensioned array "a".
```

assignment

value of an expression to a variable.

The variable can be a simple variable, an array element, a dynamic array element, a substring, or all of the above.

Syntax

variable = expression

Example

```
name = "george"
The variable "name" is assigned the string "george".
result = (unit.price * qty)
The variable "result" is assigned to the result of the arithmetic
expression.
line = result: " is the answer"
The variable "line" is assigned the result of the string expression.
item<-1> = code
Adds the value of "code" as the last attribute of the dynamic array
"item".
item<codes,-1> = code
Adds the value of "code" as the last value in an attribute of codes.
customer.array(5) = "fred"
The value, "fred", is assigned to the fifth element of the
dimensioned array, "customer.array".
item<1,2,2> = "100"
The second subvalue in the second value in the first attribute of the
string in "item" is replaces with the string "100".
string[3,2] = "ca"
Character positions 3 and 4 of "string" are replaced with the string
"ca".
item<1,2,1>[3,2] = "ca"
The 3rd and 4th character positions of the string found in the first
subvalue in the second value in the first attribute of "item" is
replaced with the string "ca".
```

Boolean Evaluation

brief discussion on how zero's, null's, strings and numbers are evaluated on various D3 platforms.

Traditionally, the result of a logical or Boolean expression is considered "true" if it evaluates to 1 and false if it evaluates to 0 (zero).

In the D3 System, expressions which depend on a result of true or false also will evaluate other values as true or false. This tends to vary somewhat between implementations. In generic D3, any non-zero integer that is positive or negative, evaluates to true. For example:

x = 5

if x then stop

Since x is a non-zero integer, the program takes the then branch and stops. In generic D3, any zero or null value evaluates to false. For instance:

y = ""

if y then stop else print "yup"

This prints "yup" since "y" is evaluated as false. Some D3 implementations additionally evaluate any negative numbers as false, and positive numbers as true. This also holds true for "+", "-", ".", "+." and "-.".

This means that you will have to take the "truth test" with your system to determine how it handles true and false. This program will do it:

loop

print "value to test " :

input value

until value = "quit" do

if value then print "true" else print "false"

repeat

Try it with negative numbers, null (<return>), positive numbers, letters and whatever else you can think of.

A relational expression evaluates to 1 if the relation is true, and evaluates to 0 (zero) if the relation is false.

Relational operators have lower precedence than all arithmetic and string operators. Therefore, relational operators are only evaluated after all arithmetic and string operations have been evaluated.

In the logical expression:

x=y

The resolution of equal and unequal character pairs are handled as follows:

- When both "x" and "y" are numeric, the comparison is numeric.

- When either "x" is numeric and "y" is a string, or "x" is a string and "y" is numeric, the string is converted to an equivalent numeric, if possible. If the conversion is successful, the comparison is numeric. If the conversion is not possible, the number is converted to a string, and the comparison is lexical.

- When both "x" and "y" are strings, both are converted to numeric, if possible, and the comparison is numeric. If the conversion is not possible, the comparison is lexical.

The case of characters does not affect a comparison if "casing" is off. For example:

if "a" = "A" then..

This is true with "casing off". It is false with "casing on".

Example

```
x = "1"
y = "001"
if x=y then crt 'equal' else crt 'not equal'
"equal" is displayed, as both "x" and "y" evaluate to a numeric "1".
```

Boolean expressions

See: logical expressions

c function

can be called from a FlashBASIC program or subroutine in D3 Unix implementations using a syntax similar to that of normal C.

Commonly used functions are included in the run time package and require no special linking. These functions are referred to as built-in functions. A 'user-defined' function may also be included in a FlashBASIC application by linking it with the Monitor, thus making a 'user-defined built-in' function.

C function calls can be embedded in FlashBASIC expressions, FlashBASIC statements or can be used as arguments to other C function calls. Combined size of all parameters passed may NOT exceed 32 KB. This size can be changed, however, by the "set-cmem" TCL command.

If a "type" specifier is omitted before an argument name, the argument is assumed to be of type integer when "arg" is a number, or to be of type pointer when "arg" is a dynamic array (string). If type is omitted before the function name, the function is assumed to be of type integer.

"arg0" ... "argn" are the optional arguments to the function; up to 15 arguments are supported.

The return type of a function is "void" if the function is not part of an assignment statement or used as an argument to another C function or FlashBASIC statement. Whenever the return value of a function can be neglected, the type can be overridden by (void).

When a call to a function is made:

- All arithmetic arguments are passed by value.

- All dimensioned arrays are passed by reference.

- All dynamic arrays (strings) are passed by reference.

- String constants (strings between quotes) and sub-strings (string[m,n]) cannot be modified. If a C function tries to modify a string beyond the end of the string, the string will be truncated to its original size but no error will be reported. As is usual in C, no data translation will occur implicitly.

- Pointers are all treated as pointers to characters. When assigned to a FlashBASIC variable, a pointer can only be passed to another C function or be tested for the value 0 (NULL pointer), anything other than this makes no sense. The following statement has unpredictable results:

ptr=(char*)%malloc(1024)

if ptr > 0 then print 'ok'

The only valid form is to test for ptr = 0 or ptr # 0.

Argument Types

The following argument types are supported: <nothing>

Default. If "arg" is a number, an (int) is passed to the function. If "arg" is a dynamic array (string), a pointer to the first character of the string is passed to the function. If "arg" is a dimensioned array, a pointer to an image of the dimensioned array is passed to the function. See 'Passing Dimensioned Arrays' below.

(int) (D3 Unix)

Integer. The "arg" is divided by the PRECISION factor and passed to the C function. Integers are 32 bit signed elements.

(char)

Character. If "arg" is a dynamic array (string), the first character is passed to the C function. If "arg" is a number it is divided by the PRECISION factor and the result is truncated to 8 bits and passed to the C function.

(char*)

Pointer. The "arg" is a number which is passed to the C function without being divided by the PRECISION factor. The only legal use of this type is to pass a null pointer or a pointer returned by a previous call to a C function to another C function.

All types, except pointers can be prefixed by the keyword "unsigned", (unsigned char) for example.

NOTE: The PRECISION factor is a number between 0 and 9, with 4 being the default value. When a number is divided by the PRECISION factor, the number is actually being divided by 10 raised to the power of PRECISION.

Passing Dimensioned Arrays

Dimensioned arrays of integers can be passed to an external C function. When the array is passed, the C function receives a pointer to the array and all the integers in the array are divided by the PRECISION factor. If the array is multidimensional, the integers are organized column by column. For example, if we have an array that is dimensioned to (2, 5) in FlashBASIC, the values in the C "array" would be in this order: (1, 1), (2, 1), (1, 2), (2, 2), (1, 3), (2, 3), etc.

Function Types

The following function types are supported: <nothing>

Default. The return value is assumed to be an integer. It is multiplied by the PRECISION factor.

(void)

The return value is discarded. When prefixed with this type, the function cannot be part of an assignment or be used as an argument to another C function or FlashBASIC statement or function.

(int) (D3 Unix)

Integer. The return value is a signed integer. It is multiplied by the PRECISION factor.

(char)

Character. The return value is stored as a dynamic array containing only one character.

(char*)

Pointer. The return value is stored without being multiplied by the PRECISION factor. The only legal use of this type is to store a pointer that will be used as an argument to another C function.

'Address of' Unary Operator

The C unary operator, & (ampersand), is used to pass the address of an integer to an external C function. This is the only valid form. Note that FlashBASIC converts numbers which are too large into strings. In this case, the C function would receive a pointer to a character instead of a pointer to an integer.

Space Reserve Statement

When a FlashBASIC variable (a dynamic array or an element of a one or two-dimensional array) is used to store data returned by a call to a C function (by means of a pointer passed as an argument), the FlashBASIC variable must have been assigned a size before the call to the function. This needs to be done because C has no notion of dynamic arrays. If a size is not assigned before the result of a C function call is stored in the variable, the data is truncated. Space can be reserved using the following statement:

char variable[size] {,variable[size], ...}

This reserves at least "size" bytes for "variable". "size" can be a constant or an expression. "variable" can be a dynamic array or an element of a one or two-dimensional array. After reserving space for "variable", the content of the variable is undefined.

If a string longer than "size" is assigned to a variable, it is truncated and the characters beyond the given size are ignored. If a string shorter than "size" is assigned to variable, its content after the defined size remains undefined.

Static Data

When using 'user-defined built-in' functions, static data defined as part of a user-defined function remains valid as long as the D3 process is not disconnected from the virtual machine. This is true even if the FlashBASIC program is finished executing and has returned to TCL. The scope of static data lasts longer than in conventional Unix programs.

Since static data takes up space in the data section of each D3 process (therefore main memory), it is not advisable to have large amounts of static space.

D3 Unix includes:

The following 'header files' are provided in the file bp,unix.h, in the dm account.

errno.h 'errno' values.

fcntl.h Codes for %open(), %creat

ipc.h Semaphores, shared memory, messages.

mode.h File access codes.

sem.h Semaphore operations.

signal.h Unix signal signal numbers.

Use "see.also" for information on all system-defined built-in C functions. In most cases, no description is given unless there is something special to mention. Please refer to the Unix or the appropriate system documentation if more details are necessary.

The following rules apply when making a call to a C function:

- The parameters for each funtion must be enclosed in parentheses.

- There must be no spaces between the percent sign (%), the function name and the left parenthesis.

- The function is assumed to return an integer unless explicit casting is performed.

Syntax

{variable=}{(type)}%function({{(type)} arg0 {, ... {(type)} argn}})

c function, user-defined

can be incorporated in the D3 Monitor, thus making C functions available to all applications running on the system.

This way of 'customizing' the Monitor should be reserved to functions which are widely used in the applications running on the system (such as a math package, graphics, communications, etc.). It is a fairly complex procedure which should be attempted only by programmers experienced both in D3 and Unix.

The main advantage of this procedure is efficiency. The cost, however, is obviously the loss of portability to non D3 Unix platforms. Also, the user-defined built-in function must be integrated in the Monitor at every new release.

Integrating a C function in the Monitor requires the following steps:

- Build a table to inform the FlashBASIC compiler about the new user written function.

- Build a branch table which will be used by the Monitor to access the user written function.

- Create a new Monitor.

The following describes the necessary procedures through the example shown in the 'example' section below. The tools used in this section are described completely under their respective entries in the documentation.

It must be emphasized that, especially while debugging with Unix debuggers, changing data in the D3 environment can cause damage to the data base.

Most of the following procedures can be done from TCL (by prefixing Unix commands with a "!"). The D3 process has to be stopped (disconnected), however, when the new Monitor is about to be created, since most Unix versions will not allow rebuilding an object module which is currently being executed.

The steps required for adding a new function to the Monitor are:

- Step 1: Logon to the dm account.

- Step 2: Add the new module to the FlashBASIC-C interface.

addbi fpadd fpsub fpmul fpdiv

The "addbi" commands adds the functions to the item dm,messages, user.builtin and creates and compiles the file px_user.c which is the built-in User branch table.

This step needs to be done only once.

- Step 3: Edit and compile the C program

After entering the C program, compile it:

!cc -c fp.c

This creates the module fp.o.

- Step 4: Add the new module to the user function library

!ar vru libgmu.a fp.o

- Step 5: Disconnect the current D3 Process

If the current line is the line 0, the D3 virtual machine will shutdown. At this stage, since user processes normally execute the reference Monitor /usr/bin/ap, it is not necessary to stop the virtual machine.

disc

This drops back to the shell.

- Step 6: Build the new monitor.

make -f /usr/lib/pick/Makefile

This creates a monitor named ap on the current directory. If errors occur while "make"ing it may be necessary to make modifications to the Makefile (see Makefile).

- Step 7: Test the new Monitor

Generally, it is not necessary to shutdown the D3 virtual machine to start a user process on a new Monitor. The new Monitor can be executed by one or more user processes for test purpose. Just type: ./ap

This will start a User process executing the new Monitor. To make any change, repeat the procedure from steps 3 to 7 until the C function is working properly. Debugging can be done with the usual Unix debuggers like sdb or dbx.

Logon to dm. Enter the following FlashBASIC program:

bp fpdiv:

cfunction unix.builtin, user.builtin

* Reserve space for the resulting string

char result[64]

* Call the C function (no return code)

%fpdiv("1.23e-9", "1.2e-3", result, 64)

* Extract result (0 terminated string)

print 'Res=':field(result,char(0),1)

Compile it and run it:

compile bp fpdiv

run bp fpdiv

Res=1.025000000000000013890527183341e-06

Repeat steps 3 to 7 until the new function works properly.

- Step 8: Move the monitor to the working directory.

When the new Monitor is properly tested, it can be moved to the common directory, where Users can access it. Copying the new Monitor must be done as super-user (root) since the directory is write protected.

su (enter password)

chmod 06755 ap

chown root ap

cp ap /usr/bin

Shutdown and reboot the D3 virtual machine. User processes now execute the new monitor.

Example

```
Consider the following C functions which perform the four basic
operations on two strings, s1 and s2, representing floating point
```

```
numbers in FORTRAN F-format or E-format ("[-]d.dddE[+-]dd"). A
string, up to p characters long, representing the result, is returned
into s3.
fp.c :
double atof();
fpadd( s1, s2, s3, p)
char *s1, *s2, *s3;
int p;
{
  gcvt( atof(s1)+atof(s2), p, s3 );
}
fpsub( s1, s2, s3, p)
char *s1, *s2, *s3;
int p;
{
  gcvt( atof(s1)-atof(s2), p, s3 );
}
fpmul( s1, s2, s3, p)
char *s1, *s2, *s3;
int p;
{
  gcvt( atof(s1)*atof(s2), p, s3 );
}
fpdiv( s1, s2, s3, p)
char *s1, *s2, *s3;
int p;
{
  gcvt( atof(s1)/atof(s2), p, s3 );
}
```

CFUNCTION

provides access to C libraries from FlashBASIC.

Three libraries are defined. "unix.builtin" and "gm.builtin" are provided as part of D3. "user.builtin" is provided for user-written C routines.

If only the C programs defined in the "unix.builtin" libraries are used, the "cfunction" statement is optional. If user-defined or third party C programs are used, the "cfunction user.builtin" statement is required.

C program availablility is dependent upon the release of D3 used:

unix.builtin	Available on D3 Unix-based releases.
user.builtin	Available on D3 Unix
gm.builtin	Available on all releases of D3.

The "user.builtin" table is maintained with the "addbi" TCL command.

Syntax

cfunction name {, name ...}

Example

```
cfunction unix.builtin
cfunction user.builtin, gm.builtin
```

Compile stamps

contains descriptive internal information about programs compiled for FlashBASIC.

When a program is compiled, information is placed in the dictionary level of the file in which the source program resides. The structure of the information is as follows:

Attr Description

0 item-id

1 CC

2 starting frame number of object code

3 Pick/BASIC object frame count

4 date, internal, when last compiled

5 time in seconds, internal, when last compiled

6 port number of who compiled program last

7 user who compiled program last

8 account name that compiled program last

9 release version from where program was last compiled

10 options used when last compiled, alphabetically

11 bytes in Pick/BASIC object code

12 FlashBASIC object frame count

Example

```
ct dict bp prog
001 CC
002 740380
003 1
004 9424
005 37743
006 12
007 dm
008 qa
009 6.1.0.A5
010
011 43
012 0
 This program was compiled October 19, 1993 at 10:29:03 a.m., and
takes up 43 bytes of 1 frame. It was compiled on port 12 by user "dm"
in the account "qa" with no options and was not Flash compiled.
```

conv.expression

any expression which evaluates to a valid processing code (conversion).

Syntax

ICONV(string.expression, conv.expression)

OCONV(string.expression, conv.expression)

Example

```
print oconv(time(),'mth')
Prints the system time using the "mth" AQL processing code as a
literal string.
conv.expression = 'd2-'
d = oconv(date(),conv.expression)
The "conv.expression" is the AQL date processing code stored as a
string in the variable, "conv.expression".
preci= '2'
scale = '4'
money = oconv(income,'mr':preci:scale)
The "conv.expression" is the compound string expression:
```

```
'mr':preci:scale
which results in the string, "mr24".
```

data representation

data types employed in FlashBASIC.

There are primarily two types of data: numeric and string.

Numeric data consists of a series of digits and represents an amount, such as 123.

String data consists of a set of characters, such as "Jan Smith".

A numeric value in the range of: {+/-}140737488355327*(10(-p)) where "p" is the precision value between 0 and 9, is stored as numeric data. A numeric value outside this range is stored as string data.

In order to maintain maximum accuracy, when an operation causes an overflow condition, a precision of up to 18 digits to the right of the decimal is kept internally.

Results of addition (+), subtraction (-), division (/), and multiplication (*), and remainder (/) operations that exceed this range of {+/-}140737488355327*(10(-p)) have no limitation on their numeric range.

For the comparison operators such as less than (<), equal (=), greater than (>), and the arithmetic functions int(x), abs(x), mod(x,y), and rem(x,y), there is no limitation, since the result is limited to a 1 or 0.

The FlashBASIC functions "ln", "sqrt", "pwr", and "exp" can be as large as

(+/-)140737488355327x(10(-p)).

A string may contain up to 2 gigabytes of characters in D^3. A string constant is represented by a set of characters delimited by single quotes ('), double quotes ("), or backslashes (\). If any of the string delimiters are to be part of the string, then one of the other delimiters must be used to delimit the string.

Numeric and string data may be represented within the FlashBASIC program as either constants or variables. A constant, as its name implies, has the same value throughout the execution of the program. A variable may contain data types other than string and numeric data (root, active list, file.variable), but these representations are limited to specific statements.

default file variables

a brief discussion on the use of default file.variables in FlashBASIC programs.

The following group of statements makes use of a feature known as default file.variables: "clearfile", "delete", "matread", "matreadu", "open", "read", "readu", "readv", "readvu", "release", "select", "write", "writev", "writevu", "matwrite" and "matwriteu".

Most experts agree that this feature should be avoided, but here is a quick discussion of how they work.

When a file is opened with an "open" statement, it is usually assigned to a file.variable for referencing the file later in the program with any of the above statements. This takes the general form:

open "customers" to customer.file else ...

The operative word here is "to". It assigns the actual location (represented internally as a base, modulo and separation) to the variable called "customer.file", in this case. Later in the program,

when an item is read from the file, the file.variable appears in the appropriate form of a "read" statement, as in the form:

read customer.item from customer.file else ..

Here, the operative word is "from". With "default" file.variables, nothing is explicitly assigned during the "open" statement, as in the form:

open "customers" else ..

Hence, any subsequent attempt to read from or write to the file does not require the file.variable reference, as illustrated in the form:

read customer.item else ...

There may be only one default file.variable in a FlashBASIC program. Any subsequent file needed for input or output must have an explicitly assigned file.variable during the "open" statement, or it takes the place of the default file.

dimensioned array

a table where the contents of the elements can be altered (replaced) but the position of each element is fixed. In other words, new elements cannot be inserted and existing elements cannot be deleted.

Any alteration of elements in a "dimensioned" array does not directly affect the other elements of the array. Again, each element is a separate variable. In contrast, "dynamic arrays" are strings contained in single variables. Any change to the array requires that the entire string be rebuilt.

Since any element in a "dimensioned array" can be a string variable, and any string can be treated as a dynamic array, a "dimensioned array" can be an array of "dynamic arrays". For example:

print rec(7)<1,vmc>

dynamic array

a string containing any of the three main system delimiters; attribute marks, value marks, and subvalue marks. It is called "dynamic" for its ability to insert and delete array elements without having initially declared the array size.

When referencing a dynamic array, it may be treated as any other variable, with or without a subscript.

Example

```
s = ''
Sets the entire array to null.
s<3>=''
Sets the third attribute to null.
```

error condition

in FlashBASIC or FlashBASIC, the process state that occurs when a program encounters a condition which is unresolvable, forcing an abort to the FlashBASIC debugger. For example, if a reference is made to a dimensioned array subscript less than 1 or greater than the number defined in the "dim" statement a non-recoverable error occurs .

file.variable

is the symbolic way to access a file after being previously opened in a FlashBASIC progam. "file.variables" are assigned with the "open" statement.

"file.variables" contain the assigned file pointers and are used by subsequent FlashBASIC file input and output statements.

Once explicitly assigned, all subsequent "{mat}read{u}", "readv{u}", "{mat}write{u}", "delete", "clearfile", "release", "select" and "close" statements reference the given file by its "file.variable".

Unlike numeric and string variables, file.variables cannot be output with a "print" or "crt" statement. Nor can they be used in a string or arithmetic expression, nor displayed in the FlashBASIC debugger. "file.variables" can be assigned to other variables in a standard assignment statement. They may, however, be copied to other variables, in the form:

file.variable.b = file.variable.a

Syntax

open filename to file.variable ...

read variable from file.variable...

write variable on file.variable...

Example

```
open "invoices" to inv.f else ...
This opens the "invoices" file to a "file.variable" called "inv.f".
Throughout the rest of the program, "inv.f" is used for any reference
to the "invoices" file.
readu item from inv.f,"cust.345" else ...
An item is read from the "inv.f" (a file.variable opened in the first
example).
old.invoice.file = inv.f
Both "inv.f" and "old.invoice.file" are file.variables which
reference the same file.
```

FlashBASIC

converts Pick/BASIC source code into a list of binary instructions called object code.

The "run" verb initiates a program called an interpreter which reads these instructions one at a time and executes the desired actions. Taking this interpretive approach provides swift translation from the Pick/BASIC source, platform independence, small object size, and reasonable performance.

If better performance is desired, the Pick/BASIC compiler can produce "FlashBASIC", or native assembly code from a standard Pick/BASIC object. This FlashBASIC code may subsequently be run in the Pick environment in the same manner as a regular Pick/BASIC program with the exception that FlashBASIC code runs significantly faster.

The "o" option with the "compile" or "basic" verbs invokes FlashBASIC. When complete, the native code produced is appended to the standard interpreted object code. Because FlashBASIC generates machine-specific code, such code must be recompiled when moved to a different platform. Furthermore, FlashBASIC code cannot call interpreted code, or vice-versa. In other words, if a mainline program is compiled with the "o" option, all of its subroutines or "entered" modules must also be created as FlashBASIC.

Choosing a FlashBASIC Level:

The "o" option may be followed by a number, from 1 to 9, which indicates the FlashBASIC (optimization) "level". If not specified, the level defaults to 1, which gives a large performance increase while minimizing compile time and object size. Higher levels increase object size by as much as 100% over that achieved with level 1, and increase compile time exponentially. Furthermore, levels above 1 yield relatively small performance increases for most applications. High levels also tend to use very large amounts of Unix swap space. If the system runs out of swap space, FlashBASIC retries with a lower level. Line numbers (for error messages) and the state of variables within the debugger are also not guaranteed when using a level greater than one. Finally, the exact behavior of a specific level above one is undefined and may be different on different platforms or releases.

Pick Systems advises running routines with various levels to see if the run-time performance increase is worth the compile-time and object-size trade-offs. The best candidates for higher levels are small modules (less than about 200 lines) which have little or no string manipulation and lack significant amounts of screen or disk I/O. Using a level of 9 can cut in half such programs' run time as compared to their speed at level one, while not taking an unreasonably long time to compile.

Producing FlashBASIC without source:

FlashBASIC may be used on an application lacking source code by using the "w" option when compiling (see "compile" for specific instructions.) Therefore, an applications vendor need not ship source code or even a new version of the object code. Customers equipped with FlashBASIC can compile their current applications "as-is". However, it is suggested that future applications be shipped with a Pick/BASIC program, PROC, or macro, that automatically produces the FlashBASIC code at the customers site.

Shipping FlashBASIC code is only recommended when a vendor wishes the code to run on a single platform, and when media size is not an issue.

Compiler Options for use with FlashBASIC:

b Array bounds checking is off by default with FlashBASIC. (It is always generated in the interpreter.) Using the "b" option while compiling adds this feature to the FlashBASIC code, but increases compile time by about 20%, and slows down run-time array accesses.

c Eliminates EOL's (end-of-lines) in both FlashBASIC and interpreted code. Using this with FlashBASIC decreases compile time by 20%, and decreases run-time slightly. However, this prevents the ability to single-step in the debugger.

f Uses floating point arithmetic

h By default, the FlashBASIC compiler attempts to create native code that can be shared by all users. Using the "h" option forces the module to be loaded locally, and is typically used for modules that will be used by only one or two people at any given time.

k This option, when used without the "h" option, forces shared code to remain loaded until the machine is shut down. This option should be used for programs needed by most users of the system. It reduces memory usage and load time drastically when applied to multi-user situations. See the "shpstat" program for monitoring the status of shared programs.

o Triggers the FlashBASIC compiler.

w This option is used when producing FlashBASIC code without source.

Performance Tips for FlashBASIC:

To obtain the best possible run-time performance and compile-time, Pick Systems advises breaking up applications into many small modules. FlashBASIC removes the traditional run-time overhead of large numbers of call's and is able to create more efficient code when the modules are smaller. Furthermore, an application broken into several small modules takes considerably less time to compile than the same application stored in a single item.

The Pick/BASIC "@(x,y)" function speeds up from 10-20 times if the "term" definitions are compiled with the "o" option. For instance, if the current terminal is an IBM 3151 terminal, the "term ibm3151 (co" command compiles and produces FlashBASIC code for the terminal driver. This creates a hidden module for use with FlashBASIC code. For best results, use this feature with high baud rates and/or fast display devices.

Avoid user exits. Standard Pick/BASIC routines compiled with FlashBASIC will probably run just as fast, and possibly faster, than the same routines coded in virtual assembly.

Avoid disk I/O. If an application has read-only tables, it is best to read them all into memory at the beginning of the program rather than reading them piece-meal throughout execution. When reading an item, do a "matread" into a dimensioned array, rather than using several readv statements.

Notes:

Pick/BASIC source code is compiled to a tokenized object code which is then interpreted when run. For D3 Unix, Pick/BASIC source code may be compiled in the traditional way, or it may be optimized or *flash*ed. When optimized, the Pick/BASIC source code is translated to C source code, and then compiled using the host machine's C compiler, producing machine-specific assembly language object code.

For D3 NT, Pick/BASIC source code may also be compiled in the traditional way, or it may be optimized or *flash*ed. In this sense, though, the Pick/BASIC source code is compiled to an "optimized" tokenized object code which is then interpreted when run. This object code runs considerably faster than the original tokenized object code, but not quite as fast as the assembly language code of the C compiler.

FlashBASIC differences

describes the differences between Pick/BASIC and FlashBASIC.

Platforms Supported

FlashBASIC is supported on the AIX, the DG/UX, and the SCO Unix.

Mixing FlashBASIC and Interpreted Code:

If a mainline program is compiled with the "o" option, all its subroutines must also have been produced with the FlashBASIC compiler. (see "compile").

Access:

FlashBASIC code is never run when called directly from a dictionary definition.

The "chain" statement:

The Pick/BASIC "chain" may be used freely to transfer control to other programs, but variables will NOT be restored if a "run" with the "i" option is attempted later. This restriction holds if a FlashBASIC application is involved with the "chain" in any way.

"%" (Percent) calls:

FlashBASIC currently supports calling C routines, but NOT assembly routines with the percent call. (see the various functions in Pick/BASIC that begin with "%").

FlashBASIC code allows a maximum of 62 parameters on an external subroutine call.

The "rnd" function:

The Pick/BASIC Debugger:

The FlashBASIC debugger does not support logical conditions on breaks.

The FlashBASIC debugger does not allow any arguments on the "g" command.

The FlashBASIC debugger does not actually do anything when it prints "object verifies" on the "$" command.

The FlashBASIC debugger does not allow continuing after an abort.

The practice of patching variables in the debugger after a warning message does not work.

Arithmetic:

String math is not supported by FlashBASIC.

Higher arithmetic functions may give slightly different results in FlashBASIC. Addition, subtraction, multiplication, division, and remainder (modulo) will give the EXACT results with FlashBASIC as R83. Otherwise, the FlashBASIC compiler results are based on the host machine's internal arithmetic. Any arithmetic functions not listed above use the host machine's arithmetic. If using the (f option, all arithmetic operations use the host machines arithmetic.

Note also, that comparison will not function properly in FlashBASIC if two numbers are compared whose total precision is equal to or excedes 14 digits. Note that one must subtract the current precision from this so that the total precision limit is 10 digits at precision 4.

Parameter Passing:

FlashBASIC passes parameters by reference, rather than copy-restore (except for constants and expressions which are passed by copy). This could cause some problems in certain exception cases. For example, if a subroutine is called with the statement "call x(y, y(10))", where "y" is an array, and within that subroutine y(10) is modified, the current interpreter may restore the value differently than in the compiler.

FlashBASIC allows a maximum of 62 arguments on subroutines.

FlashBASIC does not complain when the number of parameters passed does not equal the number of parameters contained in the subroutine. However, this is NOT supported and can cause subsequent program failure and data corruption.

Arrays:

Array bounds checking is absent in FlashBASIC by default. The "b" option invokes this when compiling. This must be used with caution, as it is easy to destroy other program data due to application bugs.

FlashBASIC error logging

describes how to record compiling errors when using FlashBASIC.

When compiling ("flashing") Pick/BASIC programs with the "o" option, the compiler automatically logs all compilation errors, if there is a data section called "$log" present in the

user's Pick/BASIC program file. The log is updated ONLY when errors occur. Each log entry's id is the same as the id of the item being compiled.

The first attribute of the entry consists of the time and date that the error occurred as well as the "phase" of compilation where the error occured. Other attributes may contain additional undefined information which may contain Unix error messages.

Errors logged as "phase 0" errors are problems detected by the standard Pick/BASIC compiler.

Errors logged as "phase 1" and higher are FlashBASIC compilation errors.

Errors occuring higher can indicate an installation problem or lack of some resource, such as swap space. In these cases, attributes 2 and higher contain more exact error reporting.

Example

```
To enable logging for a file called "bp", type the following:
create-file bp,$log 7
From then on, all compile's which use the "o" option will log errors
into the "bp,$log" file and may be displayed by typing:
   ct bp,$log
or
   list-item bp,$log
or
   sort-item bp,$log
```

FlashBASIC performance

discusses three methods of increasing performance: replacing dynamic arrays with dimensioned arrays, replacing common with named common, saving screen images in a variable.

Replacing dynamic arrays with dimensioned arrays is especially effective when an item is read from a file, and manipulated extensively afterwards. If the number of attributes in the item is known, the "matread" statement may be used to read the dynamic string on disk into a dimensioned array. If the size is not known, or subject to change, read the item into a dynamic array, then use "dcount" to get the number of attributes, redimension a dimensioned array to the desired size, and assign the dynamic array to the dimensioned one. For example:

equ am to char(254)

read xx from "big.item"

size = dcount(xx,am)

dim stat(size)

stat = xx

Another great speed increase can be realized by using named commons to store data that must be used between various applications. See "common" for more information on this topic. If named commons are used for this purpose, all applications involved must be compiled EITHER with the interpreter or with FlashBASIC.

If an application does large amounts of screen output, it is suggested that screens be built as large strings at the beginning of the program. This way, a screen refresh is simply a "print" of a string variable.

x=@(-1):@(1,1):"Name:":@(1,2):"Number:":@(1,3):"Address:"

print x

format strings

See: masking

functions

elements in FlashBASIC language expressions, returning a single value in place, as a function of the arguments. A function can be used anywhere a variable or expression is used.

Syntax

function({argument{,argument{,..}}})

Example

```
int.date=iconv(ext.date,"d")
if data='' then kc=1 else kc=dcount(data,am)
```

global common

named common, FlashBASIC variable space used to store variables and arrays in the 'id'entified area which is only initialized at logon.

Global common space is common for all levels of the same process (i.e., level pushes, executes, filetime bridges & indexes, etc.)

Variables and|or arrays can be used for communication between programs and subroutines providing inter-program communication.

Syntax

COMMON /id/variable|array{,variable|array...}

Example

```
common /person/ name, company
```

id.expression

is a string expression which evaluates to an item-id, for use in all types of "read" and "write" statements.

Example

```
read invoice.item from invoice.file,"12345" else...
In this example, the id.expression is "12345".
input id
read record from file,id else...
In this example, the value entered into the variable "id" is used as
the id.expression.
read lastid from ctlfile,'lastid' then
 write item on datafile,lastid+1
 write lastid+1 on ctlfile,'lastid'
end
Both the literal string 'lastid' and the arithmetic expression
"lastid+1" are id.expressions.
```

LOCKED

used when the requested item is locked and waiting is not desired.

See the "read" statement.

Syntax

LOCKED statement.block

Example

```
read cust.item from fv.cust,id locked print id:" is locked by port
":system(0)
```

logical expressions

application of logical (Boolean) operators to relational or arithmetic expressions. The result of an operation has two states; "true" and "false". Logical expressions are considered false when equal to zero, and are considered true when non-zero.

Logical operators have the lowest precedence and are evaluated after all other operations have been evaluated. If two or more logical operators appear in an expression, the leftmost is performed first.

Logical operators act on their associated operands as follows:

a OR b is true (evaluates to 1) if a is true or b is true.

a ! b is false (evaluates to 0) if a and b are both false.

a AND b is true (evaluates to 1) only if both a and b are true.

a & b is false (evaluates to 0) if a is false or b is false or both are false.

The "not" function negates the effect of a logical expression:

not(a or b) is true (evaluates to 1) if a is false or b is false.

not(a and b) is true (evaluates to 1) if a and b are false.

Example

```
if x then...
The "then" clause is taken when "x" is a non-zero numeric. Whether x
is a numeric constant, or an ASCII string, any non-zero numeric value
of "x" is true.
word = "apple"
chr = word[1,1]
print "A":
print str("n",chr="a" or chr = "e" or chr = "i" or chr = "o" or chr =
"u") :
print " ":word
This prints "An apple", because chr = "a".
visit.date = customer.item<5>
if visit.date then ...
In this example, the "then" clause is executed if "visit.date"
evaluates to a non-zero numeric.
if x > 1 and x < 10 then...
The "then" clause is taken if "x" is between 1 and 10, exclusive.
if (x > 1 and x < 10) or (x >= 100 and x <= 200) then...
The "then" clause is taken if either "x" is between 1 and 10,
exclusive, or between 100 and 200, inclusive.
if not(print.flag = "n") then printer on
The "printer on" statement is executed if the value of "print.flag"
is not "n".
```

masking

formatting expressions using numeric masks and format masks.

Where the mask expression is:

{r|l|d{n}{s}{z}{,}{c}{$}({{{f},l}...})

Where:

r|l|d (r)ight or (l)eft justified. The default is left. The "d" specifier functions identically to the "r" and is provided for compatibility reasons only.

n Number of digits after decimal. This is a single numeric digit that specifies the number of digits to print following the decimal point. If "n" = 0, the decimal point is not output. If the number of digits following the decimal point is greater than the number specified by "n", the output is rounded.

s Scaling factor. This is a single numeric digit that specifies to 'descale' the value by the 's - precision' power of 10. For example, an "s" value of 2 changes 1000 to 10.

z (A literal "z"). Suppresses leading zeros.

, Inserts commas between every "thousands" position of the value.

$ Appends dollar sign to beginning of value.

f "fill" character

l length

c signcode:

c Outputs "CR" after negative values. Two blanks follow positive or zero values.

d Outputs "DB" after positive values. Two blanks follow negative or zero values.

e Encloses negative values within a "<" and ">". A blank follows positive or zero values.

m Places a minus sign to the right of negative values. A blank follows positive or zero values. Ordinarily, the minus sign appears to the left of negative numbers.

n Suppresses the minus sign on negative values.

Special fill characters:

%n fills with "n" zeros.

#n fills with "n" blanks.

*n fills with "n" asterisks.

Any other character, including parentheses, may be included in the fill mask. The characters are output exactly as they appear in the mask. If a dollar sign is placed outside of the format mask, it is output just prior to the value, regardless of the filled field. If a dollar sign is used within the format mask, it is output in the leftmost position regardless of the filled field. Parentheses are ignored if they are the first character of a mask.

The numeric mask code controls justification, precision, scaling, and credit indication.

The format mask code controls field length and fill characters. The entire format string is enclosed in quotation marks. If a format mask is used, it should be enclosed in parentheses within the quotation marks.

The entire format string may be used as a literal, or it may be assigned to a variable. A format string literal can immediately follow the string it is to format. A format string variable must be separated by at least one space from the string it is to format. The format string may also be used directly in conjunction with the print statement.

The following mask codes must be entered in the order they are listed.

Mask Description

d{ } If specified before any other mask, this specifies standard date conversions, and any other mask code is ignored.

c If specified before any other mask, the various "character" masks may be specified (like "cu" or "cl"), just like the oconv(string,"mcu").

Syntax

string.expression mask.expression

Example

```
print x "l(#12)"
This value in "x" is left justified, in a field of 12 characters.  If
the value of "x" is the word "test", the output looks as follows:
(spaces are represented by lower case "b")
testbbbbbbbb
print y "r(#15)"
"y" is printed right justified, fill space, in a field of 15
characters. If the value of "y" is the word "hello", the output looks
as follows: (again, "b" represents spaces)
bbbbbbbbbbhello
print z "r24z,e$(*12)"
Unlike the "mr2" conversion used in an "oconv" or "iconv" function,
this masks the value of "z" based on the current value of "precision"
in FlashBASIC runtime.
If the value of "z" is 78657767 and the current "precision" is 4,
then the amount of scaling is the requested scaling factor minus the
precision. Because the scaling factor is equal to the precision, no
scaling takes place. After evaluating the scaling, the mask then adds
two zeros after the decimal point, and justifies the result in a
field of asterisks. Since the mask is not specified with a sufficient
amount of asterisks to contain the result, only the 12 rightmost
characters are displayed.
If the value of "z" is 78657767, the output looks as follows:
,657,767.00
print oconv(date(),"d2/") "r(#10)"
This example outputs the external format of the current system date,
right justified, fill space, in a field of 10 characters.  The output
is as follows:
bbmm/dd/yy
```

named common

a global, common, FlashBASIC variable space used to store variables and arrays in the area identified by "id" . Named common space is initialized only at logon and released at logoff.

Named common space is common for all levels of the same process (i.e., level pushes, executes, filetime bridges & indexes, etc.)

Variables and|or arrays can be used for communication between programs and subroutines providing inter-program communication.

Syntax

COMMON /id/variable|array{,variable|array...}

Example

```
common /expenses/ emp,date,category
```

non-fatal error condition

a type of error that occurs when a FlashBASIC program encounters a condition which is resolved by the FlashBASIC runtime package, although not to the specific need of the program.

The most common occurrence of this is the "variable has not been assigned a value, zero used" message. This particular instance occurs if a reference is made to a variable on the RIGHT side of an "=" (equal) BEFORE being referenced (assigned) on the LEFT side of an "=".

Example

```
dim array(10)
for i = 1 to 10
array(i) = x
next i
This results in 10 non-fatal error messages:
[B10] in program "name", Line n:
Variable has not been assigned a value; zero used.
```

nuclear tokens

Any function or expression may itself be an argument of another function or expression. The compiler evaluates expressions starting with the innermost set of parentheses.

Example

```
crt oconv(array(1)-array(2),"mr2")
name = field(trim(sentence),' ',2)
```

null, evaluation

represents an empty string or decimal zero value depending on the context, string or arithmetic operation.

num.expression

any expression which evaluates to a number.

Example

```
x = 1
x = '1'
y = x:'0'
If "x" contains a numeric expression, "y" is also a valid numeric
expression.
```

ONERR

identifies the statements to execute when an error occurs during commands which perform operations on peripheral storage devices.

The "system(0)" function contains the error message associated with the "onerr" condition.

The "onerr" condition is used in the same construct as the "then/else" construct.

Either an "onerr" clause or an "else" clause is allowed in a statement, but not both. They are similar, in that they are taken when the peripheral operation takes a decision-point or failure path. The "onerr" form provides a great deal more functionality, but at a higher cost. When the "onerr" clause exists, all media handling must be handled by the program. This means, for example, that the program is responsible for handling an "end of reel" condition, thus the program would have to prompt the operator for the next reel. By contrast, the "else" clause would handle the same situation by passing through the standard system routine to "mount next reel and press 'c' to continue".

Syntax

ONERR statement.block

Example

```
readt tape.rec onerr
  crt "oops. we've got a system(0) error of " : system(0)
end
In this example, the "onerr" path is taken in the event of any
abnormal condition, and it simply displays the value of system(0).
readt tape.rec onerr
    condition = system(0)
    begin case
    case condition = 1
         print "tape is NOT attached..."
    case condition = 5
         print "process end of reel"
    case condition = 6
         print "tape is write-protected"
    end case
end
This example illustrates how processing for an error could be divided
out based upon the error which occured.
```

pattern matching statements

See: match

precedence

defines the set of rules imposed upon the evaluation of components of an expression that are not otherwise overridden by the presence of parentheses.

Based on the operation, expressions are evaluated from left-to-right (left-associative), from right-to-left (right-associative), or with no association (non-associative). For example, multiplication and division are left-associative. The following arithmetic expression evaluates left-to-right:

x = 9 / 3 * 2

This first divides 9 by 3, yielding 3, which is then multiplied by 2, yielding the result 6. This expression evaluated as a right association gives the result 1.5.

Logical operations are non-associative. For example, the following expression does not compile:

x = 1 < x < 3

Adding parentheses makes it a valid expression:

x = (1 < x) < 3

Adding the parentheses, however, does NOT give the results of "(1 < x) AND (x < 3)".

x = (99 < 1) < 3

x = (0 < 1) < 3

The above two examples both evaluate as 1. In the first example, (99 < 1) evaluates as 0, and 0 < 3 evaluates as 1. In the second example, (0 < 1) evaulates as 1 and 1 < 3 evaluates as 1.

The following is a table of operations, precedence and associativity.

Sym	priority	operation	association
()	0(high)	parenthesis	none
[]	1	substring extraction	none
<>	1	dynamic array ref.	none
^	2	exponentiation	left
**	2	exponentialtion	left
*	3	multiplication	left
/	3	division	left

```
\          3         remainder              left
+          4         addition               left
-          4         subtraction            left
+          4         unary positive         right
-          4         unary negative         right
mask       5         string masking         left
cat        6         concatenation          right
:          6         concatenation          right
eq,  =     7         logical equal          none
ne,  #     7         not equal              none
<>         7         not equal              none
lt,  <     7         less than              none
le,  <=    7         less than or =         none
gt,  >     7         greater than           none
ge,  >=    7         greater than or =      none
match      7         pattern match          none
matches    7         pattern match          none
and,  &    8         logical and            left
or,  !     8         logical or             left
```

Expressions are evaluated in order of precedence unless placed within parentheses. 10+2*10 is evaluated as 30. Expressions within the innermost parentheses are evaluated first. (10+2)*10 evaluates as 120.

relational expressions

see "relational operators".

reserved words

The rules for reserved words are:

- Intrinsic function names cannot be dimensioned array names, but can be variable names.

- Special tokens cannot be used as variable names. (i.e., "and", "do", "else",..)

- Any other word can be both an array or variable name.

sc.expression

an expression that evaluates to a valid subvalue count ("sc").

Any expression which evaluates to a valid numeric value is allowed.

Example

```
customer.item<2,1,3> = "test string"
In this statement, "3" is used as the sc.expression.
```

secondary

see "readnext".

simple variable

See "variables".

statement labels

used as the destination of a "goto" or "gosub" statement.

A statement label consists of a numeric or alphanumeric string of characters placed at the beginning of any FlashBASIC statement or by itself on a line in the program.

FlashBASIC requires labels only on those statements used as destinations for "goto" or "gosub" statements.

Numeric statement labels can be any of the following formats:

10 1 100.5 01 001 10.02

Each of the above labels is unique.

Alphanumeric statement labels must begin with an alpha character. Any subsequent character in the label may be alphabetic or numeric. An alphanumeric statement label must be followed by a ":" (colon). If a statement is to be placed on the same line, it must be separated from the "statement label" by one or more spaces.

Since D3 allows null statement lines, a statement label can stand by itself on a line.

Syntax

statement.label{:} {FlashBASIC statement{s}}

Example

```
numeric statement label:

1000 * start main loop
alphabetic statement label:
mainline:
```

statement.block

one or more statements that follow a FlashBASIC decision-path related token such as, but not limited to, "then", "else", "locked" and "onerr".

"am" is used in the syntax to represent an attribute mark. It is interchangeable with the ";" (semi-colon).

Syntax

[am | ;] statement {[am | ;] statement...} end or statement {; statement ...}

Example

```
readt tape.rec then
   crt "grabbed record"
   .
   .
end
Which is the same as:
readt tape.rec then ; crt "grabbed record' ; . ; .
In the single-line form, "end" is not allowed.
readt tape.rec onerr crt "error code: " : system(0) ; stop
```

statements & functions

differences between statements and functions in FlashBASIC, a working definition of variable, constant, labels, blanks, and program formatting.

If the syntax requires that the instruction be followed by a set of parentheses (optionally containing an argument or arguments), then it is a function. For example, the following are functions: "abs(num.expression)", "rnd(num.expression)" and "iconv(string.expression, conv.expression)".

Any instruction that does not have to be followed by a set of parentheses is most often a statement. For example: "print x", "input y", "execute sentence".

There is at least one peculiar exception to the above rule. One variant of the "locate" statement is considered a statement, even though its syntax looks like it should be a function.

Another feature of statements is that they tend to "stand alone". For instance: "execute sentence" or "print array<10,1>".

Functions, by contrast, do not stand alone. They are either used as part of an expression within a statement: "raise = rnd(20)", or output immediately: "crt int(amount)", "print abs(balance)".

A FlashBASIC program is composed of statements made up of FlashBASIC commands, variables, constants, expressions, and functions.

FlashBASIC statements may contain arithmetic, relational, and logical expressions. These expressions are formed by combining specific operators with variables, constants, or FlashBASIC intrinsic functions.

The value of a variable may change dynamically throughout the execution of a program. A constant, as its name implies, has the same value throughout the execution of a program.

An intrinsic function performs a pre-defined operation on the supplied parameter(s).

Normally, within a FlashBASIC program, each physical line contains only one statement. This is done for ease of reading and tracking of logic.

More than one statement may be placed on a program line by separating each statement with a ";" (semicolon). Statements which end with "then" or "else" on a line create a block or multi-line structure. See the "if" statement for details on "then...else" statements.

Labels may be placed at the beginning of any FlashBASIC statement. A statement label may be numeric or alphanumeric. See "statement labels".

Blank spaces appearing in the program line which are not part of a literal are ignored. Blank spaces and blank lines may be used freely within the program for purposes of appearance.

string expressions

a string constant, a numeric constant, a variable with a string or numeric value, a substring, a concatenation of string expressions, an intrinsic function, or a format mask.

Concatenation of strings is specified by a colon (:) or by the "cat" operator. The "cat" operator must be preceded and followed by blanks.

The result of a string expression is an ASCII character string.

String expressions may include arithmetic expressions. The results of any numeric function or arithmetic expression is converted to an equivalent ASCII string.

Example

```
print "the total amount is " : amount
The string expression consists of a literal string, "the total amount
is ", followed by the contents of the variable "amount".
line = "the total amount is " : unit.price * quantity
The results of the arithmetic expression "unit.price * quantity" is
converted to ASCII and concatenated to the literal string.
```

string.expression

any expression which evaluates to, or can be converted to a string of characters.

Example

```
name = "Pick Systems"
In this example, "name" is a string expression.
```

substring assignment

allows assigning a substring to a variable

"beg.pos.expression" indicates the starting position in the expression.

If "beg.pos.expression" is 0, the expression is inserted at the beginning of the target string.

If "beg.pos.expression" is less than 0, blanks are inserted between the expression and the string for a length equal to the absolute value of "beg.pos.expression".

If "beg.pos.expression" is greater than the length of the string, blanks are appended to the end of the string to account for the value of "beg.pos.expression" prior to the expression being added to the result.

"len.expression" is the number of characters to overlay within the string expression.

If "len.expression" is 0, the expression is inserted prior to the starting character position within the string.

If "len.expression" is less than 0, then "len.expression" defaults to 0.

If "len.expression" is greater than 0, the corresponding number of characters in the string are overlayed.

If "len.expression" is greater than the number of characters in the string, starting from the starting position count, then "len.expression" defaults to the number of characters within the string, minus the starting position count.

If the length of the "substring.expression" to overlay is greater than the "len.expression", the additional characters are included in the resulting expression. Only the number of characters specified in "len.expression" are overlayed. The additional characters are inserted prior to the next position in the original expression.

Syntax

variable[beg.pos.expression, len.expression] = expression

Example

```
string = "xbcd"
string[1,1] = "a"
This assigns the letter "a" to the first character of the "string"
variable. After assigning, "string" contains "abcd".
string = 'abcd'
string[2,2] = 'xxx'
This results in the string, 'axxxd'. Notice that the additional
characters beyond the length of the overlay are inserted. The following
program tests various combinations of substring
replace.
string = "abcdef"
s = string ; s[ 0 , 0]= "xx"; crt s
s = string ; s[ 0 , 1]= "xx"; crt s
s = string ; s[ 1 , 1]= "xx"; crt s
s = string ; s[-1 , 0]= "xx"; crt s
s = string ; s[-1 , 1]= "xx"; crt s
s = string ; s[-1 ,-1]= "xx"; crt s
s = string ; s[ 7 , 0]= "xx"; crt s
s = string ; s[ 7 , 1]= "xx"; crt s
s = string ; s[ 7 ,-1]= "xx"; crt s
s = string ; s[ 8 , 1]= "xx"; crt s
The output of this program is as follows:
xxabcdef
xxbcdef
```

```
xxbcdef
xx abcdef
xx bcdef
xx abcdef
abcdefxx
abcdefxx
abcdefxx
abcdef xx
The following program creates "waves" of output:
string="#####"; s=string
for x=-3 to 7
  for j=-3 to 7
    s[x,j]='..'
    crt s
  next j
next x
end
```

substring expressions

extract or assign substrings by using the "[" and "]" characters.

Extraction: see "substring extraction".

Assignment: see "substring assignment".

Field store: see "substring field store".

Syntax

Extraction: variable[beg.pos.exp,len.exp]

Assignment: variable[beg.pos.exp,len.exp] = expression

Field store: variable[delimiter,beg.pos.exp,len.exp]

substring extraction

allows the extraction of a string from within another string.

The "beg.pos.expression" must evaluate to a number indicating the starting position within the string, and the "len.expression" must evaluate to a number indicating the number of characters to retrieve.

If the starting character specification is past the end of the string value, an empty substring value is selected. If the starting character specification is negative or zero, the substring is assumed to start at character one.

If the substring length specification exceeds the remaining number of characters in the string, the remaining string is selected. If the substring length specification is negative or zero, an empty substring is selected.

Syntax

string.expression[beg.pos.expression, len.expression]

Example

```
if response[1,1] = "y" then printer on
This tests the first character of "response" to determine if it is
the letter "y". If it is, the "printer on" statement is issued. See
"casing" for case-sensitive input.
area.code = phone[1,3]
This extracts the first three characters of the "phone" variable and
assigns it to the variable "area.code"
```

substring field store

change one or more fields or substrings within a character string variable which are delimited by a specific character.

"variable" is the string expression or variable.

"substring" is the substring or group of substrings (separated by a delimiter) to be substituted.

"delimiter" is the delimiter character separating fields or substrings in the original string.

"beg.fld.expression" is the starting field within the string for the substring to be placed.

"num.flds.expression" is the number of fields or substrings to be replaced in string.

The expression, "beg.fld.expression", determines the starting field to be changed by the operation. If "beg.fld.expression" is less than 1, 1 is assumed. If "beg.fld.expression" is greater than the number of fields within the string, the number of fields in the string is increased by adding null fields separated by the delimiter so that string has the length of "beg.fld.expression" fields.

Multiple substrings separated by the specified delimiter can be substituted in the original string.

The expression, "num.flds.expression", has different effects depending on whether it is positive, negative or zero:

If it is positive, then "num.flds.expression" fields in the string are replaced with "num.flds.expression" substrings. If "beg.fld.expression + num.flds.expression" is greater than the number of fields in string, the number of fields in the resulting string is increased to the value "beg.fld.expression + num.flds.expression", and the extra substrings are concatenated to the end of the original string. If "beg.fld.expression + num.flds.expression" is less than the number of fields in string but "num.flds.expression" is greater than the number of substrings specified, only the specified substrings are substituted and the remainder of the "num.flds.expression" fields in string are nulled. See example 1.

If "num.flds.expression" is zero, no fields in the string are deleted and the specified substring(s) are inserted into the string before the "beg.fld.expression"th field. See example 2.

If "num.flds.expression" is less than zero, "num.flds.expression" fields greater than or equal to "beg.fld.expression" are deleted from the string and the entire expression containing the substring(s) is inserted at that point in the string. See example 3.

Syntax

variable[delimiter,beg.fld.expression, num.flds.expression] = substring

Example

```
1) string = "a,b,c,d"
string[",",2,3] = "x,y,z"
This results in the string: "a,x,y,z"
string = "a,b,c,d"
string[",",4,2] = "x"
This results in the string: "a,b,c,x,"
string = "a,b,c,d"
string[",",2,2] = "x,y,z"
This results in the string: "a,x,y,d"
2) string = "a,b,c,d"
string[",",3,0] = "x,y"
This results in the string: "a,b,x,y,c,d"
3) string = "a,b,c,d"
string[",",3,-1] = "x,y"
```

```
This results in the string:  "a,b,x,y,d"
string = "a,b"
string[",",4,-5] = "x"
This results in the string: "a,b,,x"
string = "a,b,c,d"
string[",",2,-2] = ""
This results in the string: "a,,d"
```

THEN | ELSE statement.block

are found in conjunction with conditional statements.

The most notable case is the "if" statement, which requires either a "then" or an "else" clause.

In all statements which allow or require "then" and/or "else", the "then" branch is taken if the operation is performed successfully, or a "true" is returned. The "else" clause is taken when the operation is unsuccessful, or "false" is returned.

For purposes of our documentation, we will call all the statements which allow or require "then" and/or "else" clauses "initiators".

The possible syntactic structures are:

1) initiator then statements else statements

example: if ans = "y" then crt "yes" else crt "not yes"

2) initiator then statements

example: if ans = "y" then crt "yes"

3) initiator else statements

example: if ans = "y" else crt "not yes"

When both "then" and "else" clauses are present, the "then" clause may be considered to be the initiator of the "else" clause. All other characteristics are identical, except that an "else" clause may succeed a "then" clause.

Within the "then" clause, the "then" token is the initiator, which requires a terminator. The clause may exist on one or more than one line. The nature of the terminator varies between these cases.

The Single-line form:

If the "then" clause is complete in one physical line, its terminator is a <return> or an "else" token.

1) initiator then statement{s}<return>

2) initiator then statement{s} else statement{s}<return>

3) initiator else statement{s}<return>

The form for the single-line clause:

then statement; statement; ... ;<return>

-or-

then statement; statement; ... ;else<return>

The syntax of the "else" clause is the same as that of the "then" clause, except that it may not be followed by another "else" clause.

The Multi-line form:

If the "then" clause spans more than one physical line, the "then" token must immediately be followed by an eol (end-of-line) character. This may be either a <return> or a ";" (semi-colon). In this case, the clause is terminated by "end" preceded by an eol character.

initiator then<return>

statement{s}<return>

statement{s}<return>

statement{s}<return>

end

There must be one or more statements between the "then" and its terminator. If there are several statements, they must be separated by eol characters. If the "then" clause is the one-physical-line clause, the eol character must be a semi-colon. If the then clause spans more than one physical line, the eol characters may be either (or both) <return>'s or semi-colons.

The form of the multi-line clause:

1) then eol statement eol statement eol eol end eol

2) then eol statement eol statement eol eol end else eol statement eol statement eol end

In this case, each eol character may be either a <return> or a semi-colon. This means that a multi-line clause may be contained in either one or more than one physical line. This case will normally appear as:

then<return>

statement<return>

statement<return>

statement<return>

end<return>

For program clarity, the statements are indented from the beginning of the initiator, and the "then" or "end else" are outdented to the beginning of the initiator.

Multiple end statements:

Multi-line statements may have more than one "end" statement. The "end" statement then becomes the initiator for the "else" condition as follows:

initiator then<return>

statement<return>

statement<return>

statement<return>

end else<return>

statement<return>

statement<return>

statement<return>

end

Multiple "end else" sequences may be used, provided that each ends with an "end" statement.

Historical Development of the then/else clause

Early in the development of Pick, some statements required "else" clauses, such as the "read" statement. This provided the ability of taking a separate "pathway" through the program if the item being read was NOT found.

Later, the "then" clause was introduced and the rules changed. Under this new provision, statements like "read" can have a "then" and/or an "else" clause, but one or the other had to be present.

The "read" statement still supports "else" and "then" clauses, but neither is required. Some statements, such as "if", still require one or the other.

Syntax

then statement.block

else statement.block

then statement.block else statement.block

update locks

see "retrieval locks".

variables

store a number, string, file descriptor, or select list, and may change dynamically throughout the execution of the program.

A program variable is similar to a mathematical variable found in formulas. Variable as a term first found widespread acceptance with the popularity of the FORTAN (FORmula TRANslation) programming language. The variable "X", for example, may be assigned the value 100 at the start of a program, and may then later be assigned the value "This is a string".

A numeric string, when used in an arithmetical expression, is converted to a numeric expression, and a numeric expression is converted to a string expression when used with a string operator.

A variable name identifies the variable. Variable names consist of an alphabetic character which can be followed by additional letters, numerals, periods, dollar signs and the underscore character. The length of a variable name may be from 1 to 48 characters.

Up to 214,748,363 variables may be defined.

vc.expression

an expression that evaluates to a valid value count ("vc").

Any expression which evaluates to a valid integer numeric value is allowed.

Example

```
crt customer.item<2,1> = "test string"
In this statement, "1" is used as the vc.expression.
```

zero, evaluation

Evaluation of zero.

!

indicates a remark (comment) statement or an "or" operator in a logical expression.

Syntax

! comment{s}

logical.expression ! logical.expression

Example

```
if x < 0 ! x > 10 then...
This illustrates the "!" as an "or" operator.
! mainline ....
This shows the "!" as a remark.
```

"

String delimiter.

#

"not equal to".

$CHAIN

continues FlashBASIC compilation in a different source item.

When the file.reference is omitted, the current file is assumed.

There is no limit to the number of "$chains", but any statements after a "$chain" command will be ignored.

Syntax

$CHAIN {file.reference} item-id

Example

```
bp myprog
003 equ bell to char(7)
004 equ am to char(254), vm to char(253), svm to char(252)
005 open "cust" to customer.file
006 $chain prog
bp prog
002 read item1 from customer.file,"01234" else print bell
```

$INCLUDE

Alternate method of "include".

Syntax

see "include".

$INSERT

Alternate method of "insert" (or "include").

$OPTIONS

sets compatibility options for the BASIC compiler.

By default, the compiler generates code compatible with the classic Advanced Pick product. The "$options" statement allows the use of new statements and functions as well as closer functional compatibility with other flavors of Pick. The following "tag"'s are currently available:

AP	Advanced Pick - This is the default setting and provides code compatible with previous releases (provided new features are not used).
DEFAULT	Default - This is the same as EXT and should be used as the new development default setting.

EXT	Extended - This tag provides as many statements and functions as are available and should be used for all new development. Note that because many new functions are allowed, that array references in existing code may need to be changed to some other name for proper compilation.
GA	General Automation - This is similar to the EXT option, but provides tighter GA compatibility with regards to the locate and match statements. It is suggested not to use this mode for new development as this setting will disallow any functions not provided by General Automation Pick.
IN2	IN2 - Currently the same as "EXT". Further compatibility changes may be made in the future to bring this setting closer to IN2 compatibility. Never use this setting for new development.
INFORMATION	Prime Information - Currently the same as "EXT". Further compatibility changes may be made in the future to bring this setting closer to Prime Information compatibility. Never use this setting for new development.
PICK	Pick System's R83 - Same as R83.
R83	Pick System's R83 - Currently the same as D3. Future development may make minor changes to allow slightly better run-time R83 compatibility. Never use this option for new development.
REALITY	Reality - Currently the same as "EXT". Further compatibility changes may be made in the future to bring this setting closer to Reality compatibility. Never use this setting for new development.

It is suggested to use the "$options" statement at the top of the desired modules. Note that the default setting is "AP", but this can be changed by setting the shell variable "@SYS.COMPILER" to one of the above tags.

It is illegal to use the "$options" statement multiple times within the same source item.

Syntax

$OPTIONS {tag}

Example

```
$options ext
x="abc"
print ereplace(x,"a","b")
"bbc"
```

%

the first character of a built-in C function called from FlashBASIC.

%ACCEPT

extracts the first connection on the queue of pending connections, creates a new socket and allocates a new file descriptor.

To compile successfully, the statement 'cfunction socket.builtin' must be included in the source code.

'socket' Is the file descriptor of the local socket returned by a previous call to the FlashBASIC C function '%socket'.

Upon successful completion, a value of 0 is returned in 'code', and the following FlashBASIC variables are updated:

'address' Originating address of the incoming call.

'port' Originating port number or the incoming call.

In the case of an error, a value of -1 is returned and the (FlashBASIC) function 'system(0)' is set to the value of 'errno'.

Syntax

code = %ACCEPT(socket, &address, &port)

Example

```
cfunction socket.builtin
include dm,bp,unix.h socket.h
* Create a socket
fd=%socket( AF$INET, SOCKET$STREAM, 0 )
* Bind the socket to a local Ethernet port.
* Use default address.
if %bind( fd, AF$INET, INADDR$ANY, 1024 )<0 then
  crt 'bind failed'; stop
end
* Wait for incoming connection
%listen( fd, 1 )
* Accept a connection
addr=0; port=0
fd=%accept( socket, &addr, &port )
crt 'Called by address ':addr:', port #':port
* Read data from the data link
%read( fd, buffer, 1024 )
```

%ALARM

instructs the alarm clock of the calling process to send the signal "SIGALRM" to the calling process after the number of seconds specified in "number".

The default alarm handler is null (i.e.: it just interrupts the process). If "number" is 0, any previously made alarm request is canceled. If a user-written built-in function changes the alarm signal handler, it remains in effect, even if the FlashBASIC program terminates, until the process is disconnected from the virtual machine.

The alarm signal handler can be assigned to any D3 command using the (TCL) "trap" command.

To differentiate between an alarm and another interrupt, the FlashBASIC program may have to check the time. See the example below.

Syntax

variable = %ALARM(number)

Example

```
* Set a timer
alarmend=time()+3
%alarm( 3 )
* Read from device
n=%read(fd, buffer, BUFSIZE)
if n<0 and system(0)=EINTR then
  * Read was interrupted
  if time() >= alarmend then
    * Was the timer
    stop "time out"
  end else
```

```
    * Not the timer. Disarm it
    %alarm(0)
  end
end else
  * Read completed. Cancel timer
  %alarm( 0 )
end
```

%BIND

binds a socket to a named resource. In case of a network, the resource would be an access point into the system (see the example below).

To compile successfully, the statement 'cfunction socket.builtin' must be included in the source code.

'socket' Is the file descriptor of the local socket returned by a previous call to the FlashBASIC C function '%socket'.

'addr.family' Specifies the addressing scheme used by the protocol. This field must match the address family used when creating the socket. Valid values are defined in the include: 'dm,bp,unix.h socket.h'.

'address' Legal values are defined in the include: 'dm,bp,unix.h socket.h'.

'port' Port number on the local host. The legal value for this field depends on the protocol. On TCP/IP, for example, valid port number are from 1024 to 32767. This field may not be applicable for all protocols.

Upon successful completion, a value of 0 is returned in 'code'. In case of an error, a value of -1 is returned and the (FlashBASIC) function 'SYSTEM(0)' is set to the value of 'errno'.

Syntax

code = %BIND(socket, addr.family, address, port)

Example

```
cfunction socket.builtin
include dm,bp,unix.h socket.h
* Create a socket
fd=%socket( AF$INET, SOCKET$STREAM, 0 )
* Bind the socket to a local Ethernet port.
* Use default address.
if %bind( fd, AF$INET, INADDR$ANY, 1024 )<0 then
  crt 'bind failed'; stop
end
* Wait for incoming connection
%listen( fd, 1 )
* Accept a connectionn
addr=0; port=0
fd=%accept( socket, &addr, &port )
* Read data from the data link
%read( fd, buffer, 1024 )
```

%CHDIR

changes the current directory to the directory specified by "string".

A "-1" (minus one) returned from this function indicates an error condition.

Syntax

variable = %CHDIR(string)

Example

```
if %chdir('/usr/pick') = -1 then crt 'errno=':system(0)
If an error condition is encountered in this example, the error
message is displayed from function "system(0)".
```

%CHMOD

changes the mode of the file "string" to the value specified in "mode".

The valid values for "mode" are in the include "mode.h". Combinations of the modes are obtained by adding several elementary mode values together.

Syntax

variable = %CHMOD(string, mode)

Example

```
include dm,bp,unix.h mode.h
file='/usr/pick'
n=%chmod(file, O$WRONLY)
```

%CHOWN

changes the owner ID and group ID of the file "string" to the values specified in "owner" and "group" respectively.

Syntax

variable = %CHOWN(string, owner, group)

Example

```
if %chown( "/tmp/ap.log", 2000, 0) < 0  then
  crt 'Cannot change owner'; stop
end
```

%CLOSE

closes the Unix file specified by "file.descriptor" returned by a previous call to %open(), %creat(), %dup(), %socket() or %accept().

All Unix files opened from a FlashBASIC program are closed automatically at main program termination. It is a safe practice, though, to close files explicitly.

Syntax

variable = %CLOSE(file.descriptor)

Example

```
if %close(fd) then
  crt 'Cannot close file'
  stop
end
Note the use of the implicit (void) casting by not including the
statement as part of an assignment. The return code of the close is
thrown away.
```

%CONNECT

requests a connection between two sockets.

To compile successfully, the statement 'cfunction socket.builtin' must be included in the source code.

'socket' Is the file descriptor of the local socket returned by a previous call to the FlashBASIC C function '%socket'.

'addr.family' Specifies the addressing scheme used by the protocol. This field must match the address family used when creating the socket. Valid values are defined in the include: 'dm,bp,unix.h socket.h'.

'host' Destination host name. This string must be known to the local network manager (normally defined in the '/etc/hosts/ Unix file). Internally, this string calls the BSD function 'gethostbyname' to read the '/etc/hosts' file.

'port' Port number on the distant host. Legal value for this field depends on the protocol. On TCP/IP, for example, valid port numbers are from 1024 to 32767.

Upon successful completion, a value of 0 is returned in 'code'. In case of error, a value of -1 is returned and the function 'system(0)' is set to the value of 'errno'.

The connection is closed when the socket is closed.

Syntax

code = %CONNECT(socket, addr.family, host, port)

Example

```
cfunction socket.builtin
include dm,bp,unix.h socket.h
* Create a socket
fd=%socket(AF$INET,SOCKET$STREAM,0)
* Connect to the distant host
if %connect(fd,AF$INET,"prod",1024)<0 then
  crt 'Connect failed'; stop
end
* Write data to it
msg="CONNECTED"
%write(fd,msg,len(msg))
* Terminate connection
%close(fd)
```

%CREAT

creates a new ordinary Unix file or prepares to rewrite an existing Unix file designated by "string" for write only. "mode" controls the file mode. Valid values of "mode" are in the include "mode.h". Combinations of the modes are obtained by adding several elementary modes together.

Syntax

file.descriptor = %CREAT(string, mode)

Example

```
include dm,bp,unix.h mode.h
* Create a unique temporary file
* read and write permission for user,
* read for group
fd=%creat( "/tmp/":system(19),
           S$IRUSR+S$IWUSR+S$IRGRP)
Note the usage of the plus operator to combine flags, where a regular
C program would have used an 'or'.
```

%DUP

returns a new file descriptor associated to the same file as the one associated with "file.descriptor".

Syntax

new.descriptor = %DUP(file.descriptor)

%FCLOSE

closes the stream specified by "stream" returned by a previous call to %fopen().

Note the "stream" is defined as a pointer to a character, instead of a pointer to a FILE.

Syntax

variable = %FCLOSE((char*)stream)

%FDOPEN

associates a stream with the "file.descriptor" obtained from %open(), %dup(), %creat().

See "%fopen" for a list of "types" and their descriptions.

Syntax

stream = (char*)%FDOPEN(file.descriptor, type)

%FGETC

returns the next character from the named input "stream".

Note that the character is returned as a number. Normal usage would override the type (see example below). The Unix "getc()" and "getchar()" cannot be used because they are macros, rather than functions.

Syntax

variable = %FGETC((char*)stream)

Example

```
c=(char)%fgetc( (char*)strm )
crt c
Note the character type casting to override the default (int) type.
```

%FGETS

reads characters from the named input "stream" into "variable" until 'n'-1 characters are read or a new line character is read and transferred to the string.

The string is then terminated with a null character (hex '00'). A string of a size at least equal to 'n' must have been assigned to 'variable' before the call, otherwise the data is truncated. If less than n-1 bytes are returned, the content of the string beyond is undefined, as is usual in C. No data translation occurs.

The "pointer" returned has no meaning, except for a null value which indicates an error or that end of file has been encountered.

Syntax

ptr = (char*)%FGETS(variable, n, (char*)stream)

Example

```
char line[128] ;* make a buffer
pointer=(char*)%fgets( line, 128, (char*)stream )
if pointer#0 then
  line=field(line,char(0),1)
  print line
end
```

%FOPEN

opens the Unix file designated by "string" and associates a stream with it.

"type" is a character string having one of the following values:

"r" Opens for reading only.

"w" Truncates or creates for writing only.

"a" Append; opens for writing at the end of file or creates for writing.

"r+" Opens for update.

"w+" Truncates or creates for update.

"a+" Append; opens for writing at the end of file or creates for update.

The stream is returned as a pointer to a character or 0 if an error occurred. System(0) contains the error number.

Streams are not automatically closed at program termination.

Syntax

stream = (char*)%FOPEN(string, type)

Example

```
stream=(char*)%fopen('/usr/pick/fname',"w+")
if stream=0 then
  crt 'Cannot open stream. errno=':system(0)
end
```

%FPRINTF

writes formatted output on "stream" under control of the "format". See the Unix Programmer's Reference Manual.

Floating types are not supported.

Syntax

variable = %FPRINTF((char*)stream, format, arg....)

%FPUTC

writes the character specified by "character" on the named output "stream".

Note: The character is passed as a number. Normal usage would override the type (see example below). The Unix "putc()" and "putchar()" cannot be used because they are macros, rather than functions.

If successful, the character is returned as a number, else EOF (-1) is returned.

If the character is contained in a FlashBASIC dynamic array, an explicit (char) type override must be used, otherwise a pointer would be passed to the C function.

Syntax

variable = %FPUTC(character, (char*)stream)

Example

```
c='a'
n=%fputc( (char)c, (char*)strm )
```

%FPUTS

writes the string designated by "string" on the named output stream.

EOF (-1) is returned in case of an error.

Syntax

variable = %FPUTS(string, (char*)stream)

Example

```
equ NL to char(10)
n=%fputs('Input file name':NL,(char*)stream)
```

%FREE

frees a block of memory allocated by a call to %malloc().

"pointer" is the pointer to the area of memory allocated by a previous call to %malloc().

Syntax

(void)%FREE((char*)pointer)

Example

```
%free( (char*)ptr )
```

%FREOPEN

substitutes the named file specified by "string" in place of the opened stream "stream1".

The stream is returned as a pointer to a character or 0 (zero) if an error has occurred. System(0) is then set to the error number.

Syntax

stream = (char*)%FREOPEN(string, type, (char*)stream1)

%FSIZE

returns the size in bytes of the Unix file associated to the opened "file.descriptor" returned by a previous call to "%open()" or "%creat()".

If the file does not exists or is not readable, "-1" is returned.

Syntax

size = %FSIZE(file.descriptor)

%GETENV

searches the environment for a string of the form "name=value" and returns a pointer to "value" in the current environment if such a string is present, otherwise, a NULL is returned.

Syntax

pointer = (char*)%GETENV(name)

Example

```
* Get the Unix name of our terminal: eg TERM=wy50
pointer=(char*)%getenv( 'TERM' )
if pointer=0 then
  crt 'Unix TERM is not defined'
end else
  * We got a C pointer to the 'name'. Copy it into Pick
  * BASIC variable
  char pickterm[32]
  %strcpy( pickterm, (char*)p )
  pickterm=field(pickterm,char(0),1)
end
```

%GETHOSTID

allows a FlashBASIC application to retrieve the unique 32-bit identifier for the current host. The id is returned as a decimal number.

If the function fails, a value of -1 is returned and the (FlashBASIC) system(0) function returns the value of "errno".

To compile successfully, the program must include the statement:

cfunction socket.builtin

Syntax

variable = %GETHOSTID()

Example

```
cfunction  socket.builtin
id=%gethostid()
if id=-1 then
  crt 'Cannot get host id. errno=':system(0)
  stop
end
```

%GETPGRP

returns the Unix process group ID of the calling process.

Syntax

variable = %GETPGRP()

%GETPID

returns the Unix Process Id (PID) of the calling process.

Syntax

variable = %GETPID()

%GETPPID

returns the Unix parent process ID of the calling process.

Syntax

variable = %GETPPID()

%IOCTL

A general purpose control function which passes arguments "request" and "arg" to the device designated by "file.descriptor".

The format of the arguments is device-specific. Note that most of the time, these arguments will be binary data that must be kept out of FlashBASIC data space. See the example below.

Syntax

variable = %IOCTL(file.descriptor, request, arg)

Example

```
* Allocate some data
ptr=(char*)%malloc( 128 )
* Open the terminal and get its termio structure
execute "!exec tty" capturing ttyname
fd=%open( ttyname, O$RDWR )
n=%ioctl( fd, TCGETA, (char*)ptr )
...
* Release data
%free( (char*) ptr )
```

%KILL

sends the signal specified in "signal" to the process "pid".

All D3 processes normally catch signals for their internal use. The built-in "%pgetpid" allows finding the PID of a process by knowing its port.number (pib). Only "SIGUSR2" should be sent to a D3 process. Other signals are used internally and may cause problems if used out of context.

"SIGTERM" will logoff the D3 process and disconnect it.

"SIGHUP" will logoff the D3 process, but leave it connected to the D3 virtual machine. This behavior can be modified by providing a user writtem signal handler. See the 'trap' command.

"SIGINT" will emulate a <BREAK>, possibly sending the D3 process to the debugger.

Signal numbers are defined in "dm,bp,unix.h signal.h".

Syntax

variable = %KILL(pid, signal)

Example

```
Get its pid, and send hangup
* (SIGHUP=1) to it.
pib=32
pid=%pgetpid( pib )
if %kill( pid, 1 ) = -1 then
  print "Cannot logoff process ":pib
end
```

%LISTEN

marks the specified socket as accepting incoming connections and limits the number of outstanding connections in the system queue.

To compile successfully, the statement 'cfunction socket.builtin' must be included in the source code.

'socket' Is the file descriptor of the local socket returned by a previous call to the FlashBASIC C function '%socket'.

'backlog' Maximum number of outstanding connections. The maximum value for this number is SOMAXCONN defined in the include: 'dm,bp,unix.h socket.h'

Upon successful completion, a value of 0 is returned in 'code'. In case of error, a value of -1 is returned and the function 'SYSTEM(0)' is set to the value of 'errno'.

Syntax

code = %LISTEN(socket, backlog)

Example

```
cfunction socket.builtin
include dm,bp,unix.h socket.h
* Create a socket
fd=%socket( AF$INET, SOCKET$STREAM, 0 )
* Bind the socket to a local Ethernet port.
* Use default address.
if %bind( fd, AF$INET, INADDR$ANY, 1024 )<0 then
  crt 'bind failed'; stop
end
* Wait for incoming connection and allow up to
* three more to be waiting.
%listen( fd, 3 )
* Accept a connection until got them all
```

```
loop
  addr=0; port=0
  fd=%accept( socket, &addr, &port )
until fd<0 do
  crt 'Called by address ':addr:', port #':port
* Read data from the data link
  %read( fd, buffer, 1024 )
* Done with this connection, close it
  %close( fd )
repeat
```

%LSEEK (D3 Unix)

moves the read/write pointer. See the Unix programmer's reference manual.

Syntax

variable = %LSEEK(file.descriptor, offset, whence)

%MALLOC

allocates a memory block of "size" bytes and returns the address of this block.

The memory is freed by the "%free()" built-in. If the system fails to allocate memory, a NULL pointer is returned.

If the FlashBASIC program terminates, the malloc'ed space is not released automatically. This feature can be used to create 'static' data, live across program executions. The TCL command 'environ' uses this fact.

"malloc'ed" memory may be "free'd" by "exit"ing the D3 process.

Syntax

variable = (char*)%MALLOC(size)

Example

```
ptr=(char*)%malloc( 2048 )
if ptr=0 then stop 'cannot allocate memory'
```

%MEMCOPY

copies characters from memory area specified by "s2" into "s1", stopping after the first occurrence of character designated by "character" has been copied, or after the number of characters specified by "length" have been copied, whichever comes first.

"s1" and "s2" are either FlashBASIC strings or pointers to a character.

"variable" is a pointer to the character after the copy of "character" in "s1" or a NULL pointer if "c" was is found in the first "length" characters of "s2". If "s1" is a D3 string, the value returned by this function, except NULL, has no meaning.

Syntax

variable = (char*)%MEMCOPY(s1, s2, (char)character, length)

%MEMCPY

copies the number of characters specified by "number" from memory area "s2" into "s1".

"s1" and "s2" are either FlashBASIC strings or pointers to a character. "variable" is a pointer to "s1". If "s1" is a D3 string, the value returned by this function has no meaning.

Syntax

variable = (char*)%MEMCPY(s1, s2, number)

Example

```
* Read 1024 bytes from device
* and copy them in a Pick buffer
precision 0
char buffer[1024]
p=(char*)%malloc( 1024 )
n=%read(fd, (char*)p, 1024 )
%memcpy( buffer, p+32, n-32 )
In this example, note the operation on pointers, allowed because of
the statement  'precision 0'. This allows reading data from a device,
and copy only the portion after a fixed size header (32 bytes in this
example).
```

%MEMXCPY

copies "number/2" characters from memory area "s2" into "s1", and converts each input character into two ASCII hexadecimal characters '0' to '9' and 'A' to 'F'.

"s1" and "s2" are either FlashBASIC strings or pointers to a character. "variable" is a pointer after the last converted character in "s2" . If "s2" is a D^3 string, the value returned by this function has no meaning.

Syntax

variable = (char*)%MEMXCPY(s1, s2, number)

Example

```
* Obtain Unix space
ptr=(char*)%malloc( 2048 )
* Read from Unix file
n=%read( fd, (char*)ptr, 1024 )
* Convert the binary data into HEX ASCII
char buffer[2048]
%memxcpy( buffer, (char*)ptr, n*2 )
```

%OPEN

opens the Unix file specified by "string" and sets the file status flags according to the value of "oflag".

Files opened by %open() are closed automatically when the FlashBASIC programs terminates.

Valid values of "oflag" are defined in the include fcntl.h. Combinations of the modes are obtained by adding several flags together from the following list:

O$RDONLY	Opens for reading only.
O$WRONLY	Opens for writing only.
O$RDWR	Opens for reading and writing.
O$NDELAY	Non blocking I/O. The effect of this flag varies depending on the type of the file. See the Unix Programmer's Reference Manual.
O$APPEND	Moves the file pointer to the end of the file.
O$SYNC	Sync writes.
O$CREAT	If the file exists, this flag has no effect. Otherwise, owner ID and group ID are set and the mode of the file is set according to the value of mode modified as follows: all bits in the file mode creation mask of the process are cleared and the sticky bit is cleared.
O$TRUNC	If the file exists, its length is set 0.

O$EXCL If O$EXCL and O$CREAT are set, open will fail if the file exists.

The file descriptor is returned as a number or "-1" if an error occurred. System(0) contains the error number.

Syntax

file.descriptor = %OPEN(string, oflag {, mode })

Example

```
include dm,bp,unix.h fcntl.h
fd=%open( '/usr/pick/fname', O$WRONLY+O$APPEND )
if fd<0 then
  crt 'Cannot open. errno=':system(0)
end
```

%PAUSE

suspends the calling process until it receives a signal.

<BREAK>, <ESCAPE>, a logoff, a signal or a message sent by another process causes the process to resume execution. In case of a logoff, control is not returned to the FlashBASIC program. This is equivalent to an infinite sleep. Only "SIGUSR2" should be used to wake up a process waiting in "%pause".

Syntax

(void)%pause()

%PCLOSE

closes the stream designated by "stream" opened by a "%popen()" and returns the exit status of the command.

Syntax

variable = %PCLOSE((char*)stream)

%PGETPID

returns the Unix Process Id (PID) associated to the D3 process "line".

If "line" is equal to -1, the PID of the current process is returned. If "line" is equal to -2, the PID of the flush process is returned. If the D3 process is not connected, 0 (zero) is returned.

Syntax

variable = %PGETPID(line)

Example

```
* Get our own pid
pid=%pgetpid( system( 22 ) )
```

%POPEN

creates a pipe between the calling process and the command to be executed.

"command" Shell command line.

"type" I/O mode, either "r" for reading or "w" for writing.

"pointer" Stream pointer which directs a write to the standard input of the command if the I/O mode is "w", and reads from the standard output of the command if the type is "r".

A stream opened by "%popen()" must be closed by "pclose()". The stream is closed automatically at FlashBASIC program termination.

Syntax

pointer = (char*)%POPEN(command, type)

Example

```
char buffer[128]
ptr=(char*)%popen("ls -l","r")
if ptr=0 then crt 'error'; stop
loop
```

```
  n=(char*)%fgets(buffer,128,(char*)ptr)
while n#0 do
  print field(buffer,char(0),1)
repeat
%pclose((char*)ptr)
This example executes a C command, capturing the result.
```

%PUTENV

makes the value of the environment variable name equal to the value designated by "string", by altering an existing variable or by creating a new one.

"string" String of the form name=value.

"result" Non-zero if the variable is added successfully to the environment, or zero if an error occurred.

The environment contains a pointer to 'string', making it a mistake to use a FlashBASIC variable as 'string' (FlashBASIC variables are essentially volatile). Proper use requires using "%malloc()" to obtain non-FlashBASIC space, use "%strcpy()" to copy the FlashBASIC variable into it, and pass the malloced space to "%putenv", as in the example below.

Syntax

result = %PUTENV(string)

Example

```
Add the string 'port=current port.number' to the Unix environment of
the process. After this small program has been executed, the TCL
command 'env' now shows the port.number of the process. The TCL
command "environ" does this kind of environment setting.
* Build the string
string='PORT_NUMBER=':system(22)
* get workspace (+1 for terminating NULL)
ptr=(char*)%malloc( 1+len(string) )
* Put the string into the allocated buff
%strcpy( (char*)ptr, string )
* Add it to the environment
%putenv( (char*)ptr )
```

%RDHEX

reads the number of bytes designated by "size" into "variable" from the file specified by "file.descriptor" returned by a previous call to "%open()", "%creat()" or "dup()" converting each byte into two ASCII hexadecimal characters.

A string of a size at least equal to 2 * "size" must have been assigned to "variable" before the call by either an explicit assignment (e.g. buffer=space(1000)) or by the "char" reserve statement, otherwise data will be truncated.

The function returns the number of bytes actually read or "-1" if an error occurred. If less than 2 * "size" bytes are returned, the content of the string beyond is undefined, as is usual in C. This function is the opposite of "%whex()".

Syntax

n = %RDHEX(file.descriptor, variable, size)

%READ (D3 Unix)

reads the number of bytes designated in "size" into the "variable", from the file specified by "file.descriptor" returned by a previous call to "%open()", "%creat()" or "dup()".

A string of a size at least equal to "size" must have been assigned to "variable" before the call by either an explicit assignment (eg buffer=space(1000)) or by the "char" reserve statement, or else the data is truncated.

This function returns the number of bytes actually read. If less than "size" bytes are returned, the content of the string beyond is undefined, as is usual in C. No data translation occurs. If the data read in variable contains Segment Marks (x'ff'), results are unpredictable. Use "%malloc()" to obtain a Unix memory in which to read binary data.

Syntax

n = %READ(file.descriptor, variable, size)

Example

```
item=''
char buffer[10000] ;* reserve buffer space
loop while true do
  n=%read(fd, buffer, 10000)
  if n=-1 then
    print "Error ":system(0)
    stop
  end else
    if n<10000 then
      * The whole file has been read.
      item=item:buffer[1,n]
      convert char(10) to char(254) in item
      write item on itemname
      stop
    end else
    * Some more to read
      item=item:buffer
    end
  end
repeat
```

%SEMCTL

performs semaphore control operations.

semid Semaphore id returned by a call to semget().

semnum Semaphore number in the set, when applicable (depending on "cmd").

cmd Command code (see below).

arg Dimensioned array which contains the argument (see below).

The following "cmd" are executed with respect to the semaphore specified by "semid" and "semnum":

GETVAL Returns the value of "semval".

SETVAL Sets "semval" to the value of arg.

GETPID Returns the value of "sempid".

GETNCNT Returns the value of "semncnt".

GETZCNT Returns the value of "semzcnt".

The following "cmd" apply to all semaphores in the set:

GETALL Place all the semaphore values in the dimensioned array "arg".

SETALL Set all the semaphores to the values contained in the dimensioned array "arg".

The following commands are also available:

IPC$STAT Return a dimensioned array which contains the various s fields of the Unix sempahore structures. The elements are (see the Unix Programmer's Reference guide):

1 sem_perm.uid

2 sem_perm.cid

3 sem_perm.guid

4 sem_perm.gcid

5 sem_perm.mode

6 sem_perm.seq

7 sem_perm.key

8 sem_nsems

9 sem_otime

10 sem_ctime

IPC$SET Set the values of the following members of the data structure associated to "semid" to the first three elements of the array "arg":

1 sem_perm.uid

2 sem_perm.gid

3 sem_perm.mode

IPC$RMID Removes the semaphore identifier.

Valid values for "cmd" are defined in "sem.h" and "ipc.h".

Syntax

code = %SEMCTL(semid, semnum, cmd, arg)

Example

```
include dm,bp,unix.h ipc.h
include dm,bp,unix.h sem.h
* Set initial semaphore values
dim val(3)
mat val=0
%semctl( semid, 0, SETALL, val )
```

%SEMGET

creates a semaphore set.

Valid values for "key" and "semflg" are defined in "sem.h" and "ipc.h". See the Unix Programmer's guide.

Syntax

value = %SEMGET(key, nsems, semflg)

%SEMOP

performs an array of semaphore operations on a set of semaphores.

semid Semaphore identifier returned by a call to semget().

sops Array of semaphores operation. "sops" is a "nsops" by 3 two-dimension array. Each row in the array is the equivalent of a "sops" structure, as defined in the Unix Programmer Reference

Guide. In other words, sops(i,1) is sem_num, sops(i,2) is sem_op, sops(i,3) is sem_flg. See the Unix documentation for the description of these fields.

nsops Number of rows in the "sops" array.

Syntax

code = %SEMOP(semid, sops, nsops)

Example

```
include dm,bp,unix.h ipc.h
include dm,bp,unix.h sem.h
include dm,bp,unix.h mode.h
* Get a set of 4 semaphores
semid=semget( key, 4, IPC$CREAT )
if semid<0 then
    crt "semget error=":system(0)
    stop
end
* Wait for: (1st sem >=1) and (2nd sem >=2)
dim sops(2,3)
* - Set 1st condition
sops(1,1)=0           ;* 1st sem
sops(1,2)=-1          ;* Wait until can do -1
sops(1,3)=0           ;* No flag
* - Set 2nd condition
sops(1,1)=1           ;* 2nd sem
sops(1,2)=-2          ;* Wait until can do -2
sops(1,3)=0           ;* No flag
n=semop( semid, sops, 2)
if n<0 then
    crt "semop error=":system(0)
    stop
end
```

%SETFLUSH

sets the flush periods.

The normal flush period is set to the number of seconds specified by "newp", the forced flush period to "newfp" and the previous values are returned into "oldp". The upper 16 bits of the (int) "oldp" is the value of the forced flush period, and the lower 16 bits, the value of the normal flush.

If "newp" is null, no change is made to the normal flush.

If "newfp" is equal to -2, no change is made to the forced flush.

If "newfp" is equal to -1, the forced flush is disabled.

If "newfp" is equal to 0, the forced flush is set into 'immediate write' mode.

Syntax

oldp = %SETFLUSH(newp, newfp)

%SHMAT

attaches a shared memory segment associated with the shared memory identifier "shmid" and returns the address of the segment.

Valid values for "shmflg" are defined in "shm.h" and "ipc.h". See the Unix Programmer's guide.

Syntax

pointer = (char*)%SHMAT(shmid, (char*)shmaddr, shmflg)

%SHMDT

detaches the shared memory segment located at the address specified by "shmaddr".

Syntax

value = %SHMDT((char*)shmaddr)

%SHMGET

creates a shared memory segment and returns a shared memory identifier.

Valid values for "key" and "shmflg" are defined in "shm.h" and "ipc.h". See the Unix Programmer's guide.

Syntax

shmid = %SHMGET(key, size, shmflg)

%SOCKET

creates a socket in the specified 'addr.family' and of the specified 'type'. A protocol can be specified or assigned by the system. If the protocol is left unspecified (with a value of 0), the system selects an appropriate protocol in the specified address family.

To compile successfully, the statement 'cfunction socket.builtin' must be included in the source code.

'addr.family' Specifies the addressing scheme which will be used later to decode addresses. Valid values are defined in the include "dm,bp,unix.h socket.h". Commonly used values are:

AF$UNIX Unix path names.

AF$INET ARPA Internet.

'type' Specifies the semantics of communication. Valid values are defined in the include "dm,bp,unix.h socket.h". Commonly used values are:

SOCK$STREAM Unix streams.

SOCK$DGRAM Datagram

'protocol' Should be left to 0, to let the system assign the protocol.

Upon successful completion, a valid file descriptor is returned. If the call fails, a value of -1 is returned and the function 'SYSTEM(0)' returns the value of 'errno'.

The socket is closed by a %close call or automatically when the basic program terminates

Syntax

n = %SOCKET(addr.family, type, protocol)

Example

```
cfunction socket.builtin
include dm,bp,unix.h socket.h
fd=%socket( AF$INET, SOCK$STREAM, 0 )
if fd<0 then
  crt 'Socket creation failed. Error ':system(0)
```

%TTYNAME

returns a pointer to a static area containing the null terminated path name of the terminal device associated to the file descriptor "fd". If "fd" does not describe a device in the directory "/dev", a NULL pointer is returned.

When run on a phantom process, this function returns a NULL pointer.

Syntax

ptr = (char*)%TTYNAME(fd)

Example

```
char buffer[32]
* Get the device name associated to our standard input
ptr=(char*)%ttyname( 0 )
if ptr=0 then
* This is not a terminal (pipe?)
  return
end
* Copy the data from the static string to Pick
char buffer[32]
%strcpy( buffer, (char*)ptr )
mytty=field( buffer, char(0), 1)
```

%UNLINK

removes the directory entry named by the path name pointed to by "path". The named file is unlinked, and when all the links have been removed, and the file is no longer open, the file is removed.

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and system(0) is set to the value of errno.

Syntax

code = %UNLINK(path)

Example

```
include dm,bp,unix.h fcntl.h
* Get a unique temporary file name
fn='/tmp':system(19)
fd=%creat( fn, O$RDWR )
...
* Remove the temporary file
if %unlink( fn ) # 0 then
  crt 'Cannot remove temporary file. Error ':system(0)
end
```

%WAIT

returns the exit code of the child process(es) created by the current process.

The "status" is returned in the integer "status", passed by its address.

Note that even though "status" is an integer, only the lower 16 bits are updated by the system call. See the Unix Programmer's Reference Manual for more details on the format of the returned status.

Syntax

variable = %WAIT(&status)

Example

```
status=0
(void)%wait( &status )
high$status=int(status/256)
low$status=mod(status,256)
begin case
  case low$status=255
```

```
  * Child stopped due to a signal
  case low$status=0
  * Child stopped by calling exit()
  case high$status=0
  * Child was terminated due to a signal
end case
```

%WHEX

writes to the file specified by "file.descriptor" returned by a previous call to "%open()", "%creat()" or "dup()" a string of bytes resulting from converting the ASCII string variable into binary.

Two digits are converted into one eight bit binary character. variable is assumed to contain only hexadecimal characters '0' to '9', 'A' to 'F' or 'a' to 'f'. The function returns the number of bytes actually written or -1 if an error occurred. The size written to the file is the size of variable divided by 2. This function is the reverse of "%rdhex()".

Syntax

byte.count = %WHEX(file.descriptor, variable)

Example

```
* Write "EOF"x'ff' on device
n=%whex( fd, oconv("EOF","mx"):'FF' )
```

%WRITE (D3 Unix)

writes the number of bytes desinated by "size" from "variable" to the file specified by "file.descriptor" returned by a previous call to "%open()", "%creat()" or "dup()".

This function returns the number of bytes actually written. If the length of variable is less than size, the content of the string written is undefined, as is usual in C. No data translation occurs.

Syntax

n = %WRITE(file.descriptor, variable, size)

Example

```
n=len(item)
if %write(fd,item,n)<n then crt "write error"
```

&

Logical "and" statement.

'

String delimiter.

(

used to surround arguments within functions, or to enclose subscript refererences within dimensioned arrays. It is also used to override the normal order of precedence evaluation.

Example

```
"(" as a function delimiter:
if not(num(response)) then crt "must be numeric!"
As a dimensioned array reference:
if cust.array(22) = "" then cust.array(22) = "hold"
Changing precedence in an arithmetic expression:
age = (today - birthday) / "365"
Without grouping precedence, the above calculation produces an
```

```
inaccurate result due to the fact that division occurs at a higher
level of precedence than subtraction.
```

)

surrounds arguments within functions, encloses subscript refererences within dimensioned arrays, and overrides the normal order of precedence evaluation.

Example

```
")" as a function delimiter:
if not(num(response)) then crt "must be numeric!"
As a dimensioned array reference:
if cust.array(22) = "" then cust.array(22) = "hold"
Changing precedence in an arithmetic expression:
age = (today - birthday) / "365"
Without grouping precedence, the above calculation produces an
inaccurate result due to the fact that division occurs at a higher
level of precedence than subtraction.
```

*****

used as a "remark" statement, text after this statement are ignored.

Syntax

* comment(s)

Example

```
*
* Init Customer Counter
*
  customer.count = 0
*
* Open Customer File
*
  open ...
This illustrates "*" as a remark or comment statement.
```

*****

A mathematical operator indicating a multiply operation.

Syntax

num.expression * num.expression

Example

```
gross.wages = hours.worked * hourly.rate
This example illustrates the use of "*" to indicate multiplication.
```

*****

A "wildcard" array reference within dimensioned and dynamic arrays.

Syntax

item<*>

item<*,*>

item<*,*,*>

Example

```
if orders<price,*> = 10 then print "Price is 10"
```

***=**

an assignment operator which multiplies a given numeric expression and assigns it to the given variable.

Syntax

variable *= num.expression

Example

```
x *= 2
Multiplies the current value of "x" by "2" and assigns the result to
"x". This is equivalent to: x = x * 2
```

+

performs addition or indicates a positive numeric value.

Syntax

num.expression + num.expression

Example

```
x = x + 1
Adds one to the variable, "X".
subtotal = price + tax
Adds the values in the variables "price" and "tax" and assigns the
result to "subtotal".
subtotal = tax + (unit.cost * qty)
This is a complex arithmetic statement which first multiplies the
"unit.cost" by "qty" and then adds the results to the variable called
"tax".  The parentheses "()" are used to explicitly group precedence.
```

+=

an assignment operator which adds a given numeric expression and assigns it to the given variable.

Syntax

variable += num.expression

Example

```
x += 1
Adds "1" to the variable, "x".
total += tax
Is equivalent of the statement:
total = total + tax
```

, (comma)

has several purposes:

1) To separate subscript references within multi-dimensional arrays.

2) To separate "ac.exp", "vc.exp", and "sc.exp" in dynamic array references.

3) To delimit arguments within functions.

4) To separate "passed" arguments in a "call" statement.

5) When used with the "crt" or "print" statement, to "tab" to the next tab stop. Tabstops are established every 18 columnar positions.

Syntax

dim.array.variable(col.exp , row.exp)

dynamic.array.variable<ac.exp , vc.exp, sc.exp>

function(argument1 , argument2 , ...)

call subroutine.name(argument1 , argument2 , ...)

crt/print string.exp , string.exp , ...

Example

```
if screen.array(13,12) ...
The comma delimits the "row" and "column" arguments in a
two-dimensional array reference.
customer.item<17,-1> = invoice.pointer
The comma delimits the "ac.exp" (17) and the "vc.exp" (-1) in a
dynamic array reference.
cnt = dcount(string,char(253))
The comma delimits the arguments in the "dcount" function.
call display.lines(cust.item,balance,inv.list)
The comma separates the arguments in the list being passed to the
subroutine.
crt "hello" , "there"
This prints "hello", followed by 13 spaces and "there". This is like
a "tabstop", and moves to the next multiple of 18 on the print or
display line.
```

-

designates a subtraction operation, or to indicate a negative numeric value.

Syntax

num.expression - num.expression

Example

```
x = x - 1
Subtracts "1" from the contents of variable "x".
x = x + (-z)
Negates the contents of "z" and adds to the contents of "x".  The
result is placed in "x".
amount = price - discount.amount
Subtracts the contents of the variable "discount.amount" from the
variable "price".  Places the result in the variable "amount".
amount = (unit.cost * qty) - discount.amount
Subtracts the contents of variable "discount.amount" from the product
of "unit.cost" and "qty".  The parenthesis "()" used to group
precedence are unnecessary due to default precedence. They are added
for program clarity.
counter = -1
This assigns the value, "-1", to the variable "counter".
```

-=

decrements a variable by a numeric expression.

Syntax

variable -= num.expression

Example

```
x -= 1
Subtracts one from the variable, "x" and assigns the result to "x".
total -= tax
This is equivalent to the statement:
total = total - tax
```

/

arithmetic division operator.

Syntax

num.expression / num.expression

Example

```
hourly.cost = total.price / total.hours
Divides the contents of the total.price variable by the contents of
```

```
the total.hours variable and places the results in the hourly.cost
variable.
```

/=

divides a variable by a numeric expression and assigns it to the given variable.

Syntax

variable /= num.expression

Example

```
x /= 2
This divides the current value of "x" by "2" and assigns the result
to "x". This is equivalent to: x = x / 2
```

:

concatenates strings in an assignment statement or, when the ":" appears at the end of a "print" or "crt" expression, suppresses the automatic carriage return/linefeed combination, thus holding the current print/display position. It is also used as a token within the "input @" statement.

Syntax

expression : expression

crt/print expression {:}

input @(x,y):variable ...

Example

```
z = "Amount due = ":amount
print z
Variable "z" is assigned the result of the string expression, "Amount
due = " concatenated to the value of the variable "amount"
print "Enter command ":
The string "Enter command " is printed and the cursor is left at the
end of the string.
```

:=

concatenates a string to a variable and assigns it back to the given variable.

Syntax

variable := string.expression

Example

```
x := y
Concatenates "y" to "x". This is equivalent to: x = x : y
```

;

separates multiple FlashBASIC statements on a single line.

D3 allows a FlashBASIC statement to end with a semi-colon because it allows "null" or blank lines in a program. Non D3 implementations do not allow this feature.

Syntax

statement {; statement...}

Example

```
x = 10 ; y = 15 ; z = x * y ; crt z
This shows multiple statements on a single line.
for i=1 to 9; crt field(s,' ',i); next i
This shows a "for-next" loop on the same line.
```

<

either references subscripts within dynamic arrays, or is a logical operator within conditional expressions.

A "-1" used in a dynamic.array.variable expression will insert data into a new field at the end of the of the specified element. An "*" can be used in a dynamic.array.variable expression to conditionally test all fields expressed against a value.

The "lt" symbol is an alternate form of representing a "less than" condition.

Syntax

dynamic.array.variable<ac.expression{, vc.expression{, sc.expression}}>

expression < expression

Example

```
customer.item<1> = name
Assigns the value of variable "name" to attribute 1 in dynamic array
"customer.item".
item<2,-1> = 10
Inserts the literal 10 into a new value appended to the end of
attribute 2 in dynamic array "item".
if x<*> < 3 then ...
Tests if any attribute in dyanamic array "x" is less than 3.
if x < 0 then...
"<" is used as a logical operator representing "less than".
```

<=

"Less than or equal to" operator.

<>

"not equal to".

=

represents either the assignment operator in an assignment statement or a relational operator in a conditional expression.

In logical (conditional) expressions, the "eq" symbol is an alternative to the "=" sign.

See "relational.operators" for the exact description of how operands are compared in logical expressions.

Syntax

variable = expression

expression = expression

Example

```
customer.item<1> = name
This use of the "=" sign illustrates assignment. The array element on
the left side of the "=" sign is assigned the current value of the
variable, "name".
if answer  = "quit" then stop
This example illustrates a "conditional" expression. If the value of
the variable "answer" is "quit", the expression evaluates to "true"
and the statement following the "then" condition is executed.
```

>

terminates a dynamic array subscript reference, or acts as the logical operator "greater than".

Syntax

dynamic.array.variable<ac.expression{, vc.expression{, sc.expression}}>

expression > expression

Example

```
customer.item<1> = name
```

>

Greater than operator

><

Logical "not equal" operator.

>=

"Greater than or equal to" operator.

@

see "@ function".

@ function

provides a standard means of performing control functions on a wide array of terminals.

Special cursor-control characters for the current terminal type (as defined by the "term" settings in effect at the time) can be generated by using negative values with the "@" function. The values generated by the "@" function are specified in the TCL verb, "define-terminal".

@(column) Generates terminal output codes to position the cursor to a specified column on the current row. The value of the expression "column" must be within the column width limits for the terminal device. The leftmost column is column 0. The rightmost column is determined by the TCL "term" command terminal width setting.

@(column,row) This "@" function generates terminal output codes to position the cursor to a specified location. The values of the expressions used in the "@" function must be within the row and column limits of the terminal screen. The topmost row is row 0. The bottom row is determined by the TCL "term" command terminal depth setting. Location (0,0), the top-left corner, is known as the "home" position. Using an @(column,row) function on output to a printer can have unpredictable results. Use of the @(row,col) function will not change the results from the SYSTEM(4), SYSTEM(5) and SYSTEM(6) functions.

Note: many of the following functions will not behave properly unless the "term type" setting of the "term" verb corresponds to the "emulation type" currently defined in the terminal.

@(-1) Clears screen and positions the cursor to column 0, row 0 (the "home" position).

@(-2) Positions to "home" cursor position: column 0, row 0.

@(-3) Clears the screen from the current cursor position to the end-of-screen.

@(-4) Clears the screen from the current cursor position to the end of the current line.

@(-5) Begins "blink" mode. All characters written after this command "blink" on and off. This mode remains in effect until it is terminated by the @(-6) function.

@(-6) Terminates "blink" mode.

@(-7) Begins "protected-field" mode. Characters are now "protected" from being overwritten by data entry. Usually, the characters are displayed at a lower intensity to indicate "protected-field" mode. This mode remains in effect until it is terminated by the @(-8) function.

@(-8) Terminates "protected-field" mode.

@(-9) Cursor back. Moves the character one position to the left (non-destructively). What happens at the margin depends on the brand of terminal. It may either stay at the margin, or "wrap" to the last position on the previous line.

@(-10) Cursor Up. Moves the cursor up one position upwards (non-destructively). What happens at the top row depends on the brand of terminal. It may either stay on the top row, or "wrap" back to the bottom row.

@(-11) Enables Protect Mode. Enables the terminals ability to accept "protected-field" mode commands @(-7) and @(-8).

@(-12) Disables Protect Mode. Disables the terminal's ability to accept "protected-field" mode commands @(-7) and @(-8).

@(-13) Starts Reverse Video. Begins "reverse-video" mode. The Foreground color is changed to the Background color, and vice-versa. This mode remains in effect until it is terminated by the @(-14) function. The Output processor uses this mode as "boldface-on" mode.

@(-14) Stops Reverse Video. Terminates "reverse-video" mode.

@(-15) Begins "underline" mode. All characters written after this command will be underlined. This mode is terminated by the @(-16) function.

@(-16) Terminates "underline" mode.

@(-17) Begins "slave-printer" ("transparent") mode. This mode enables the terminals auxiliary serial (slave printer) port. This mode remains in effect until it is terminated by the @(-18) function.

@(-18) Terminates "slave-printer" mode.

@(-19) Cursor Forward. Moves the cursor one position to the right (non-destructively). What happens at the margin depends on the brand of terminal. The cursor will either stay at the margin, or it will "wrap" to the first column on the following line.

@(-20) Cursor Down. Moves the cursor one position downwards (non-destructively). What happens at the margin depends on the brand of terminal. The cursor will either stay at the at the bottom row, or it will "wrap" back up to the top row.

@(-21) Graphics Character Set On. Enables the terminals ability to accept and display graphics characters (char-128 through char-255). This mode remains in effect until it is terminated by a @(-22) function. NOTE: Also see the TCL command "xcs-on".

@(-22) Graphics Character Set Off. Disables the terminals ability to accept and display graphics characters.

@(-23) Keyboard Lock. Disables the terminals ability to accept ANY input from the keyboard. This mode remains in effect until it is terminated by the @(-24) function.

@(-24) Keyboard Unlock. Enables the terminals ability to accept input from the keyboard.

@(-25) Control Character Enable. Enables the terminals ability to accept and transmit control characters (char-00 through char-31). D3 does not re-display control characters.

@(-26) Control Character Disable. Disables the terminals ability to accept and transmit control characters.

@(-27) Write Status Line. Positions the cursor to the "status line", and writes the text that follows on that line. The "status line" is the bottom line of the terminal display which is not considered part of the active display area; that is, it does not erase when a "clear-screen" command is issued, nor is the line used for data input. This line is usually used as a Legend for the top row of function keys, or as a status display of various information.

@(-28) Erase Status Line. Positions the cursor to the "status line", and erases any data on the status line.

@(-29) Initialize Terminal Mode. Resets (initializes) the terminal back to its original power-on settings.

@(-30) Download Function Keys. Accepts the character string that follows and applies it to a function-key translate table, so when a function key is pressed, a (previously-defined) string of keystrokes will be transmitted.

@(-31) Non-Embedded Stand-Out On. Enables the terminals ability to display (normally hidden) terminal display attribute codes.

@(-32) Non-Embedded Stand-Out Off. Disables the terminals ability to display (normally hidden terminal display attribute codes.

Functions @(-33) through @(-40) set color terminals background colors. See also the TCL "color" command.

@(-33) Set Background Color = WHITE

@(-34) Set Background Color = BROWN

@(-35) Set Background Color = MAGENTA

@(-36) Set Background Color = RED

@(-37) Set Background Color = CYAN

@(-38) Set Background Color = GREEN

@(-39) Set Background Color = BLUE

@(-40) Set Background Color = BLACK

Functions @(-41) through @(-48) set color terminals foreground colors to a full-intensity color. See also the TCL "color" command.

@(-41) Set Foreground Color = WHITE

@(-42) Set Foreground Color = BROWN

@(-43) Set Foreground Color = MAGENTA

@(-44) Set Foreground Color = RED

@(-45) Set Foreground Color = CYAN

@(-46) Set Foreground Color = GREEN

@(-47) Set Foreground Color = BLUE

@(-48) Set Foreground Color = BLACK

@(-49) (Reserved)

@(-50) (Reserved)
@(-51) (Reserved)
@(-52) (Reserved)
@(-53) (Reserved)
@(-54) (Reserved)
@(-55) (Reserved)
@(-56) (Reserved)

Functions @(-57) through @(-64) set color terminals foreground colors to a low-intensity color. See also the TCL "color" command.

@(-57) Set Foreground Color = WHITE
@(-58) Set Foreground Color = BROWN
@(-59) Set Foreground Color = MAGENTA
@(-60) Set Foreground Color = RED
@(-61) Set Foreground Color = CYAN
@(-62) Set Foreground Color = GREEN
@(-63) Set Foreground Color = BLUE
@(-64) Set Foreground Color = BLACK
@(-65) (Reserved)
@(-66) (Reserved)
@(-67) (Reserved)
@(-68) (Reserved)
@(-69) (Reserved)
@(-70) (Reserved)
@(-71) (Reserved)
@(-72) (Reserved)
@(-73) (Reserved)
@(-74) (Reserved)
@(-75) (Reserved)
@(-76) (Reserved)
@(-77) (Reserved)
@(-78) (Reserved)
@(-79) (Reserved)
@(-80) (Reserved)
@(-81) (Reserved)
@(-82) (Reserved)
@(-83) (Reserved)
@(-84) (Reserved)

@(-85) (Reserved)

@(-86) (Reserved)

@(-87) Set to full-intensity white background.

@(-88) (Reserved)

@(-89) Activate monochrome monitor to 80x25 mode. Sets the terminal to Black & White in 80 columns by 25 rows mode. See also the TCL "mono" command.

@(-90) (Reserved)

@(-91) (Reserved)

@(-92) (Reserved)

@(-93) Activate color monitor to 80x25 mode. Sets the terminal to Color 80 columns by 25 rows mode. See also the TCL "color" command.

@(-94) Activate color monitor to 80x25 B&W mode. Sets the terminal to Black & White in 80 columns by 25 rows mode. See also the TCL "mono" command.

@(-95) (Reserved)

@(-96) (Reserved)

@(-97) (Reserved)

@(-98) (Reserved)

@(-99) Embedded Visual Attributes. Some terminals, when they change visual display modes such as "blink", "underline", and "protected-field", take up a position on the visual display to activate the mode change. Such a terminal is said to have the visual attribute "embedded" in the display, rather than having "hidden" visual attributes. This is not a terminal function, but rather a way to notify the program that the terminal does or does not use a space in visual attributes, so it can properly keep track of the current cursor position.

"0" = Visual Attributes are "hidden".

"1" = Visual Attributes are "embedded".

@(-100) Half Intensity. Sets the foreground brightness to half-intensity. All characters written after this command will now be at half-intensity. This mode remains in effect until it is changed by the @(-101) function. Function @(-7) usually has the same visual effect, depending on the terminal.

@(-101) Full Intensity. Sets the foreground brightness to full-intensity. All characters written after this command will now be at full-intensity. This mode remains in effect until it is changed by the @(-100) function. Function @(-8) usually has the same visual effect.

@(-102) Italics On. Enables "italics" mode. All characters written after this command will now be in italics. This mode remains in effect until it is terminated by the @(-103) function.

@(-103) Italics Off. Terminates "italics" mode.

@(-104) Shift Half-Line Downwards. Shifts the line printing by a half-line downwards (for printing subscripts). All characters written after this command will be written as subscripts until they are terminated by the @(-105) function.

@(-105) Shift Half-Line Upwards. Shifts the line printing by a half-line upwards (for printing superscripts). All characters written after this command will be written as a superscript until it is terminated by the @(-104) function.

@(-106) Insert line. Used by the Update processor for backwards scrolling.

@(-107) Delete line.

@(-108) (Reserved)

@(-109) (Reserved)

@(-110) Resets the printer back to its original power-on settings.

@(-111) (Reserved)

@(-112) (Reserved)

@(-113) Set Vertical Motion Index. Sets the height of a single line by specifying how many "units" to skip down before printing the next line. The "units" differ between devices, but they are usually in 48ths, 72nds, or 300ths of an inch.

@(-114) Set Lines Per Inch. Sets the height of a single line by specifying how many "lines" will be printed in an inch. This is typically 6 or 8 lines per inch. Setting the Lines Per Inch has no effect on the actual height of the character, which may be different.

@(-115) Character Printing: Enlarged. Sets the character width to "enlarged". Assuming the standard character set is 10 chars per inch, the "enlarged" characters will be 5 chars per inch.

@(-116) Character Printing: Condensed. Sets the character width to "condensed". Assuming the standard character set is 10 chars per inch, the "condensed" character set is 16.7 chars per inch.

@(-117) Character Printing: Proportional. Sets the character width to "proportional". The "proportional" character set's width will vary depending on the width of each individual character.

@(-118) Perf Skip On. Enables the printer's ability to "skip over" the page perforation on continuous forms. This may also be referred to as a Top or Bottom Page Margin. This mode remains in effect until it is terminated by the @(-119) function.

@(-119) Perf Skip Off. Terminates "perf skip" mode.

@(-120) (Reserved)

@(-121) (Reserved)

@(-122) (Reserved)

@(-123) (Reserved)

@(-124) (Reserved)

@(-125) Set Horizontal Motion Index. Sets the width of a single character by specifying the amount of space (in "units") to move the print head to the right for each character. Usually, the "units" are in 48ths, 72nds, 120ths, or 300ths of an inch.

@(-126) Set Default Font. Specifies which font is to be the default font. This function is also issued by AQL at the beginning of each printed report.

The @(-127) through @(-241) functions involve selecting / downloading fonts to a laser printer.

@(-127) Specifies or downloads Font #1

@(-128) Specifies or downloads Font #2

@(-129) Specifies or downloads Font #3
@(-130) Specifies or downloads Font #4
@(-131) Specifies or downloads Font #5
@(-132) Specifies or downloads Font #6
@(-133) Specifies or downloads Font #7
@(-134) Specifies or downloads Font #8
@(-135) Specifies or downloads Font #9
@(-136) Specifies or downloads Font #10
@(-137) Specifies or downloads Font #11
@(-138) Specifies or downloads Font #12
@(-139) Specifies or downloads Font #13
@(-140) Specifies or downloads Font #14
@(-141) Specifies or downloads Font #15
@(-150) Specifies or downloads Font #16
@(-151) Specifies or downloads Font #17
@(-152) Specifies or downloads Font #18
@(-153) Specifies or downloads Font #19
@(-154) Specifies or downloads Font #20
@(-155) Specifies or downloads Font #21
@(-156) Specifies or downloads Font #22
@(-157) Specifies or downloads Font #23
@(-158) Specifies or downloads Font #24
@(-159) Specifies or downloads Font #25
@(-160) Specifies or downloads Font #26
@(-161) Specifies or downloads Font #27
@(-162) Specifies or downloads Font #28
@(-163) Specifies or downloads Font #29
@(-164) Specifies or downloads Font #30
@(-165) Specifies or downloads Font #31
@(-166) Specifies or downloads Font #32
@(-167) Specifies or downloads Font #33
@(-168) Specifies or downloads Font #34
@(-169) Specifies or downloads Font #35
@(-170) Specifies or downloads Font #36
@(-171) Specifies or downloads Font #37
@(-172) Specifies or downloads Font #38
@(-173) Specifies or downloads Font #39

@(-174) Specifies or downloads Font #40
@(-175) Specifies or downloads Font #41
@(-176) Specifies or downloads Font #42
@(-177) Specifies or downloads Font #43
@(-178) Specifies or downloads Font #44
@(-179) Specifies or downloads Font #45
@(-180) Specifies or downloads Font #46
@(-181) Specifies or downloads Font #47
@(-182) Specifies or downloads Font #48
@(-183) Specifies or downloads Font #49
@(-184) Specifies or downloads Font #50
@(-185) Specifies or downloads Font #51
@(-186) Specifies or downloads Font #52
@(-187) Specifies or downloads Font #53
@(-188) Specifies or downloads Font #54
@(-189) Specifies or downloads Font #55
@(-190) Specifies or downloads Font #56
@(-191) Specifies or downloads Font #57
@(-192) Specifies or downloads Font #58
@(-193) Specifies or downloads Font #59
@(-194) Specifies or downloads Font #60
@(-195) Specifies or downloads Font #61
@(-196) Specifies or downloads Font #61
@(-197) Specifies or downloads Font #63
@(-198) Specifies or downloads Font #64
@(-199) Specifies or downloads Font #65
@(-200) Specifies or downloads Font #66
@(-201) Specifies or downloads Font #67
@(-202) Specifies or downloads Font #68
@(-203) Specifies or downloads Font #69
@(-204) Specifies or downloads Font #70
@(-205) Specifies or downloads Font #71
@(-206) Specifies or downloads Font #72
@(-207) Specifies or downloads Font #73
@(-208) Specifies or downloads Font #74
@(-209) Specifies or downloads Font #75
@(-210) Specifies or downloads Font #76

@(-211) Specifies or downloads Font #77
@(-212) Specifies or downloads Font #78
@(-213) Specifies or downloads Font #79
@(-214) Specifies or downloads Font #80
@(-215) Specifies or downloads Font #81
@(-216) Specifies or downloads Font #82
@(-217) Specifies or downloads Font #83
@(-218) Specifies or downloads Font #84
@(-219) Specifies or downloads Font #85
@(-220) Specifies or downloads Font #86
@(-221) Specifies or downloads Font #87
@(-222) Specifies or downloads Font #88
@(-223) Specifies or downloads Font #89
@(-224) Specifies or downloads Font #90
@(-225) Specifies or downloads Font #91
@(-226) Specifies or downloads Font #92
@(-227) Specifies or downloads Font #93
@(-228) Specifies or downloads Font #94
@(-229) Specifies or downloads Font #95
@(-230) Specifies or downloads Font #96
@(-231) Specifies or downloads Font #97
@(-232) Specifies or downloads Font #98
@(-233) Specifies or downloads Font #99
@(-234) Specifies or downloads Font #100
@(-235) Specifies or downloads Font #101
@(-236) Specifies or downloads Font #102
@(-237) Specifies or downloads Font #103
@(-238) Specifies or downloads Font #104
@(-239) Specifies or downloads Font #105
@(-240) Specifies or downloads Font #106
@(-241) Specifies or downloads Font #107
@(-242) Font Parameter - Point Size
@(-243) Font Parameter - Point Size
@(-244) Font Parameter - Pitch
@(-245) Font Parameter - Pitch
@(-246) Font Parameter - Style - Reverse-Off & Italics-Off
@(-247) Font Parameter - Style - Reverse-Off & Italics-On

@(-248) Font Parameter - Block

@(-249) Font Parameter - Reverse

@(-250) Font Parameter - Black

@(-251) (Reserved)

@(-252) (Reserved)

@(-253) (Reserved)

@(-254) (Reserved)

@(-255) (Reserved)

@(-256) Font Parameter - Stroke - 1 (L. Light)

@(-257) Font Parameter - Stroke - 2 (M. Light)

@(-258) Font Parameter - Stroke - 3 (H. Light)

@(-259) Font Parameter - Stroke - 4 (L. Medium)

@(-260) Font Parameter - Stroke - 5 (M. Medium)

@(-261) Font Parameter - Stroke - 6 (H. Medium)

@(-262) Font Parameter - Stroke - 7 (L. Bold)

@(-263) Font Parameter - Stroke - 8 (M. Bold)

@(-264) Font Parameter - Stroke - 9 (H. Bold)

@(-265) Font Parameter - Stroke -10 (V. Bold)

@(-266) Font Parameter - Spacing - Proportional

@(-267) Font Parameter - Spacing - Fixed

@(-268) (Reserved)

@(-269) Inter-job Sequence. (For Shared Printers on Unix Systems). Sends a string of trailing characters at the end of a print job to indicate to the Unix Spooler services that this print job has terminated (closed).

@(-270) Font Parameter - Orientation - Portrait

@(-271) Font Parameter - Orientation - Landscape (sideways)

@(-272) through @(-300) (Reserved)

Note: If additional, customized @() functions are added, it is recommended that they start at @(-1001) or so. This is to ensure that they do not conflict with existing or future functions.

Syntax

@(num.expression{,num.expression})

Example

```
The following program tests most of the commonly used "@" functions:
print @(-1)
print @(-2)
print @(1,2):'clear from cursor position to end of the screen'
print str('*',250)
print @(12,2):
print @(-3)
print @(1,2):'clear from cursor position to end of the line'
print str('*',250)
print @(12,2):
```

```
print @(-4)
print @(1,6):
print @(-5):'blink on':@(-6):'blink off'
print @(-13):'reverse video on':@(-14):' reverse video off'
print @(-15):'underline on':@(-16):' underline off'
print @(-100):'half intensity':@(-101):' full intensity'
print 'backspace 10 times':
for i=1 to 10
  print @(-9):
  sleep(1)
next i
print
print 'forward 10 times':
for i=1 to 10
  print @(-19):
next i
print
print 'move cursor down 5 lines':
for i=1 to 5
  print @(-20):i:@(-9):
  sleep(1)
next i
print
print 'move cursor up 5 lines':
for i=1 to 5
  print @(-10):i:@(-9):
next i
print @(2,22):"we're done!"
The following example uses two "@" functions. The first, "@(-1)",
clears the screen and positions the cursor to the "home" position.
The second, "@(20,0)", positions the cursor at column position 20,
row position 0, and leaves the cursor positioned there to output the
literal "main menu". The ":" character at the end of the statement
leaves the cursor positioned immediately to the right of the literal,
suppressing the automatic carriage return/linefeed combination
ordinarily sent out after a print string.
print @(-1): @(20,0): "main menu" :
```

@AM

returns a string containing an attribute mark.

Syntax

@AM

@FM

returns a string containing an attribute mark.

Syntax

@FM

@SM

returns a string containing a subvalue mark.

Syntax

@SM

@VM

returns a string containing a value mark.

Syntax
@VM

ABORT

immediately stops program execution and unconditionally returns to the TCL command prompt.

The "abort" statement specifies an abnormal end and can appear anywhere in the program. The "error.message#" parameter is the item-id of the desired message in the "dm,messages," file to be output.

The parameters are the values required by the message. If the "string" form is used, the message string is displayed on the screen and placed in the "dm,messages," file.

The "abort" statement is similiar to the "stop" statement, except the "abort" statement also terminates execution of any macro or PROC which might have called the program containing the "abort" statement and aborts the FlashBASIC program.

The "abort" statement automatically generates error message "[b1] run-time abort at line 'n'".

Syntax
ABORT {message#}{,"parameter"{,"parameter"..}}

ABORT {error message string}

Example
```
open "invoices" to invoice.file else abort 201,"invoices"
If "invoices" can not be opened, the program aborts with error
message 201 and passes in the string "invoices". This causes the
program to stop with the message '"invoices" is not a file name'.  In
this case, both [b201] and [b1] error messages are displayed.
```

ABS

returns the absolute (positive) portion of the integer number derived from the given numeric expression.

Syntax
ABS(num.expression)

Example
```
x = -1
y = abs(x)
print y
"y" will be a positive "1".
```

ACCESS

provides data about the AQL or UP environment when the program is called from a dictionary.

Access() can only be used in subroutines that are called from a dictionary item when in AQL or UP and allows information about the controlling environment to be read and altered by the subroutine.

The following list represents the valid numeric expressions which may be specified in the "access()" function:

1 file.variable for data portion of file.

2 file.variable for dictionary portion of file.

3 Item from the associated file in dynamic array format. If the item is new, this contains "" (null).

4 Item counter (the number of items processed to the current point).

5 Attribute counter (the number of attributes processed).

6 Value counter (the number of values processed).

7 Subvalue counter (the number of subvalues processed).

8 Detail line counter (the number of lines processed). Valid only when a roll-on clause has been specified.

9 Break-on level counter in AQL reports.

10 Item-id.

11 File name.

12 Item delete bit. Returns 1 if the item is being deleted.

13 Root variable.

14 Returns current cursor column position (in UP).

15 Returns current cursor row position (in UP).

16 New item bit. Returns 1 if item is new.

17 Returns 1 if doing input-conversion.

18 Used in the input-conversion attribute of the file "d-pointer" to specify which value of attribute 15 ("macro" attribute) to use.

19 Returns the character which ended the last input, usually a <return>, <ctrl>+u, etc.

20 Item changed bit (0-item not changed, 1-item changed). This is valid only when called from the Update-Processor using a call correlative.

21 This is used from a FlashBASIC subroutine called by UP to place characters in the UP command string. For example, access(21) = "xe" in a subroutine called from the input-conversion causes the Update processor to exit after the attribute calling the subroutine is re-entered.

22 Returns a non-zero (usually 1) if the spelling checker is enabled, or 0 if it is not. Access(22) may be assigned a "1" in a subroutine called by the input conversion of a file-defining item. Even if the spelling checker is disabled for the user session, this invokes the speller for the editing of this item or list of items. The speller will only on attributes with an attribute type of "w". See "attribute-type".

23 Returns the calling environment. It returns 0 if the function was called from FlashBASIC, 1 if the function was called from a callx or from an index, 2 if the function was called from the Update processor (except for a callx or index), and 3 if the function was called from AQL.

24 Returns the actual attribute number of the attribute definition.

25 Returns the left margin from the Update processor.

The access(3) and access(10) functions can be used to pass data both ways between a FlashBASIC program and a data file.

When either of these two statements appear on the right of the assignment (= sign), data is passed to the program. When they are used to the left of the assignment, data is passed from the program to the file.

access(3) can be updated only from an input-conversion of a dictionary item. It cannot be updated as a conversion, correlative or in a "d-pointer" input-conversion.

The item-ids as well as the remainder of the attribute values within those items can be modified directly through the UP using input conversion or correlative attributes in dictionary items.

Syntax

ACCESS(num.expression)

Example

```
The following program is called from a "hot" key in the Update
processor.  It gets the current file name from access(11) and
executes a special "help" processor. The "access(21)" macro return
function is set to <ctrl>+z and p which rediplays the current
screen.
subroutine do.help(item)
file.name = access(11)
tcl "help ":file.name
access(21)= "z'p'"
return
This example works as well for Update as it does for AQL to
extract the last element of a value list. (Or subvalue for that
matter). The date stamp is attribute 11 of the item. "max" is the
total number of multi-valued elements on the stamp.  If "max" is
zero, there is no stamp. If max is non-zero, the last value is
extracted and placed into the active data value, "fld".
subroutine get.last.date.stamp(fld)
date.stamp = access(3)<11>
max = dcount(date.stamp,char(253))
if max = 0 then return
fld = access(3)<11,max>
return
```

ALPHA

evaluates the expression and returns "1 ("true") if every character in the string is alphabetic (i.e. a-z upper & lower case), or "0" (zero - "false") if any character is non-alphabetic.

The empty (null) string ("") is considered non-alphabetic.

Syntax

ALPHA(expression)

Example

```
if alpha(response) then array(1) = response
If the value of the variable, "response", is made up of only alpha
characters, it is assigned to the first element of "array".
```

AND

indicates that both components of a logical expression must be true.

Syntax

expression AND expression

Example

```
if tax = 0 and value > 0 then...
The expression "tax = 0 and value > 0" is true when "tax" equals
```

```
"0" and "value" is greater than "0".
if counter > max and value > 10 and y < 100 then...
The expression "counter > max and value > 10 and y < 100"
requires more all three logical expressions to be true for the
complete expression to be true.
```

ASCII

converts a string of EBCDIC-encoded characters into their equivalent ASCII values.

The "ascii()" function is the inverse of the "ebcdic()" function.

Syntax

ASCII(expression)

ASSIGNED

determines if the variable has been assigned a value.

If the variable has an assigned value, a non-zero value (true) is returned. If no value was ever assigned, 0 (false) is returned.

"assigned" can tell if common variables have been previously assigned.

"assigned" works well in FlashBASIC subroutines called from the "call" AQL processing code. It prevents files from being repeatedly opened and variables from continually being re-initialized.

Syntax

ASSIGNED(variable)

Example

```
if not(assigned(fv.entity)) then file entity
This determines if the file.variable "fv.entity" has been assigned
previously with the "file" statement. If it has not been assigned,
the "file" statement is executed.
```

AUX

enables or disables spooling to the auxiliary port

"aux on" enables spooling to the auxiliary port and saves the "p" and "h" flags of the current "sp-assign" status.

"aux off" reverses the effect of the last executed "aux on" command. It turns off auxiliary port printing, and restores the previous status of the "p" and "h" flags of the current "sp-assign" status. Note that "aux off" should NEVER be executed without a preceding "aux on" since "aux off" assumes that information has been saved by "aux on".

In the "expression" form, auxiliary printing is disabled when the expression evaluates to false. It is enabled when the expression evaluates to true. In an expression, 0 = "aux off" and non-zero = "aux on".

Syntax

AUX [OFF | ON | expression]

BEGIN CASE

See: case

BEGIN WORK

starts a transaction

A transaction is used to perform a series of updates to the database in an atomic manner. To begin the sequence, use the begin work command followed by the desired updates. To commit the updates to the database, use the "commit work" statement. If a problem develops during the series of writes preventing the correct completion of the transaction, then use the "rollback work" statement to undo all changes made so far.

The "work" or "transaction" keywords are optional and are provided purely for additional SQL compatibility.

The optional "name" parameter provides the ability to name the transaction. This name appears when using the TCL "transaction status" command to monitor transaction processing, or when the system is recovering after a halt. The name makes it easier to identify work-in-progress in these situations.

Once a transaction is active, the following special variables contain useful information:

@TRANSACTION This numeric value is non-zero when a transaction is active.

@TRANSACTION.ID This is the current transaction id number or zero if no transaction is active.

@TRANSACTION.NAME If the optional "name" parameter is specified in the "begin work" statement, then this variable contains that name. This variable is null if no transaction is active.

Nested transactions are not allowed and cause the "begin work" statement to fail.

The following operations work differently in a transaction:

clearfile	Clear all items in a file. This operation is not allowed within a transaction.
execute	Execute a system command. No special checking is done on executed system-level commands, and no atomicity is asserted by the system. The programmer is responsible for assuring the atomicity of all executed retrievals by doing the necessary "filelock" and "readu" commands before the execute. Updates performed by an executed system command do NOT participate in the transaction. The following execute operations generate a run-time abort if attempted within a transaction: "create-file," "clear-file," "delete-file," "create-index," "delete-index," and "resize-file."
filelock	Sets an exclusive lock on a file so that no other user may update or set an item lock on that file. It is necessary to execute this statement before an operation which may reference the file sequentially during a transaction.
fileunlock	Releases a file lock. This statement is ignored during a transaction since all locks are held until the transaction completes.
key	Return index information. This operation never participates in a transaction. Index entries are never updated until commit time, and no user-specific cache is provided. This means that it is invalid to access index entries for records which have been updated within a transaction before that transaction has been committed. Also, if strict transaction ordering is required, it is necessary to set a file lock previous to using the index.
read/readu	Read a record. By default, this operation will "see" updates already made by the process, but not yet committed (unless "transaction cache off" has appeared

	in the program previously). The "readu" form should always be used within a transaction to guarantee serialized schedules.
readnext	Read an item id from a select list. To use this validly within a transaction, each "readnext" MUST be followed by a read of the associated item id. This allows verification that the item has not been deleted within the transaction. Items which have been inserted within the transaction are returned by the "readnext" after all of the pretransaction id's are exhausted.
release	Release item locks. This operation is ignored during a transaction since all locks are held until commit or rollback.
select	Selects a list of items. To enforce serializable transactions, the user must do a "filelock" of the desired file before doing the "select". IF the select list is based on a prior "execute", then the "filelock" must be done prior to that execute.
write	Write a record. This operation always participates in the transaction (meaning that the effects of the write, including the release of any associated locks, are not committed to the database until the commit). All other variants of the WRITE follow identical rules including: "writeu," "writev," "writevu," "matwrite," and "matwriteu."
STOP	Stop the program. If a transaction is active, then the stop command or any other condition which terminates the program causes the equivalent of a "rollback" to occur.

The SYSTEM-COLDSTART procedure automatically rolls-backward any pre-commit transactions and rolls-forward (if possible) any post-commit transactions if they system halts unexpectedly.

The following additional features are not currently supported within a transaction:

- The callx and bridge correlatives.

- The use of remote (distributed) files.

- The use of non-D3 files.

Syntax

BEGIN {WORK | TRANSACTION} {name} {THEN | ELSE statement.block}

BREAK

enables or disables the <break> key.

The break key, if enabled, interrupts the current process and either invokes the FlashBASIC debugger or pushes a level.

"break off" disables the <break> key.

"break on" reverses the effect of the last executed "break off" command. If multiple "break off" statements are executed, an equal number of "break on" statements must be executed to return to a "breakable" state.

In the "expression" form, the break key is disabled when the expression evaluates to false. It is enabled when the expression evaluates to true. In an expression, 0 = "break off" and non-zero = "break on".

Syntax

BREAK OFF

BREAK ON

break expression

Example

```
break user = "fred"
If the value of "user" is "fred", this has the same effect as issuing
the statement "break on".
```

BREAK OFF

see "break".

BREAK ON

See: break

CALL

transfers control to an external FlashBASIC subroutine and optionally passes a list of arguments to it.

Arguments must be separated by commas (,). When passing arguments, the same number must occur in the "call" statement as are declared in the "subroutine" statement, which must occur as the first executable line of the external subroutine. There is a maximum of about 200 arguments. The subroutine can return values to the calling program in variable arguments.

The external subroutine must ultimately execute a "return" statement to continue program execution at the statement immediately following the "call" statement. Subroutines which do not return will simply terminate execution.

The "call @", or "indirect call" form allows the statement to use the subroutine name assigned to a specific variable.

Called subroutines must be compiled and cataloged separately from the calling program. The arguments passed between (to and from) the program and subroutine are not label-sensitive, but are order-sensitive. The arguments listed in the "subroutine" statement and the arguments listed in the "call" statement may be different. The subroutine receives the results of arguments in the order that they are specified in the argument list.

Variable arguments are passed to the subroutine at "call" time and from the subroutine at "return" time.

Arrays may be passed between programs and subroutines. The array in the program and called subroutine must contain the same number of elements. If dimensioned arrays are used, the arrays should be dimensioned exactly the same in both the program and subroutine (see the "dim" statement). Alternately, a "dim" statement may be specified without the actual number of elements. The array is properly initialized at run-time.

Arguments listed in both the "call" and "subroutine" statements should not be duplicated in the argument lists. Arguments that are also defined as "common" variables in both calling programs and subroutine programs should not be used in argument lists since the data is already accessible through the common allocations. Violation of these rules can result in unpredictable values being passed between the programs.

On releases 6.1.0 and above, it is possible to specify the file path name followed by a space followed by the actual subroutine name. To do this, the file path and program name can be

passed via a variable to an indirect call, or the string can be enclosed in quotes and embedded directly into the program text. Specifying a direct file reference and program name eliminates the need for cataloging subroutines when an application is used from other accounts.

On releases 6.1.0 and above, if the subroutine is not catalog'ed, then BASIC will attempt to locate the subroutine in the current program's dictionary. This eliminates the need to catalog most subroutines.

Syntax

CALL cataloged.program.name{(argument{,argument...})}

CALL @program.name.variable{(argument{,argument...})}

CALL "file.reference program.name"{(argument{,argument...})}

Example

```
direct call:
call process.lines(id,order.item(1))
-or-
call "process.lines"(id,order.item(1))
With or without quotes, this example still calls the "process.lines"
subroutine.
indirect call:
program.variable = "process.lines"
call @program.variable(id,order.item(1))
This example calls the subroutine name held as a string in the
variable "program.variable".
indirect call with full path name
program.variable = "dm,bp, process.lines"
call @program.variable(id,order.item(1))
```

CAPTURING

resumes the capturing effect from the previous "execute . . . capturing" statements in a FlashBASIC program.

The "capturing off" statement turns off the capturing effect from the previous "execute . . . capturing" statements in a FlashBASIC program. "capturing on" resumes the capturing effect.

In the "expresion" form, capturing is turned on if the expression evaluates to "true" (non-zero) and turned off if it evaluates to "false" (zero).

Syntax

CAPTURING [OFF | ON | expression]

Example

```
if system(26) then capturing off
"system(26)" checks to see if capturing is turned on.  If "on", this
turns it off.
capturing s > 5
If "s" evaluates to a number greater than 5, capturing is turned on.
```

CAPTURING

see "execute" and "get".

CAPTURING OFF

Disables output capturing.

CAPTURING ON

Enables output capturing.

CASE

delineates a conditional case construct.

Only the statements following the first "true" case statement are executed. Upon completion of these statements program execution continues immediately following the "end case" statement.

If all "case" statements evaluate false, program execution is continued at the first statement after the "end case".

A "case 1" statement always evaluates true and is used as the last case statement to be processed only when all previous case statements evaluate to false.

The "case" statement is equivalent to nested "if" expressions as follows:

if logical.expression then

{statement{s}}

end else

if logical.expression then

{statement{s}}

end

end

The statements between "case" statements are executed if the expression associated with the case evaluates to true. If the expression evaluates to false, the statements are not executed.

Syntax

BEGIN CASE

CASE logical.expression

{statement{s}}

.

CASE logical.expression

{statement{s}}

.

.

END CASE

Example

```
begin case
* check for operators response
case response = "p"
  printer on
case response = "s"
  echo off
case 1
  crt "response must be 'p' or 's'"
end case
In this example, if the response is "p", the "printer on" statement
is issued. The program would then branch to the next statement after
```

```
"end case". The "case 1" is executed only if response is neither "p"
nor "s".
```

CASING

toggles case sensitivity relative to input data and string comparisons in the current program and any called subroutines.

The FlashBASIC casing flag is automatically initialized to the case setting at TCL (which may be set using the "case" verb).

With "casing on", the strings "quit", "Quit", and "QUIT" are treated as three different strings of characters. With "casing off", they are all treated as the same string.

With "casing on", system date strings are in upper case.

See system(27) and system(28) for the ability to interrogate the current status of case sensitivity.

Syntax

CASING [OFF | ON | expression]

Example

```
casing on
crt "enter quit " :
input quit
if quit = "quit" then crt "lower case" end else crt "upper case"
Entering "quit" causes the program to display "lower case". Any upper
case character in input, like "Quit", would cause the program to
display "upper case".
```

CASING OFF

Disables case sensitivity.

CASING ON

Enables case sensitivity.

CAT

concatenates strings in an expression.

"cat" and the ":" operator are identical.

Syntax

expression CAT expression

expression : expression

Example

```
z = x cat y
This is equivalent to:
z = x : y
```

CHAIN

transfers processing control to TCL, which interprets and executes the command defined in the expression.

When the TCL expression finishes, control does NOT return to the program - "chain" is one-way only.

Syntax

CHAIN tcl.expression

Example

```
chain "file-save"
chain \off\
```

CHANGE

See: swap

CHAP

This statement is provided exclusively for licencee compatibility and has no effect on D3 execution.

Syntax

CHAP num.expression

CHAR

converts a number between 0 and 255 to its corresponding ASCII character equivalent.

Syntax

CHAR(num.expression)

Example

```
crt char(7)
This causes the terminal to beep once.
string = char(27):"k"
This concatenates an "escape" with a "k" and assigns it to "string".
equ am to char(254)
This creates a constant at compile time to reference attribute marks.
```

CLEAR

sets all local and common variables to 0 (zero) in a "main" program.

In a subroutine, only the local variables are affected. Named common variables are not affected by the "clear" statement.

CLEARDATA

clears the data stack.

Syntax

CLEARDATA

CLEARFILE

deletes all items in the specified file.variable previously opened with an "open" statement.

The default file.variable is cleared if no file.variable is specified. This statement works like the TCL "clear-file" command.

The file is not deleted, but all non-file type items within the data file or dictionary are deleted. The file retains its same base and modulo. If "dict" is specified in the "open" statement that defines the "file.variable", the dictionary of the file is cleared. If "data" is specified in the associated "open" statement, the data section of the file is cleared. If neither is specified in the associated "open" statement, the data section of the file is cleared.

Syntax

CLEARFILE {file.variable}

CLEARSELECT

clears the specified select list and releases all overflow associated with that list.

Any external list is also cleared.

Syntax

CLEARSELECT {list.variable}

CLOSE

closes the file identified by "file.variable". If file.variable is not defined, the file referenced by the default file.variable is closed. This statement must be used if a file is opened and then a "delete-file" is executed with an "execute" statement.

Files "close" by default when a program ends.

Syntax

CLOSE {file.variable}

Example

```
open 'testfile' to testfile then
close testfile
execute 'delete-file testfile'
end
execute 'create-file testfile ':newdictmod:' ':newdatamod
```

COL1()

returns the numeric column position of the character immediately preceding the substring retrieved in the most recently executed "field" function.

The parentheses following the function are required, and contain no arguments.

"col1()" and "col2()" allow non-system delimited strings to be handled like dynamic arrays.

If the "search" value does not appear in the string being searched by the "field" function, 0 (zero) is returned for the col1() function.

Syntax

COL1()

Example

```
beatles = "john*paul*pete*ringo*george"
name = field(beatles,"*",3)
start.pos = col1()    ;*  start.pos is 10
end.pos = col2()      ;*  end.pos is 15
beatles = beatles[1,start.pos]:beatles[end.pos+1,100]
This example removes "pete" from "beatles".
```

COL2()

returns the numeric column position of the character following the substring retrieved in the most recently executed field function.

The parentheses following the function are required, and contain no arguments.

"col1()" and "col2()" allows non-system delimited strings to be handled like dynamic arrays.

If the (search) value does not appear in the string being searched by the "field" function, and the field requested is "1", then col2() returns the length of the string. If the same situation occurs, but the field requested is "2" or higher, then the col2() returns 0.

Example

```
beatles = "john*paul*ringo*george"
name = field(beatles,"*",2)
startpos = col1()    ;*  start.pos is 5
endpos = col2()      ;*  end.pos is 10
beatles = beatles[1,startpos]:beatles[endpos+1,100]
This example has Paul splitting from the Beatles. (To do a solo
album).
```

COM

see "common".

COMMIT TRANSACTION

See: commit work

COMMIT WORK

Commits a transaction

This operation reliably logs all updates made since the last "begin work" command to the database in an atomic fashion. If the system halts during the commit, then the coldstart procedure automatically rolls forward the commit when the machine recovers. If the commit cannot be rolled-forward at that time (because of a missing file, trigger, etc), then that transaction is rolled back. If the roll back fails, then the transaction is aborted and an error message is printed to the screen and the errors file.

If the commit fails under normal operations, then the entire transaction is undone and the "else" statement.block is taken if present. Note that in this case, it is necessary to completely re-do all updates performed by the transaction (after correcting the problem which led to the failure).

It is invalid to have a bridge or callx on a file which is updated within a transaction. The inclusion of these items may cause unpredictable results.

Syntax

COMMIT {WORK | TRANSACTION} {THEN | ELSE statement.block}

Example

```
open "myfile" to f1
open "mylog" to f2
begin work
 readu total from f1,"total"
 total = total + 1
 write total on f1,"total"
 write total on f2,timedate();* create synchronized log
commit work else print "Could not update"
```

COMMON

declares data elements to share among different FlashBASIC modules.

The "common" (or "com") statement must appear before any variable. It is used to allocate variables to a common location, so that more than one program may have specified variables in a predetermined sequence.

Common variables (including dimensioned arrays) are allocated in the order they are declared. In the abscence of a "common" statement, variables are allocated in an undefined order.

Dimensioned arrays may be declared in a "common" statement by specifying the dimensions enclosed in parentheses. For example, "common a(10)" declares an array with 10 elements.

Arrays that are declared in a common statement must be declared with a constant number of elements, and may not be redimensioned with a "dim" statement.

The "common" statement may be used to share variables between programs and other programs or subroutines. It may also be used in FlashBASIC subroutines that are called from attribute-defining items. In this case, the values in the common variables are preserved between calls to the subroutine.

All standard variable types are allowed for common variables as well. The most frequent use of common is to store string or numeric values, but other types, such as file.variables or select variables are equally valid.

The order of variables in "common" statements is critical. The names of the variables are ignored and the order of appearance determines the association. The subroutine being called must have the same number (or less) values in its "common" statement than the main program.

The "/id/" option is used to specify a unique common area called a "named common" area. The "id" parameter must be unique within the program module where it appears. During execution, all program modules that declare named common areas using the same id reference the same variable space regardless of the location of the declaration within the program.

Multiple unique named common areas may be declared within the same program module. Named common space is preserved during an entire logon session.

All declarations of a named common in multiple modules must occupy the same amount of space (i.e., have the same number of variables and arrays, each array having the same number of elements). Multiple levels of a process share a given named common space which may be initialized at any level.

Arguments listed in both the "call" and "subroutine" statements should not be duplicated in the argument lists. Arguments that are also defined as "common" variables in both calling programs and subroutine programs should not be used in argument lists, since the data is already accessible through the common allocations. Violation of these rules can result in unpredictable values being passed between the programs.

Syntax

COMMON {/id/} variable{,variable...} {,array(dimension1{,dimension2})...}

Example

```
The main program:
common x,y,z(10)
...
call process.it
for i = 1 to 10
print z(i)
next i
...
end
The "process.it" subroutine:
subroutine process.it
common x,y,z(10)
....
for y = 1 to 10
call get.input
next y
...
return
The "get.input" subroutine:
```

```
subroutine get.input
common x,y,z(10)
....
input x
z(y)=x
....
return
The variables, "x,y,z(10)", are global within a given main program
and all of its subroutines.  Passing variables in common tends to be
more efficient than passing them as subroutine arguments.
Example of named common usage:
program get.data
common /mydata/ name,zip
input name
input zip
execute "display.data"
end
program display.data
common /mydata/ name,zip
print name
print zip
end
Both modules in this example are main programs.  All programs can
share information declared in a named common block.  The information
stored in /mydata/ is valid until the user logs off, making it
available to any other applications declaring a named common block of
the same id.
```

COMPARE

compares two dynamic arrays.

The "compare" statement scans through each element in string.exp1 to see if that element also exists in string.exp2. At completion, var1 contains a list of elements which exist in string.exp1 and in string.exp2. At completion, var2 contains a list of elements which exist in string.exp1 but do not exist in string.exp2.

Syntax

COMPARE string.exp1 TO string.exp2 {present var1} {missing var2}

CONVERT

searches a given variable and replaces each occurrence of a character by another.

Any character designated in "string.expression1" is replaced with the corresponding replacement character from "string.expression2". The correspondence is by column position in "string.expression1" and "string.expression2".

If the length of "string.expression1" is shorter than the length of "string.expression2", all characters in "string.expression2" beyond the length of "string.expression1" are ignored.

If the length of "string.expression1" is longer than the length of "string.expression2", all characters in "string.expression1" beyond the length of "string.expression1" are deleted from "variable".

Syntax

CONVERT string.expression1 TO string.expression2 IN variable

Example

```
convert char(254) to " " in text.string
All attribute marks are replaced with spaces in the variable
```

```
"text.string".
equ am to char(254)
convert am to char(10) in text.item
This converts the attribute marks in "text.item" to linefeed
characters.
x = "abc"
convert "abc" to "d" in x  ; crt x
This outputs "d".
x = "d"
convert "d" to "abc" in x  ; crt x
This outputs "a".
x = "abba"
convert "ab" to "zm" in x  ; crt x
This outputs "zmmz".
```

CONVERT

searches a given variable and replaces each occurrence of a character by another.

Any character designated in "string.expression1" is replaced with the corresponding replacement character from "string.expression2". The correspondence is by column position in "string.expression1" and "string.expression2".

If the length of "string.expression1" is shorter than the length of "string.expression2", all characters in "string.expression2" beyond the length of "string.expression1" are ignored.

If the length of "string.expression1" is longer than the length of "string.expression2", all characters in "string.expression1" beyond the length of "string.expression1" are deleted from the result.

Syntax

CONVERT(variable, string.expression1, string.expression2)

COS

calculates the trigonometric cosine of an angle of a specified numeric expression in the range 0 to 360 degrees.

Values that are less than 0 or greater than 360 are adjusted using the integer division function: mod(angle, 360).

Syntax

COS(num.expression)

COUNT

returns the number of occurrences of "string.expression2" within "string.expression1".

Syntax

COUNT(string.expression1, string.expression2)

Example

```
sentence = "sam is home alone"
print count(sentence,' ')
This displays the number of spaces in the string held in the variable
"sentence", which is equal to 3.
```

CRT

outputs unconditionally to the terminal display, regardless of whether or not the "printer on" condition is in effect.

This functions like the "print" command, but is not affected by any of the following:

1) the (p) option used with the "run" command at TCL.

2) the "printer on" statement.

3) a "heading", "footing" or "page" statement.

All of the "@" functions provided with the print "statement" are allowed.

When expressions are separated by commas, the next expression is output starting at the column position defined by the next output tab.

A colon at the end inhibits the output of a carriage return / linefeed following the output of the last printed line.

Syntax

CRT {expression{,expression..}{:}

Example

```
"crt" makes it easy to display status messages during printing.
printer on
for l = 1 to maxlines
  print line<l>             ;* to printer
  if rem(l,50) else crt l   ;* to screen
next l
printer close
printer off
The line counter "l" is displayed to the screen every 50 lines.
```

DATA

queues responses for use by subsequent input requests initiated from "chain", "enter", "execute", or "input" statements. The data is taken from the queue in the order in which it was added.

Multiple "data" statements are permitted. This is preferable to queueing all the data in one long statement.

Each expression becomes the response to one input request from succeeding processes. The "data" statement must be processed prior to an input request.

Note: Wherever "execute" is valid in these examples, "chain" or "TCL" may also be used. Text within an expression that is passed to UP must be enclosed in quotes; control characters must be outside the quotes. Text may be enclosed in single quotes and the expression enclosed in double quotes; or, the quotes may be used as shown in the examples here.

Syntax

DATA expression{,expression,...}

Example

```
data 1,2,3
input a
input b
input c
print (a + b) * c
This prints the number "9".
data '(customer,history'
execute 'copy customer *'
This copies all the items from the "customer" file into the
"customer,history" file.
The "data" statement can also be used to queue a command which uses
an active list:
data 'save-list archive'
```

```
execute 'select customer with last.puchase.date < "01/01/80"'
data 'copy customer'
data '(customer,history'
execute 'get-list archive'
data "a'H2O'r'Water'nxf"
execute "update doc example1"
crt "we're back..."
This example illustrates passing data (and commands) to the Update
processor. The "a" invokes the Update processor search function,
which is passed the string, "H20", the "r" invokes the "replace with"
prompt, which is passed the string, "Water". The "n" is the final
part of the search/replace, and indicates to replace all occurences.
After the search completes, the "xf" command files the item and
control returns to the program.
```

DATE

returns the current system date in internal format, as an integer number representing the number of days which have elapsed since December 31, 1967 (day 0 on the D3 calendar).

The parentheses following the function are required, and must not contain arguments. Without the parentheses, "date" is assumed to be a variable.

Example

```
number.days = date() - invoice.date
This calculates the number of days elapsed between the value of
invoice.date and the current system date.
weeks.old = (date() - invoice.date) / 7
This calculates the number of weeks old.
```

DCOUNT

returns the number of occurrences of the delimiter specified in "string.expression2" within "string.expression1", plus one.

The difference between this function and the "count" function is that 1 is added to the result unless the string referenced in "string.expression1" is null.

Syntax

DCOUNT(string.expression1, string.expression2)

Example

```
string = "a" : char(254) : "b" : char(254) : "c"
how.many = dcount(string,char(254))
for amc = 1 to how.many
  crt string<amc>
next amc
"how.many" contains the number of attributes in "string". In this
example, there are 3 and each attribute is displayed to the terminal
on a separate line.
input sentence
sentence = trim(sentence)
maxwords = dcount(sentence,' ')
for wordcnt = 1 to maxwords
  print field(sentence,' ',wordcnt)
next wordcnt
In this example, "sentence" is entered from the keyboard. The string
in "sentence" is trimmed of excess blanks, and the number of words
delimited by spaces is counted using "dcount". The words are then
printed on separate lines.
```

DEBUG

temporarily suspends execution of a FlashBASIC program and invokes the FlashBASIC debugger.

The program may also be "run" with the "d" option, or, if cataloged, activated with a "d" option. (see "run" and "catalog")

The "g" command in the FlashBASIC debugger returns control to the FlashBASIC program and continues execution at the next executable statement in the program.

Example

```
crt 'Command ':
input response
if response = "d" then debug
```

DEL

provides an alternate to the "delete" function for deleting a location from a dynamic array.

Syntax

DEL dynamic.array<ac.expression {,vc.expression ... {,sc.expression}}>

Example

```
equ vm to char(253)
beatles = "john":vm:"paul":vm:"pete":vm:"george"
del beatles<1,3>
This example deletes the value "pete" from the dynamic array
"beatles".  The reference to attribute 1 is required.  The absence of
attribute marks in "beatles" requires that it be treated as a single
attribute.
```

DELETE

function removes a specific attribute, value, or subvalue from a dimensioned or dynamic array.

The "delete" function returns a copy of the dynamic array without the deleted element. It does not change the original string unless specifically designated in an assignment statement.

Syntax

DELETE(dynamic.array.expression, ac.expression)

DELETE(dynamic.array.expression, ac.expression, vc.expression)

DELETE(dynamic.array.expression, ac.expression, vc.expression, sc.expression)

Example

```
equ vm to char(253)
names = "john":vm:"paul":vm:"george":vm:"ringo"
names = delete(beatles,1,3)
This example deletes the value "george" from the dynamic array
"names".  The reference to attribute 1 is required.  The absence of
attribute marks in "beatles" requires it be treated as a single
attribute.
read cust.item from cust.file,item.id else cust.item = ""
locate(date(),cust.item,date.flds;vx) then
write delete(cust.item,date.flds,vx) on cust.file,item.id
end
```

DELETE

removes a specific item from a file.

See the "delete" function for deleting an element of an array.

Syntax

DELETE {file.variable}, id.expression

Example

```
open 'customer' to customer.file else stop 201,'customer'
input customer.id
crt 'are you sure? ':
input ans
if ans = 'yes' then delete customer.file,customer.id
This program deletes a specific item from the "customer.file".  "yes"
is the only valid answer to perform the delete.
```

DIM

establishes a specific number of storage locations for a matrix of variables.

Arrays may be of one or two dimensions only. Individual elements within an array are accessed by appending the element number, enclosed in parentheses, to the array variable.

"rows" specifies the number of rows and "cols" specifies the number of columns of the array. These quantities may be any valid numeric expression.

A "dim" statement without the number of elements specified, such as in the case: "dim a()", may only be used in a subroutine. FlashBASIC run-time determines the number of elements after a "matread".

An array must be dimensioned prior to being referenced within a program. After an array has been dimensioned, it may be redimensioned later within a program. When the dimensioning parameters are changed, the new array will:

- retain those elements with the same indexes that were in the old array;

- have any new elements initialized to a unassigned state;

- lose any unreferenced elements (those elements in the old array with indicies not present in the new array).

If an array is passed to a subroutine in an argument list, it cannot be redimensioned within the subroutine. A passed array can only be dimensioned once within a subroutine. Local arrays may be redimensioned as necessary. When arrays are dimensioned both in a subroutine and within the calling program, the dimensions specified in the calling program will supercede those specified within the subroutine, if the subroutine's array was dimensioned as an empty dimension. The number of dimensions must be the same in both the calling program and called subroutine.

Syntax

DIM{ension} array.variable({rows{,cols}})

Example

```
dim rec(20),table(3,3)
"rec" is a single dimensional array with 20 subscripted variables.
"table" is a two dimensional array with 9 subscripted variables.
x = 20
dim a(x)
input y
dim b(y)
dim c(x,y)
In D3, "dimension" can be handled by FlashBASIC runtime.
subroutine process.receipt(receipt.item)
dim work.list()
```

```
...
matread work.list ...
This example illustrates a "dim" statement without arguments. It
resolves the size of the array after the "matread".
```

DIMENSION

see "dim"

DO

Part of "loop" construct.

DQUOTE

extracts a double-quoted string from string.expression.

The "dquote" function returns the first substring found which is surrounded by double-quotes. If no substring is found matching this criteria, then a null string is returned.

Syntax

DQUOTE(string.expression)

DTX

converts a given decimal number to its corresponding hexadecimal equivalent. The number is first converted to an integer by truncating the fractional part.

The expression can be in the range {+/-}140737488355327*(10(-p)), where the "p" represents the precision number.

Syntax

DTX(num.expression)

Example

```
input fid
crt "hex of " : fid : " is " : dtx(fid)
The variable "fid" is a decimal Pick frame number (frame-id).  This
program shows the frame-id in hexadecimal.
```

EBCDIC

converts a string of ASCII-encoded characters into their equivalent EBCDIC values.

Syntax

EBCDIC(string.expression)

Example

```
writet ebcdic(rec) else ...
The ASCII string "rec" is written to tape in EBCDIC.
```

ECHO

toggles terminal echo on or off.

Typically, terminals attached to the system operate in "echo plex" or "full duplex" mode. Characters typed at a terminal are sent directly to the computer and must be "echoed" back to the terminal before they are displayed on the screen.

"echo off" stops this echoing; that is, characters entered at the terminal are not seen on the screen. However, the characters are sent to the computer and processed as usual; also, any output that originates within the computer is displayed on the screen.

"echo on" returns to the normal mode of full duplex echoing.

An expression that returns zero is equivalent to "echo off". Any other value returned is equivalent to "echo on".

Syntax

ECHO {[OFF | ON | num.expression]}

Example

```
crt "enter password " :
echo off
input password
echo on
The password does not display as it is being typed in.  Once the
password is accepted, "echo" is enabled.
secure = userrec<5>
echo secure >= 6
Echo is disabled if the "secure" variable is 5 or less.  Echo is
enabled for 6 or more.
echo off
...
input string
On systems being run in "half duplex", there is no need to echo the
character to the terminal display.  The terminal has already taken
care of display.  This proves invaluable when terminals are connected
through networks or complicated multiplexing.
```

ECHO OFF

Turns off terminal character echo.

ECHO ON

See: echo

ELSE

is the in-line clause for the "if" statement to execute when it evaluates to false.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

ELSE statement.block

Example

```
if x = 5 then print "Five" else print "Not Five"
if x = 5 then
  begin
    print "Five"
  end else print "Not Five"
if x = 5 then
  begin
    print "Five"
  end else
    print "Not Five"
  end
```

END

indicates both the end of a series of statements executed conditionally from a "then" or "else" condition or the physical end of the program.

The "end" statement is used in two ways:

1) physical program end (optional)

.....statements.....

.....body of program....

.....statements.....

end

The trailing "end" statement is not required on FlashBASIC source code, but is helpful for program readability. Any statements after the "end" statement (in this case) are not compiled.

2) Multi-line then/else statements must have terminating "end" statements:

if logical.expression then

statement

.

end

Example

```
10 *
input ans
if ans='n' then
print 'Please retype ':
goto 10
end
This example reprompts for "ans" if the previous "ans" is "n".
open 'md' to md else
print 'Cannot open file'
stop
end
crt 'File opened'
If the "md" file does not exist, a warning message is printed and
program execution stops.  Notice that "stop" is the FlashBASIC
runtime directive which indicates the termination of runtime.  The
"end" statement indicates the end of the multiple line "then"
statement.  Without the "stop", program execution continues with the
next statement after the "end" statement.
```

END CASE

Terminates case construct.

ENTER

transfers control to another cataloged FlashBASIC program.

The program that is executed using the "enter" statement must be a cataloged verb in the current md, and may not be a subroutine. The program executing the "enter" statement need not be cataloged.

All variables passed between programs must be declared by a "common" declaration in all program segments that are to be entered. All other variables are initialized upon entering the program. The common area grows or shrinks if the size is different between programs.

Control does not return to the original program. In this way, "enter" is much like "chain" except that "enter" must be used to activate another FlashBASIC program and nothing else.

The "at sign" (@) indicates that the program name is stored in the specified program variable. Without the "@", the "variable" name is taken literally.

Syntax

ENTER cataloged-program-name

ENTER @variable

Example

```
program first.prog
common x,y,z
x = 5
enter second.prog
program second.prog
common x,y,z
print x
stop
 In the above example, a "5" would be output.  In the example below,
the program name is stored in a variable.
program first.prog
common x,y,z
x = 5
pgmname = "second.prog"
enter @pgmname
```

EQ

"Equal to" operator

EQU

Declares constants at compile time.

EQUATE

a compiler directive which declares a constant at compile time or a synonym of another variable or array element.

Note: "expressions" in an "equ" are limited to constant expressions, variables, and character strings.

"equate" can also be used to assign meaningful variable names to array elements. Once "equate" has been used to assign a value to a symbol, the variable cannot be used as a variable and cannot be reassigned a value. Equate definitions must be placed before any usage of the equated symbols. Failure to do so may generate compile-time errors.

Syntax

EQU{ATE} symbol TO CHAR(constant) {,variable TO expression...}

EQU{ATE} symbol TO variable

Example

```
equ form.feed to char(12)
equ ring.bell to char(7)
equ attribute.mark to char(254)
...
read array from file,id...
amcmax = dcount(array,attribute.mark)
....
if ctr=pagemax then
  print form.feed
  crt ring.bell:
```

```
end
...
This example shows how "equ" can make more descriptive cryptic
references like "char()".
equ customer.name.attribute to 1
equ title to "REF"
equ invoice.number to customer.item(13)
...
print @(15,0):title
print customer.item(customer.name.attribute)
...
read invoice from invoice.file,invoice.number then ...
"equ" designates a synonym for the numeric constant 1, the string
"REF" and the 13th variable in the dimensioned array "customer.item".
equ x to a
a = 3
print x
The symbol "x" is a synonym for the variable "a" and can be used
interchangeably.
```

EREPLACE

See: "replace"

ERROR

displays the error message from the "messages" file and continues execution of the program.

The parameter{s} are passed to the error message handler and displayed according to the requirements of the item in the "messages" file.

Syntax

ERROR message.number, "parameter" {,"parameter"...}

Example

```
read item from cust.file,"test" else
  error 202,"test"
 end
The "error" statement in this example displays:
[202] 'test' is not on file.
```

ERROR

retrieves the TCL command used to activate the program and loads it into a specified variable.

The parentheses following the function are required, and never contain any arguments. Without the parentheses, it is just a variable.

Example

```
test
001 crt error()
If the one-line, cataloged program called "test" is executed as
shown, it displays the command line.
```

EXCHANGE

converts characters in string.expression1.

This function works identically to the "convert" function except that the second two arguments are specified as hexadecimal numbers.

Syntax

EXCHANGE(string.expression1, string.expression2, string.expression3)

Example

```
print exchange("abc",dtx(seq("a")),dtx(seq("b")))
The result, "bbc", will be printed to the terminal.
```

EXECUTE

"pushes a level" temporarily and performs any valid TCL expression, then continues execution of the FlashBASIC program.

The TCL expression may be any valid TCL command, including AQL sentences, verbs, macros, menus, PROC's, or other cataloged FlashBASIC programs.

The results of the TCL command may be assigned to a variable for later processing by using the optional "capturing" clause.

Input may be passed to the TCL statements using the "data" statement prior to the "execute" statement. After the "execute" statement completes, the data queue, and any external "active" lists, are reset. Multiple "data" statements may be passed when they are separated by attribute marks.

An alternate method of stacking data is using "execute" in the form:

execute "TCL.command" : char(254) : "input data"

The optional "returning" clause directs any error message item-id(s) generated as a result of the "execute" statement into a variable. Each error message item-id is separated by a space.

In the "capturing" clause, Carriage return/linefeed combinations are converted to attribute marks, which allows the variable to be treated like a dynamic array. Clear screens and form-feeds are converted to nulls.

Certain FlashBASIC statements, like "input @" and "crt", don't put anything into the capturing stream.

The "capturing on" and "capturing off" statements are provided to enable or disable the output from the "capturing" portion of the "execute" statement.

An active list that is created by the executed command is passed back to the program. The list may be used by the "readnext" statement or it may be assigned to a specific variable using the "select to" statement for later use with a "readnext" "from" statement.

An active list generated by a TCL statement may be passed to a another TCL statement executed from a FlashBASIC program.

Control does not return from an execute that issues an "off" command.

The "to" and "logto" verbs push a level, go to the new account, and return to the next statement in the FlashBASIC program on the original account when complete.

The following verbs may alter the program environment when returning to the next program statement: Spooler verbs, tape control verbs, "debug", "charges", "term", and "tabs".

Output printed from indexes and FlashBASIC calls which are active during a file update are captured.

Each level of "execute" builds a new process workspace area. As the number of levels increase, so do disk space requirements. The maximum number of levels is 16.

A second version of execute is available which provides more control over data and select list handling. This is shown in the second line of the syntax. This additional syntax requires that the extended options be set. See "$options" for more information.

The "stacking" clause allows the user to stack data for the TCL statement. This clause is identical to using a "data" statement before the "execute".

The "passlist" clause creates a copy of the select list and passes it to the TCL command. Note that select lists which are "internally" linked to a file will be expanded to a standard list of item-id's so that the upper level may access the list.

The "rtnlist" clause provides essentially the same functionality as a "select to list.variable" done after the execute. The only difference is that rtnlist does not drop into the BASIC debugger if an external list is not available. Instead, it returns a null string in this case.

Syntax

EXECUTE tcl.expression {RETURNING variable} {CAPTURING variable}

EXECUTE tcl.expression {[STACKING | ,//in.<] expression} {[passlist | ,//select.<] list.variable} {[rtnlist | ,//select.>] list.variable } {[RETURNING | SETTING] variable} {[CAPTURING | ,,//out.>] variable}

Example

```
execute "sort staff by name name total salary (p)"
data "(qfile"
execute "copy newac *"
execute "listu" capturing output
execute "sselect invoices" returning errnum
execute \sort orders with date > "1/1/92"\
execute "sselect staff by hire.date by name"
execute "save-list staff.list"
execute "get-list staff.list"
execute "list staff name hire.date"
```

EXECUTE

executes a Unix command from within FlashBASIC.

Any valid Unix command may be executed through the standard FlashBASIC "execute" statement. By using the format, "!Unix.command", it is not necessary to create Unix verbs in a D^3 md.

The variable specified by "capturing" receives the information normally directed to "stdout". Each line is separated by an attribute mark. There is no limitation on the size of the captured data. Unix tabulations characters are not replaced. To replace tabulations, use pr(1), for instance, as a filter (see example below). "stderr" remains associated to the user's terminal.

To capture the ouput normally directed to to "stderr", the standard Unix shell syntax may be used to redirect "stderr" to "stdout" as in the example below:

execute 'cc -o mypgm mypgm.c 2>&1' capturing cc.result

See the Unix User's Reference Manual for information about sh(1).

The variable specified by "returning" receives the exit code of the "Unix.command". If the command cannot be executed, the first value returned is "-1", followed by a space and the decimal value of the error code, "errno", in the range 0 to 255.

Currently, D3 implementations do not allow reading data which has been stacked by a previous FlashBASIC "data" statement.

Syntax

EXECUTE "!Unix.command" {CAPTURING variable} {RETURNING variable}

Example

```
The following is an example of a simple file transfer:
import
001 *
002 ! Copy a file from Unix
003 tclread line
004 line=trim(line)
005 if line='' then goto usage
006 *
007 uname=field(line,' ',2)
008 pfile=field(line,' ',3)
009 pname=field(line,' ',4)
010 if uname='' or pfile='' or pname='' then goto usage
011 open pfile
012 *
013 execute '!exec cat ':uname capturing item
014 write item on pname
015 stop
016 *
017 usage:*
018 crt 'Usage: import unixfile pickfile item'
The "cat" Unix command copies a file to "stdout", which is
"captured".
To expand the tabulations into the appropriate number of spaces to
set tabulations to columns 5, 9, etc.. replace line 13 by (for
example):
execute '!exec cat ':uname:' | pr -t -e4 ' capturing item
Note in this example the usage of "!exec Unix.command" to avoid the
creation of an intermediate shell.
```

EXIT

forces an "early" exit from a loop that is under "for ... next" or "loop ... repeat" control.

The program continues with the next executable statement after the "next" or "repeat" statement.

Example

```
loop
  input guess
  if guess='end' then exit
while guess > 1 and guess < 100 do repeat
```

EXP

returns the exponential of a numeric expression, that is, base e, which is 2.718281828 rounded to "precision" to the power of the numeric expression.

This function is the inverse of the "ln" (natural logarithm) function.

The value of the expression can be in the range {+/-}140737488355327*(10(-p)), where "p" is the precision.

Syntax

EXP(num.expression)

Example

```
print exp(5)
148.4132
```

EXTRACT

retrieves a specific attribute, value, or subvalue from a dimensioned or dynamic array.

"extract" makes a copy of an element in a dynamic array rather than physically removing the element. The original syntax of the "extract" function is equally as valid as the dynamic array reference method.

Syntax

EXTRACT(dynamic.array.expression, ac.expression)

EXTRACT(dynamic.array.expression, ac.expression, vc.expression)

EXTRACT(dynamic.array.expression, ac.expression, vc.expression, sc.expression)

Alternate method, using dynamic array reference symbols:

dynamic.array.variable<ac.expression>

dynamic.array.variable<ac.expression, vc.expression>

dynamic.array.variable<ac.expression, vc.expression, ...

... sc.expression>

Example

```
print extract(customer.item,1,0,0)
print extract(customer.item,1)
print customer.item<1>
All of these examples retrieve the entire first attribute from the
dynamic array, "customer.item".
print extract(customer.item(1),1,2)
print customer.item(1)<1,2>
Both of these examples retrieve the second value from the first
attribute of the dynamic array, "customer.item".
name = extract(customer.item(1),1,1)
name = customer.item(1)<1,1>
Both of these examples assign the variable "name" to the first value
from the first attribute of the array "customer.item".
```

FIELD

returns a substring from a string expression, by specifying a delimiter and the desired occurrence.

The "occurrence" is the number of delimiters to be skipped, minus one, prior to extraction of the substring. After executing the "field" function, the "col1()" and "col2()" functions return the beginning and ending positions within the string at which the delimiters were found.

The search delimiter is limited to a single character. Any additional characters are ignored.

Syntax

FIELD(string.expression, search.delimiter, num.expression)

Example

```
gl.account = "02*4000*11"
company.code = field(gl.account,"*",1)
This causes all characters up to the first asterisk in "gl.account"
to be assigned to the variable "company.code". As a result,
afterward, "company.code" contains the value, "02". "col1()" has a
value of 0, and "col2()" has a value of 3.
acct.number = field(gl.account,"*",2)
This causes all characters from the first to the second asterisk to
be stored in the variable, "acct.number". "acct.number" contains the
value, "4000".
```

FILE

compiler directive to use attribute definition items in the file's dictionary while compiling the program.

Multiple files can be specified with the same statement, separated by commas. File paths are currently not allowed in the "file" statement.

The "file" statement is typically used to allow the use of attribute definition items to identify the attribute count ("ac") of the field and automatically apply dictionary correlative codes to the field value in the item.

When the FlashBASIC compiler encounters a "file" statement, it opens the dictionary of the file specified by "file1", creates the executable code to open the data portion of the file during execution, scans the FlashBASIC code for references to the given file and dimensions an array with the same name as the file specified.

The size of the dimensioned array is determined by the compiler. It scans through the program looking for references to attribute names as dimensioned array subscripts (e.g. file1(credit.limit)). After it has found all references to the given file, it takes the "highest" attribute count reference derived, adds 1 to it, and uses the resulting value as the size of the dimensioned array.

After an item is read from the file, attributes within the item may be referenced by using the attribute name from the associated dictionary as the array subscript. Correlative processing codes found in the file dictionary are executed, but output-conversions are not. Values within an attribute can be specified by appending a command and a value count to the attribute name or number.

"read" and "write" statements can not specify the file.variable when the array name "file1" is used as the variable.

The file pointer is assigned to the file.variable in the form "fv.file1" and references the data section of the file opened.

Syntax

FILE file1 {, file2 ...}

Example

```
file entity
id = "100"
read entity from id then
if entity(name) = "" then crt "no name!"
end
The file "entity" is bound by the "file" statement.  When item "100"
is successfully read, the "name" field is checked. If the name field
is null, "no name" is output to the terminal.
```

```
fv.entity = access(1)
if not(assigned(fv.entity)) then file entity
This determines if the file.variable "fv.entity" has been assigned by
Access of the Update processor. If it has not been assigned, the
"file" statement is executed. This technique is particularly useful
when calling subroutines from AQL, and prevents having to reopen a
file each time the routine is called. Using this logic, the "file"
statement is executed once.
```

FILELOCK

sets an exclusive lock on an entire file.

Once a file lock has been set, any other port which attempts to update, item lock, or file lock that file will fail.

If file.variable is not specified, then the default file variable is assumed.

The filelock is useful preventing updates to a file while a sequential operation is running (like a BASIC select).

If the file is already locked (meaning another user has a file or an item lock, or is currently updating that file) and the optional locked clause is present, then statement.block is executed. If no locked clause is present, then the statement blocks until the situation is resolved.

File locks are displayed by the "list-locks" TCL command as special item locks with item id's of "*" and hash's of 0. If necessary, these locks can be cleared abnormally with the "unlock-file" command.

Syntax

FILELOCK {file.variable} { LOCKED statement.block}

FILEUNLOCK

releases an exclusive lock on an entire file.

Once a file lock has been set, any other port which attempts to update, item lock, or file lock that file will fail.

If file.variable is not specified, then the default file variable is assumed.

The filelock is useful preventing updates to a file while a sequential operation is running (like a BASIC select).

Syntax

FILEUNLOCK {file.variable}

FMT

This function is identical to performing a BASIC mask.

Syntax

FMT(string.expression, mask.expression)

FOLD

"folds" a string.expression into a string of a given length or lengths.

The "fold.length.expression" specifies the length(s) at which the string.expression will be folded. If the fold.length.expression is omitted, it defaults to 25. Multiple numeric expressions, separated by value marks, may be specified in this parameter.

The text is folded so the length of the first line is less than or equal to the value of the first numeric value in the fold.length.expression, the length of the second line is less than or equal to the value of the second numeric value in the fold.length.expression, and so on. If more strings exist than corresponding number of fold.length.expressions, the last fold.length.expression is applied to the remaining strings. If possible, the text is folded on a space.

The "delimiter" parameter is the delimiter used in the folded text. This parameter is required by the compiler, but the parameter may be null, in which case a value mark (char(253)) is used. See example 2.

Syntax

FOLD(string.expression, fold.length.expression, delimiter)

Example

```
equ svm to char(252)
input string
string=fold(string,10,svm)
print string
When the string:
this is a test string to demonstrate "fold"
is entered, the string is embedded with the requested subvalue marks
as follows:
this is a\test\string to\demonstrat\e "fold"
a = fold(a,25,"")
-or-
delim = ""
a = fold(a,25,delim)
In both of these examples, a value mark is used as the fold
delimiter.
```

FOOTING

designates a text string composed of literals and special options to output at the bottom of each page.

The "footing" expression is treated as any FlashBASIC character string and may be surrounded by double-quotes (") or backslashes (\) or the expression can be contained in a variable. Embedded "footing" options must be enclosed within single quotes, to distinguish them from text. Multiple options may be enclosed in the same set of single quotes.

The "footing" may be changed by a new "footing" statement, and the new footing is output when the end of the current page is reached.

The "page" statement is used to force pagination.

The "system" function maintains line and page counters while "heading" and "footing" are active.

Syntax

FOOTING string.expression

FOOTING "{{text} {'options'}...}"

Options

* see "options: footing".

Example

```
footing "'lc'page 'pn' printed at 'tlc'"
The above footing statement uses a literal string for the "footing"
expression and produces a footing that looks as follows:
```

```
page '1' printed at 13:45:00 10 FEB 1992
footstring="'lc'page 'pn' printed at 'tlc'"
footing footstring
The same footing expression can be placed in a variable, and the
variable name used in the "footing" statement.
footing "'lc'page 'pn'":message:"'l'"
In this example, the "footing" expression is a compound string
expression.
```

FOR ... NEXT

an iterative, incremental loop statement used to repeat a sequence of statements for a specific number of iterations.

A "for ... next" loop executes a set of statements for successive values of a variable until a limiting value is encountered. Such values are specified by establishing an initial value ("expression1"), a limiting value ("expression2"), and an optional increment value ("expression3") to be added at the end of each pass through the loop.

If the loop-ending condition is not met, the loop variable is incremented by 1 or by the value of the "step" expression and program control transfers back to the beginning of the loop. This looping continues until the ending condition is met or the loop is explicitly exited with an "exit" or "goto".

"for ... next ...until" is a conditional incremental loop statement. It executes while the expression following the "until" clause evaluates to false. The "until" clause must appear on the same line as the "for ... next" statement.

General form:

for variable = num.exp to num.exp {step num.exp} ...

... until logical.expression

statement{s}

.

.

next variable

"for ... next ... while" is also a conditional incremental loop statement. It executes while the expression following the "while" evaluates to true. Like the "until" clause, the "while" statement must appear on the same line as the "for ... next" statement.

General form:

for variable = num.exp to num.exp {step num.exp} ...

... while logical.exp

statement{s}

.

.

next variable

The expression in the optional "while" or "until" clause is a logical expression. If the "while" expression is true, the "for ... next" loop continues. If the expression is false, program control passes to the statement immediately following the accompanying "next" statement.

If the "until" expression is true, program control passes to the statement immediately following the accompanying "next" statement. If the expression is false, the "for ... next" loop continues. When the loop is exited, the loop variable retains its last value. The variable in the "next" statement must be the same as the variable in the "for" statement.

"until" and "while" may not be used in the same "for ... next" loop.

Syntax

FOR variable = num.expression1 TO num.expression2 {STEP num.expression3} {[WHILE | UNTIL] logical.expression}

statements{s}

.

.

NEXT variable

Example

```
number.values = dcount(array(13),char(253))
for i = 1 to number.values
  crt i "l#4" : array(13)<1,i>
next i
This example displays every value in array element 13.
for i = 100 to 1 step -1
  crt i "l#4" :
next i
This is a decrementing counter beginning at 100 and ending at 1.
number.values = dcount(array(13),char(253))
for i = 1 to number.values until array(13)<1,i>=''
  print i
next i
This example prints an incrementing counter based on the number of
values in array element 13.  The loop terminates as soon as a the
last value is tested (number.values) or the first null value is
found.
All of the above "for ... next" constructs can be constructed using
the "loop ... repeat" constructs.
for i = 1 to 99 until array(i) = ""
  crt array(i)
next i
There are 99 elements in the dimensioned array "array".  This example
prints all elements until a null value is found.
t=time()
timeout=10
for try = 1 to 5 until ((time() - t)>timeout or system(14))
  sleep 2
next try
This for..next loop has a maximum of 5 iterations. The loop
terminates when the elapsed time in seconds exceeds 10, or there are
characters in the type-ahead buffer.
for i = 1 to 99 while array(i) # ""
  crt array(i)
next i
This loop terminates when the 99th entry is displayed or the first
null entry is encountered.
for i = 1 to 99 while not(array(i) = "")
  crt array(i)
next i
```

FOR NEXT ... UNTIL

Syntax

see "for ... next".

FOR NEXT ... WHILE

see "for ... next".

GE

denotes a "greater than or equal" conditional between two elements.

Syntax

expression GE expression

Example

```
if date ge "6/1" then print "yes"
```

GET

gets raw characters from the specified port.

"variable" receives the input. Printable characters are echoed to the line, unless the program issues a 'ECHO OFF' statement. Control characters are included, but not echoed to the screen. No prompt is displayed.

"length" is the maximum characters to get. If no length is specified, and there is no "until" clause, "1" (one) is the default. If the length is specified, but this number of characters is not entered, the system waits indefinitely or until the "waiting" clause times out, the "termination.character" is entered, or the "length" is satisfied. If the "length" is satisfied, the "then" clause is executed.

"setting" specifies the number of characters to be returned. If the "length" expression is less than the number of characters specified by "setting", input terminates at the length. If the "length" expression is greater than "setting", only the number of characters specified by the "setting" are returned to the variable.

"from" is the port from which to get characters. This port must have been previously attached with the "dev-att" command. If an invalid port number is specified or the port is not attached , then the "else" clause is taken.

"until" defines a set of characters which terminate input. If the "x" option is not in effect, the input also terminates on a char(255) (x'ff'). The "termination.character" is not included in the returned string. When a "termination.character" is received, the "then" clause is executed.

"returning" returns the character which terminated input. If the "returning" variable is null, and either no length is specified, or the returned data length is less than the length limit, a segment mark (x'ff') is assumed as the "termination.character".

"waiting" specifies how long, in tenths of a second, the system should wait until an input termination condition exists (such as satisfying the length requirement or the "until" clause). If a timeout occurs, all input received up to this point is returned in the "variable", and the "else" clause, if present, is executed.

"getx" returns an exploded ASCII hexadecimal string which allows binary data to contain a decimal 255 (x'ff').

Syntax

GET{x} variable{,length} {SETTING character.count} FROM port.expression {UNTIL termination.character(s)} {RETURNING termination.character(s)} {WAITING seconds.expression} {THEN | ELSE statement.block}

Example

```
get xx,80 setting 5 from 10 until (char(13):char(10)) returning term
waiting 60
This will get up to 80 characters from port 10 until that port
generates a carriage return (char(13)) or linefeed (char(10)).   If
more than 6 seconds elapse, then the statement terminates.  Note that
xx will contain a maximum of 5 characters.  The "term" variable will
contain which character actually terminated input.
getx xx,10 from 10 waiting 50
If "HELLO" is entered on port 10, xx will contain "48454C4C4F". If
the string, "HE<backspace>LLO" is entered, xx will contain
"4845084C4C4F", where the "08" is the backspace character.
```

GOSUB

transfers control to a local subroutine identified by a statement label within the program. Control returns to the next statement after the "gosub" statement when the "return" statement is encountered.

Syntax

GOSUB statement.label{:}

GOTO

Transfers control to the location in the FlashBASIC program that begins with the given statement label.

On statement labels beginning with a non-numeric value, the colon following the "statement.label" is optional in the "goto", but is required following the actual statement label.

Syntax

GO{TO} statement.label{:}

GO TO statement.label{:}

Example

```
goto 99
go to error
if ans='y' then go begin
```

GT

logical operator within conditional expressions to represent a "greater than" condition.

">" is identical to "gt" when used as a logical operator.

Syntax

expression GT expression

Example

```
if x gt 10 then stop
```

HEADING

designates a text string composed of literals and special options to output at the top of each page.

The "heading" expression is any valid literal string, string expression, or variable. A literal string must be enclosed in either double-quotes (") or backslahes (\) since "heading" options must be

enclosed within single quotes, to distinguish them from text. Multiple options may be enclosed in the same set of single quotes.

The "heading" may be changed by a new "heading" statement, and the new "heading" is output at the beginning of the next page. The very first time the "heading" is activated, it is output directly on the screen.

The "page" statement is used to force pagination.

The "system" function maintains line and page counters while "heading" and "footing" are active.

Syntax

HEADING string.expression

HEADING "{{text} {'options'}...}"

Options

* see "options: heading".

Example

```
heading "'lc'page 'pn' printed at 'tlc' this end into shredder
first.'l'"
page 1 printed at 10:22:30 29 feb 1992
this end into shredder first
head = \'c'this is a heading'l'\
heading head
In this example, the heading option string is assigned to a variable
("head") and subsequently assigned as the heading.
```

HUSH

enables or disables hush mode.

When hush mode is active, all output, including captured data, is suppressed.

Syntax

HUSH OFF

HUSH ON

HUSH expression

ICONV

converts a value from its external format to its internal equivalent, according to the processing code being applied.

Syntax

ICONV(string.expression, conv.expression)

Example

```
check.amount = 12.5
print iconv(check.amount,"mr2")
1250
This example converts "check.amount" to internal format, using the
"mr2" conversion which multiplies by 100 and truncates the decimal
portion.
check.date = "10/28/93"
print iconv(check.date,"d")
9433
This example converts "check.date" to internal format, using the "d"
```

```
conversion. (see the "d" processing code.)
string = 'john*paul*george*ringo*stuart'
print iconv(string,'g0*2')
john*paul
This example performs an Access group extract of two groups.
input ans,1
if iconv(ans,"p('y');('n')")#"" then ...
If "ans" contains "y" or "n", the statements following the "then"
clause are executed.
```

IF

tests the result of a logical expression. Depending on whether the expression evaluates to either "true" or "false", the statement(s) following the "then" or "else" clauses respectively are executed.

The statements may be placed on a single line (single-line format) or in a block structure (multi-line format). The two formats are functionally identical.

If more than one statement is contained in either the "then" or "else" clause, they must be separated by semicolons.

The single-line format:

if logical.expresssion then statements {else {statements}

where "statements" represents {statement{; statement{;...}

The single and multiline formats may be combined. Thus, the general format of the multiline statement takes on three forms as shown below:

form 1:

if logical.expresssion then statements else
statements
.
.
end

form 2:

if logical.expresssion then
statements
.
.
end else statements

form 3:

if logical.expresssion then
statements
.
.
end else
statements
.
.
end

form 4:

if logical.expresssion else
statements
.
.
end

form 5:

if logical.expresssion then
statements
.
.
end

If the logical.expresssion evaluates to non-zero, it is considered true. If it evaluates to 0, it is considered false.

If there is no clause for the current condition, the next sequential statement following the entire if statement is executed.

The "then" clause of an "if" statement is optional if the "else" clause is present. However, one or the both must be present.

Complex conditions may be tested by using "and" and "or" connectives. No matter how complex the overall expression may be, each condition eventually evaluates to a simple true or false.

Many statements accomodate the "then/else" construct, and more information is available under the topic, "then/else construct".

Syntax

IF logical.expression THEN statement.block

IF logical.expression THEN statement.block ELSE statement.block

IF logical.expression ELSE statement.block

Example

```
if answer = 'exit' then stop
if answer = 'y' then print 'ok'; cnt=cnt+1; goto 20
if answer = 'y' then
  print 'ok'
  cnt=cnt+1
 end
if answer = 'exit' then gosub 100 else gosub 200
if answer = 'y' then print 'ok';cnt=cnt+1 else print 'no good';stop
if answer = 'y' then
  print 'ok'
  cnt=cnt+1
 end else
  print 'no good'
 end
```

IFR

tests the result of a logical expression. Each attribute of the first element of the expression is tested against the target. Depending on whether one of the attributes in the expression evaluates to either "true" or "false", the statement(s) following the "then" or "else" clauses respectively are executed.

See the "if" statement for more information on clause structures.

Syntax

IFR logical.expression [THEN | ELSE] statement.block

IFR logical.expression THEN statement.block {ELSE statement.block}

IN

accepts a single character of input from the keyboard, without displaying a prompt character or requiring a <return> following the input.

A character is retrieved from the type-ahead buffer without being processed and is converted to its equivalent decimal value before being assigned to the variable.

For example, when the <backspace> key is pressed, the "in" statement retrieves an "8". When the <linefeed> key is pressed, the value is "10".

The "time.expression" specifies the maximum time that the system waits for input before returning to the FlashBASIC program, until either of the following occur:

When a character is entered, the "then" clause is taken.

When a character is not entered in the allocated time, the "else" clause is taken.

The "time.expression" is expressed in tenths of a second, as a positive integer from 1 to 32767. If "time.expression" is zero and there is at least one character in the input buffer, the first character is returned and the "then" clause is taken. If "time.expression" is zero and there is no character available in the input buffer, the "in" statement returns immediately and the "else" clause is taken. If "time.expression" is negative, then no timeout is in effect. The system waits indefinitely for the character.

The "then" clause is executed if valid input occurs during the specified timeout. The "else" clause is taken in no input occurs during the allocated time.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

A "length.expresison" of 0 (zero) is allowed in the "input" statement. This syntax is more consistant with other implementations of D3. See example 3.

Syntax

IN variable {FOR time.expression {THEN | ELSE statement.block}

Example

```
loop
 in key
until key = 13 do repeat
This program loops until a carriage return (char(13)) is pressed.

maxtime = 300 ;* 30 seconds
in key for maxtime else
 crt 'timeout occured'
end
In this example, the system waits for "key" for 30 seconds. If no
keystroke is provided within 30 seconds, 'timeout occured' is
displayed and the program stops.

input x,0
x = seq(x)
```

```
These two lines perform exactly the same as:
in x
```

INCLUDE

folds in FlashBASIC code from a separate item to be compiled as part of the object module of the current program or subroutine. The original source program is not altered.

The "include" statement is typically used to include "common" or "equate" statements, format strings, and executable statements from a single source into several programs.

When the FlashBASIC compiler encounters an "include" statement, it opens the file specified by "file.reference", reads the item, and compiles it into the current program.

When the file.reference is omitted, the current file is assumed.

There is no limit to the number of "includes" in a program, but only five levels of nesting are allowed.

The $ is optional.

Syntax

{$}[INCLUDE | INSERT] {file.reference} item-id

Example

```
bp standard.equates
001 equ stx to char(1)
002 equ form.feed to char(12)
003 equ bell to char(7)
004 equ am to char(254), vm to char(253), svm to char(252)
005 dim customer.item(25)
006 open 'customer' to customer.file else stop 201,'customer'
007 equ last.name to customer.item(1)
008 equ first.name to customer.item(2)
009 equ title to customer.item(3)
bp prog
001 include standard.equates
002 read item1 from customer.file,"01234" else print bell
```

INDEX

searches through a given "string.expression" for the occurrence of the character or substring specified by "substring.expression" and if found, returns the numeric position at which the nth occurence of the substring begins. Null strings are skipped.

The "num.expression" defines a particular occurrence to find, (nth occurrence).

If the nth occurence of the specified substring is not found, "0" (zero) is returned.

Syntax

INDEX(string.expression, substring.expression, num.expression)

Example

```
d.position = index("abcdefg","d",1)
This returns the value, "4", to the variable "test", since the first
occurrence of "d" is found in the fourth position of the string.
on index("abcdefg",response,1) goto 10,20,30,40,50,60,70
This example exploits the fact that "index" returns a number
indicating the position of the search character. For instance, if
"response" is the letter "d", then it will goto statement label 40.
loop
 input ans,1
 until ans#'' and index('abc',ans,1) do repeat
```

```
If "ans" is not null and is either an a, b, c, ab, or bc, the loop is
terminated.  Notice that the null condition must be tested separately
since a null value for "ans" results in a "1" from the "index"
function.
```

INMAT

returns information about arrays.

When used without any arguments, the inmat() function returns the number elements filled in the last matparse, matread, or dynamic to static array conversion. Note that inmat() is only valid immediately after these operations. If the number of elements overflowed, then inmat() returns -1.

When used with an array variable, the inmat function returns the dimensions of that array. If the array has one dimension, then the result is returned as a numeric value. If the array has two dimensions, then the result is returned as a dynamic array of the format "{maximum rows}]{maximum columns}".

Syntax

INMAT()

INMAT(array.variable)

Example

```
dimension x(10)
print inmat(x)
x = "a"
print inmat()
When run, the program prints:
10
1
```

INPUT

temporarily suspends execution of the program until a response is provided from the keyboard and assigns that response to a specified variable.

The "input @" prompts for the input of data from the terminal, at the coordinates specified in the "col.exp" ("column expression") and "row.exp" ("row expression") parameters. Both expressions must evaluate to valid screen addresses. See the "@" function.

The input received is assigned to the specified variable after performing the optional "mask" function. The mask can contain any valid mask described under "masking (BASIC: General)" except for masks beginning with a "c". If the variable already has a value, and the "@" function has been specified, the current value of the variable is displayed as the default at the specified cursor address. A <return> accepts the default.

In the "regular" form of "input", the ":" suppresses the carriage return/line feed after input has been completed.

A "val.expression" may be specified to set a default value for the input field. This value is displayed within the input field and the user may edit this value as need be. If no editing is performed, then the default value is accepted as if the user had typed that value at the keyboard.

The "length.expression" indicates the number of characters of input to accept. When the number of characters have been entered, the input instruction terminates exactly as though the operator had pressed <return>.

A "length.expression" of 0 (zero) is allowed. This suppresses the automatic carriage return after single-character input (i.e. holds the cursor position).

A "fill.expression" may be specified to perform a background fill within the input field. The first character of this expression indicates the prompt background fill character. This character is used to help the user see the length of a prompt. The second character, if present, indicates the overstrike character used to fill in any blank spots when the input is accepted. After input is accepted, the cursor is placed at the end of the entered data unless a third fill character is specified in which case the cursor is moved to the end of the input field. The "fill.expression" is ignored if there is no "length.expression".

If the input data value is incompatible with the optionally-specified "mask" expression, the statement re-prompts for input. Mask functions may be any combination of conversions, pattern-matching operators, or text justification. See "masking".

The "_" (underscore) outputs a "beep" to the terminal for each character typed that exceeds the specified maximum length. Echoing of characters to the screen is suppressed for the extra characters and a <return> is required for acceptance. Only the number of characters less than or equal to the "length expression" are accepted. (see warnings).

If "mask" is to be used, the "@" function must also be specified. The mask is used to verify and reformat the actual entry of the data. Any valid format string may be specified. The input is verified against the mask and, if acceptable, is assigned to the variable. Data is input and verified according to the mask, then stripped of its output characteristics and stored in internal format.

For example, if the mask contains a decimal digit specification and/or a scaling factor, then numeric checking is performed; or if the mask contains a length specification, then length checking is performed. See "Format Strings".

When an "input" statement is executed, a prompt character is displayed, followed by the cursor. If the "@" function is used, the prompt is displayed preceding the location specified by "@". Unless the prompt character has been changed with the "prompt" statement, the default prompt character is "?" (question mark).

The "time.expression" is the maximum time that the system waits for input before returning to the FlashBASIC program until any of the following events occur:

1) When a valid termination occurs. The "then" clause is taken.

2) When no carriage return is entered, even though input continues, and no valid termination occurs within the specified time. The "variable" contains all characters entered until the time expired. If no characters are in the type-ahead buffer, "variable" is null. Upon timeout, the "else" clause is taken.

3) When not enough characters are entered to satisfy the "length" requirement within the allocated time, the "else" clause is taken.

The "time.expression" is represented in tenths of a second, as a positive integer from 1 to 32767. Depending on the implementation, there may be significant restrictions on the admissible values for "time.expression". See warnings.

If the "time.expression" is zero and there are characters in the input buffer, these characters are returned immediately when a valid termination criterion is met and the "then" clause is taken.

If "time.expression" is zero and there is no character waiting in the input buffer, the "input" statement returns immediately and the "else" clause is taken.

If "time.expression" is negative, no timeout is in effect.

The "then" clause is taken if valid input is completed within the allocated time. The "else" clause is taken if there is no input, insufficient input, or a non-carriage-return-terminated input occurs in the allocated time.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

INPUT variable{:}

INPUT @(col.exp,{row.exp}){:} variable {,length.expression}{mask} {:}

INPUT variable {=val.expression} {, length.expression} {, fill.expression} {:} {_} {FOR time.expression {THEN | ELSE statement.block}}

Example

```
print 'are you sure? ':
input answer
if answer[1,1]='y' then goto 100
print 'are you sure? ':
input answer,1:
"answer" is the first character entered from the keyboard. No
<return> is required.
input x,0
x = seq(x)
These two lines perform exactly the same as:
in x
 input x for 50 else stop
Wait 5 seconds for input. If there is no input, terminated by a
carriage return, the program stops.
input x for 0 then print x else x = 0
The time out value equal to 0 means read characters without waiting.
If there is any, the 'then' clause is taken, otherwise the 'else'
clause is taken.
print "Enter Cost ":
input @(5,5):cost "r2$"
 This example uses masking, the above mask "r2$" will right justify,
use 2 decimal places and put a '$' on the left of the forced numeric
entry.
 print @(5,5):"Name: ":
input name.new = item<3>, 20 ,"_ " :_
item<3> = name.new
Displays a prompt for a name on a formatted screen. The data to be
edited appears as though the user already typed it in a field of 20
underscores (the first character in fill.expression).  As the user
edits the data (when backspace or backword is used), underscores
replace any erased data. When done, the user must press <CR> or
<LF>.  The data is then repainted, the underscores dissappear
and are replaced with spaces (the second character of
fill.expression).
```

INPUTCLEAR

clears the keyboard input buffer.

INPUTCTRL

toggles ability to enter control characters on succeeding BASIC "input" statements.

Note that this setting affects only BASIC "input" statements performed at the current level.

The default value for this setting is loaded from the TCL default which is set to "on" for a virgin system. The TCL default may be changed with the "control-chars" TCL command.

Syntax

INPUTCTRL [OFF | ON | expression]

INPUTERR

displays a message on the "status" (bottom) line of the terminal.

If "inputerr" is used in a subroutine call from UP, the item cannot be filed if an error exists. In this case, control returns to the UP input screen.

Syntax

INPUTERR string.expression

Example

```
This program has a single valid entry, "e". If any other string is
entered, the string "Entry error" displays on the error message line.
string=''
prompt ''
loop
  crt @(0,10):
  input string,5
until string='e' do
  inputerr 'Entry error'
repeat
```

INPUTNULL

used with the "input @" statement to define the character used to indicate a "null" input on subsequent "input" statements.

The "inputnull" must precede the affected "input @" statement. This statement only works with single character inputs and only one "inputnull" statement is active at any one time. If there is an "inputtrap" active for the same character, the "inputnull" has precedence.

Syntax

INPUTNULL 'character'

Example

```
The "input@" statement requires a numeric value for "string". An "x",
although not numeric, is accepted as a "null" input.
string=''
inputnull 'x'
input @(10,10):string,50 "mr%3"
if string='' then stop
Even though the "inputtrap" reads "x" as an active value, the
"inputnull" overrides. "x" will not terminate this program.
string=''
inputtrap 'x?!' gosub all.done,help.me,whats.da.haps
loop
  inputnull 'x'
  input @(3,5):string,20 "mr#4"
while string='' do repeat
```

INPUTPARITY

enables or disables the extended character set (xcs) for the current process.

When xcs is on, the high order bit is not stripped from input that is transmitted to the system. This allows characters in the range 128 through 255 to be used as unique characters.

When xcs is off, the high order bit is stripped from input. Characters from 128 through 255 are not available.

The FlashBASIC "inputparity" statement is equivalent to the TCL command "xcs."

Syntax

INPUTPARITY {[OFF | ON | expression]}

INPUTTRAP

using the string.expression, it sets up an automatic "computed gosub" based on the next "input @" statements data. The position of the character in the string.expression corresponds to the position of the statement.label in the list. This acts as a "trap" which causes a "gosub" on all subsequent "input @" statements.

Upon "return" from a "gosub" specification, execution continues at the statement following the "inputtrap" command, not the "input @" statement.

"inputtrap" must precede its associated "input @" statement. Only one inputtrap statement can be active at any time.

"inputtrap off" cancels any previous "inputtrap" statement.

Syntax

INPUTTRAP string.expression GOSUB statement.label{ ,statement.label...}

INPUTTRAP OFF

Example

```
inputtrap "abcd" gosub 10,20,30,40
input @(5,5):x
stop
10 print "You entered a"
return
20 print "You entered b"
return
30 print "You entered c"
return
40 print "You entered d"
return
```

INPUTTRAP ... GOTO

using the string.expression, it sets up an automatic "computed goto" based on the next input @ statement's data. The position of each character in the string.expression corresponds to the position of the statement.label in the list. This acts as a "trap" which causes a "goto" on all subsequent "input @" statements and branches to a label in the current program.

"inputtrap" must precede its associated "input @" statement. Only one inputtrap statement can be active at any time.

If data is entered that does not correspond to the inputtrap string.expression then no "trap" occurs and execution continues with the statement after the "input @".

"inputtrap off" cancels any previous "inputtrap" statement.

Syntax

INPUTTRAP string.expression GOTO statement.label{ ,statement.label{,...}}

INPUTTRAP OFF

Example

```
inputtrap "abcd" goto 10,20,30,40
input @(5,5):x
stop
10 print "You entered a"
stop
20 print "You entered b"
stop
30 print "You entered c"
stop
40 print "You entered d"
stop
```

INS

provides an alternate to the "insert" function, for inserting a string expression into a dynamic array.

Syntax

INS string.expression BEFORE dynamic.array< ac.expression {, vc.expression {, sc.expression}} >

Example

```
x<1> = "a"
x<2> = "c"
ins "b" before x<2>
print x
a^b^c
```

INSERT

Inserts object module within object module.

INSERT

inserts the element referenced by "string.expression" into a specific attribute, value, or subvalue location in "dynamic.array.expression".

"-1" may be specified as the "ac.expression", "vc.expression", or "sc.expression". Position "-1" inserts the expression as the "last" element in the respective location.

"0" (zero) may be specified as the "ac.expression". Position "0" inserts the expression as the "first" attribute. A "0" as the "vc.expression", or the "sc.expression" is ignored.

Unlike the "replace" function, which changes the contents of a dynamic array element without changing the logical position of it or any other element, the "insert" function inserts the results of "string.expression" at the specified location and all following information at that dynamic array level (attribute, value, or subvalue) is shifted one position higher in the array. If an "ac.expression" of 1 is specified, the expression is inserted before attribute 1, shifting attribute 1 to attribute 2.

The "insert", "replace" and "delete" functions force the entire dynamic array (string) to be rebuilt. See "dynamic array".

Syntax

INSERT(dynamic.array.expression, ac.expression; string.expression)

INSERT(dynamic.array.expression, ac.expression, vc.expression; string.expression)

INSERT(dynamic.array.expression, ac.expression, vc.expression, sc.expression, string.expression)

Example

```
customer.item = insert(customer.item,1,1,0,name)
-or-
customer.item = insert(customer.item,1,1;name)
Both of these examples are axactly the same. They insert the current
value of "name" in attribute 1 (one) of the array, "customer.item".
Every other value in attribute 1 "shifts right" by one position.
(Value 1 becomes value 2, and so on.) The values of attribute one are
shifted, but the other attributes in the item maintain their relative
position.
```

INT

returns the numeric integer equivalent from a given expression.

Syntax

INT(num.expression)

Example

```
print int(5.1)
5
 This simple example truncates the decimal and returns the integer
portion of the number.
x = 10.25
y = int(x*10)/10
print y
10.2
 This is an example of how you can use the "int" function to keep one
decimal place and truncate the rest.
```

KEY

provides the ability to sequentially "cruise" on any defined b-tree index for an item-id.

The "root" statement must precede the "key" statement.

The "operator" indicates the type of index search to use, and may be one of the following choices:

c Compares, left-to-right, against the "index.key" and returns the first item-id whose index matches into "item.id". If no match is found, the next sequentially higher index and first associated "item.id" are returned.

l Returns the last partial key. If a non-null "item.id" is passed, then this operation is identical to the "p" operator. However, if a null "item.id" is passed, then this function returns the highest valued key in the index which still matches the passed key. This feature is only available in releases 6.1.0 and greater.

n Returns the next "index.key" and "item.id". If more than one "item.id" exists for an "index.key", this returns the next "item.id". After all "item.ids" for an "index.key" have been returned, it returns the next "index.key" and "item.id". If the "item.id" is passed as null, then this returns the first "item.id" which matches the passed "index.key".

p Returns the previous "index.key" and "item.id". If more than one "item.id" exists for an index.key, this returns the previous "item.id". After the first "item.id" for an "index.key" has been returned, or if "item.id" is null, it returns the previous "index.key" and the last "item.id".

r Returns the "index.key" and "item.id" only on an exact match with the "index.key".

v Verifies the index. Locates the given "index.key" and "item.id" and verifies that an exact match can be found.

x Works like the "n" option, but returns ALL item-id's matching the given key as a dynamic array. If the passed item-id is null, then the data for the passed key is returned. If the passed item-id is non-null, then the list for the NEXT key is returned. The vc.expression, if present, returns the number of elements found with an original value position not equal to 1. If vc.expression is non-zero, then the item-id list may contain duplicate item-id's.

root.variable is the b-tree root fid associated with the target index. It must be initialized by the "root" statement, and must be defined prior to using the "key" statement.

index.key specifies a mandatory variable containing the search string. The variable must be defined before using the "key" statement. The actual string found is returned into the variable when using the "c", "n" or "p" operators. The "r" and "v" operators expect an exact match.

item.id is the variable that is assigned the item-id of the item that contains the key. The "item.id" need not be preassigned when using the "p" or "n" operators, but must be preassigned when using the "v" operator.

vc.expression is the value number of the key attribute containing the "index.key", returned to the "vc.expression" by the "key" statement. This only works with the 'c', 'n', 'p' and 'r' operators.

The "then" clause is executed if the item-id is found or if the "index.key" verifies with the "v" operator.

The "else" clause is executed if the item-id is not found or if the "index.key" does not verify with the "v" operator.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

KEY('operator', root.variable , index.key, item.id {, vc.expression}) {THEN | ELSE statement.block}

Example

```
employee.index ='a5'; * phone number attribute
execute "create-index employee ":employee.index
root 'employee',employee.index to e.root else
  print "the index ":employee.index:" not found"
end
print "Enter phone number to match ":
input e.key
key('n',e.root,e.key,item.id) then print item.id
In the "root" statement, "employee" is the file that contains the
pre-defined index. If the index is found, the root fid is assigned to
the variable "e.root".
In the "key" statement, 'n' indicates that the "next" matching string
is to be located. "e.key" is the variable which contains the search
string and "item.id" is the variable to which the actual item-id will
be assigned, if it is found.
```

LE

represents the "less than or equal to" condition.

Syntax

expression LE expression

Example

```
if x le 10 then print "Less than 10 is the result"
```

LEN

returns the length of a string expression.

Syntax

LEN(string.expression)

Example

```
customer = "john"
name.length = len(customer)
print name.length
4
string = "testt"
string = string[1,len(string)-1]
print string
```

```
test
This assignment statement uses the "len" function to delete the last
character of "string".
string = 'abcde'
print len(string)
This outputs a 5.
a = 5
x = len(a * 10)
print x
2
This code gets the length of the result of an arithemtic expression.
The result of the arithmetic expression is converted to an ASCII
string when the "len" function is invoked. The output is 2, since 5
times 10 is 50, and the length of 50 is 2 characters.
```

LET

assigns the value of an expression to a variable. The variable can be a simple variable, an array element, an extraction, substring, or both extraction and substring.

The "let" statement is a holdover from the original Dartmouth BASIC, and is only included for compatibility. It is supported to facilitate program readability.

Syntax

{LET} variable = expression

Example

```
let a = 5
Which is exactly the same as:
a = 5
```

LN

returns the natural logarithm (base e, which is 2.718281828 rounded to "precision") of a given numeric expression.

This function is the inverse of the "exp" function.

If the expression is less than or equal to zero, "ln" returns a zero and prints a runtime error message.

Syntax

LN(num.expression)

LOCATE

searches for the location of a specific "string.expression" and returns the location in "position.variable" of where the string was found or if the string wasn't found, the position that it should be placed.

The elements of "dynamic.array.expression" may be specified as being in ascending or descending ASCII sequence, and sorted with either right or left justification.

Sequence parameters: (In D3, the single quotes around the sequence parameters are not required.)

al Ascending, left-justified.

ar Ascending, right-justified.

dl Descending, left-justified.

dr Descending, right-justified.

If the first character in the "sequence" expression is anything except "a" or "d", or the "l" or "r" is not specified, no sort is performed. If the second character is anything except "r", left justification is assumed. If no "sequence" parameter is specified, the "position.variable" position defaults to the end of the string.expression.

The use of the optional "ac.expression" and "vc.expression" indicates whether the value returned into "position.variable" is a value count or a subvalue count. If both are omitted, the value returned into "position.variable" is an attribute count.

"start.expression" is the first field to search. If not specified, the entire string is searched.

If "ac.expression" and "vc.expression" are both specified, "start.expression" is the first subvalue to search.

If only "ac.expression" is specified, "start.expression" is the first value to search.

If "ac.expression" is not specified, "start.expression" is the first attribute to search.

The "string.expression" must match the element exactly in order for the location to be returned.

To use the "start.expression" while not specifying the "ac.expression" and "vc.expression", a "0" (zero) must be substituted to hold the syntactical position.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

LOCATE(string.expression, dynamic.array.expression {, ac.expression{, vc.expression{, start.expression}}}; position.variable {; sequence.expression}) [THEN | ELSE statement.block]

LOCATE string.expression IN dynamic.array.expression{ <ac.expression{, vc.expression}>} {,start.expression} {BY sequence.expression} SETTING position.variable [THEN | ELSE statement.block]

Example

```
equ vm to char(253)
continents = 'africa':vm:'asia':vm:'south america'
input acontinent
locate(acontinent,continents,1;position;'al') then
  crt acontinent:' is already there'
end else
  continents=insert(continents,1,position;acontinent)
end
crt continents
"continents" is a list of continents in alphabetical order.
"acontinent" is added to the list only if it does not already exist.
The "locate" statement uses the sequence parameter "al", ascending,
left-justified.
If the value of "acontinent" is "europe", it does not exist in
"continents", causing "locate' to take the "else" clause with the
value of "position" set to 3. The "insert" function places "europe"
in front of "south america" leaving "continents" as:
africa]asia]europe]south america
The "]" (bracket) represents a value mark.
```

LOCK

sets one of 64 "execution" locks, in the range 0 to 63. This prevents program "re-entrancy", allowing only one process to run the program at any given time.

If the specified lock is found "unlocked", this statement "locks" the lock and the "then" clause, if present, is executed. If the lock is already locked by another process or by the current process at a different level, it remains locked and the "else" clause, if present, is executed. If the lock is locked by another process or by the current process at a different level and no "else" clause is used, the program waits until the lock is unlocked.

If the "lock.number" expression evaluates to greater than 63, the result is divided by 64, and the "lock.number" is equal to the remainder of the equation.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

LOCK lock.number {THEN | ELSE statement.block}

Example

```
lock 12
This unconditionally sets execution lock 12.
lock 14 else crt "Program is locked" ; stop
This attempts to set lock 14. If it is already locked, the program
advises the operator and stops.
```

LOOP

repetitively executes (loops) until some ending condition is met. The first set of statements, if present, is executed at least once.

The "loop..until" or "loop..while" forms are repetitive loop functions used to repeat a sequence of statements conditionally.

If the "until" clause is used, looping continues as long as the expression following it is false. If the "while" clause is used, looping continues as long as the expression following it is true.

The statements in the "do" clause are executed as long as the loop is executed. If "while" or "until" is specified, the "do" is required.

The "repeat" statement defines the end of the loop.

If neither a "while" nor "until" clause is used, the loop never finishes. The use of "goto" or "exit" is required to exit the loop in this construct.

If the "until" expression is initially true or the "while" expression is initially false, the statements specified in "statements1" are executed at least once and "statements2" are not executed.

Syntax

LOOP {statement1{s}} [UNTIL | WHILE] logical.expression DO {statement2{s}} REPEAT

-or-

LOOP
 {statement1{s}}
[UNTIL | WHILE] logical.expression DO
 {statement2{s}}
REPEAT

-or-

LOOP {statement{s}} REPEAT

Example

```
loop
 crt "enter value to test " :
 input value,1
 until value = "q" do repeat
This loop terminates when "value" evaluates to a "q".
loop
 x = c*2
 while x < 100 do
 c = c + 1
 repeat
This loop terminates when the calculated value of "x" is under 100.
Each time through the loop, "c" is incremented by one.
execute "select customers"
loop
 readnext item.id else exit
 print item.id
 repeat
This processes an active list. When the last item-id is read, the
"exit" statement passes control to the next executable statement
after the "repeat" statement. Notice the select list is the default
active select list. (see readnext).
```

LOOP ... UNTIL

Loops "until" condition is met.

LT

is the symbol used as a logical operator within conditional expressions as an alternate form of the "<" symbol used for representing a "less than" condition.

Syntax

expression LT expression

Example

```
if x lt 0 then print "less than zero"
```

MAT

assigns data to each element of a dimensioned array variable to a specific value in one operation.

Array variables must be defined in a "dim", "common", or "file" statement.

{mat} array.variable1 = {mat} array.variable2

Copies one array into another. The number of elements in "array.variable1" must be less than or equal to the number of elements in "array.variable2". It is possible to assign a two-dimensional array to a single-dimensional array, and vice-versa.

Syntax

MAT array.variable = expression

{MAT} array.variable1 = {MAT} array.variable2

Example

```
mat customer.item = ""
This sets every array element of "customer.item" to null.
mat customer.item.history = mat customer.item
This sets every array entry of "customer.item.history" to its
corresponding entry in "customer.item".
dim a(10,2)
dim b(20)
mat b = 1
b(3) = 2
```

```
mat a = mat b
This example assigns all of the element in the two dimensional array,
"a" into the one dimensional array, "b". The rows are "filled" before
the columns, therefore a(2,1) = 2.
```

MATBUILD

converts a dimensioned array into a dynamic array.

"dynamic.array.variable" is the variable constructed from all or part of a dimensioned array.

"array.variable" references the dimensioned array.

"start" is an integer expression which indicates the optional starting position within the array to begin parsing. If omitted, it defaults to 1 (one). The value of "start" must be numeric.

"end" is an integer expression which indicates the optional ending position within the array to stop parsing. If omitted, it defaults to the array size previously declared with a "dim" statement for the given array. The value of "end" must also be numeric.

"delimiter" is the optional delimiting character within the string to be used to separate data parsed from the array. If omitted, the default delimiter is an attribute mark (char(254)). The "delimiter" may be any character between x'00' and x'fe'.

Syntax

MATBUILD dynamic.array.variable from array.variable{,start {,end}} using delimiter

-or-

dynamic.array.variable = array.variable

Example

```
dim a(10)
for x = 1 to 10
 a(x) = x
next x
matbuild xyz from a,4,9 using char(253)
print xyz
4]5]6]7]8]9
dim a(4)
a(1) = "larry"
a(2) = "moe"
a(3) = "curly"
a(4) = ""
stooges = a
print stooges
larry^moe^curly
Note that because the final element of the dimensioned array is null,
it is not appended to the dynamic array.
```

MATCHES

Alternate to "match".

MATCH

tests a string and determines if it matches a pre-determined pattern of alphabetic, numeric, wild card, or literal characters.

The match operator compares a string value to a predefined pattern and evaluates to 1 (true) or 0 (false).

The "string.expression2" may be a composite of literals and/or match operators, appended to length specifications. During processing, the value must be the exact length of the length specifications parameter.

String operators:

na accept only "n" alphabetic characters.

mn accept only "m" numeric characters.

nx accept "n" wildcards (any character).

literal accepts any literal string enclosed in quotes.

The "n" parameter specifies the length of the match operator string. A length specification of 0 (zero) allows a length of zero or more of the following match operator. When combinations of matchstrings and literals are present, the entire matchstring must be enclosed in double quotes.

The character "n" tests only for decimal digits. "+", "-", and "." are NOT considered numeric.

The "iconv" function, using the pattern match ("p") processing code, provides similar functionality to the "match" or "matches" statement. The difference between the two, however, stems from the fact that "iconv" can perform multiple pattern matches at once, where the "match" or "matches" requires multiple statements.

Syntax

string.expression1 MATCH{ES} string.expression2

Example

```
if answer matches "3n" then print "ok"
This statement takes the "then" path if the answer is 3 characters.
if soc.sec.num matches "3n'-'2n'-'4n" then...
This checks that the value of the variable "soc.sec.num" has the
pattern of 3 numbers, a "-" (dash), 2 numbers, a dash, and 4 numbers.
if not(response matches "0n") then...
The "then" clause is taken if "response" is NOT zero or more numbers.
if response = "0n,0n,0n" then...
This accepts any length of numbers, a comma, any length of numbers, a
comma, and any length of numbers.
```

MATPARSE

converts a dynamic array into a dimensioned array.

"array.variable" references the dimensioned array.

"start" is an integer expression which indicates the optional starting position within the "array.variable" to begin updating. If omitted, it defaults to 1 (one).

"end" is an integer expression which indicates the optional ending position within the "array.variable" to stop updating. If omitted, it defaults to the array size previously declared with a "dim" statement for the given array. In all cases, the last array element will be assigned all the remaining elements in the dynamic array.

"dynamic.array.expression" is the data to be used to update the array.

"delimiter" is the optional delimiting character within the string to be used to separate data parsed into the array. If omitted, the default delimiter is an attribute mark (char(254)). The "delimiter" may be any character between x'00' and x'fe', but must be a string.

"variable" indicates the number of elements of "array" that were assigned a value.

In the form:

array.variable = dynamic.array.expression

Each attribute in "dynamic.array.expression" is assigned to an element in "array.variable".

Syntax

MATPARSE array.variable {,start {,end}} FROM dynamic.array.expression {USING delimiter} {SETTING} variable

array.variable = dynamic.array.expression

Example

```
dim a(10)
rec<1> = "test1"
rec<2> = "test2"
matparse a from rec setting attr.count
print a(1)
test1
a = ""
a<1> = "larry"
a<2> = "moe"
a<3> = "curly"
dim stooges(4)
mat stooges = "unknown"; * initialize
stooges = a;* matparse equivalent
print stooges(3) : " " : stooges(4)
curly
 Notice that stooges(4) was null.
```

MATREAD

reads the specified item from the optionally specified file.variable, or if not specified, the default file.variable, and stores one attribute per element in the array.

This statement may optionally be written "matread" or "read". If "mat" is omitted, a matrix read can be determined by the definition of the variable, "array.variable". If no "dim" statement appears, it is automatically treated as a "dynamic" array.

The "else" clause is taken when the item is not on file. The "then" condition is taken when the item is read successfully.

The "locked" clause occurs before the "then" and/or "else" clause{s} and specifies the statement{s} to execute if the item is locked when the read is attempted. The "locked" clause may be used in conjunction with a "then" or "else" clause, but not both.

The "matreadu" form tests if the item is already locked and locks the item if it is not. The item lock set by "matreadu" prevents the item from being read using the "{mat}readu" statement or updated by other processes while the lock is set.

If the "matreadu" form is specified and no "locked" clause is present, the program pauses (and "beeps" continuously) at the locked item until it is available. If a "locked" clause is specified, the statements in the "locked" clause are executed if the item is already locked by another process or the current process at a different level. "system(30)" contains the port.number which has the item locked. For backwards compatibility, "system(0)", in the "locked" clause, also returns the port.number which has the item locked.

The item lock is only released by a "release", "delete", "write", "writev", or "matwrite" statement. Items can be updated without being unlocked by using the "writeu", "writevu", or "matwriteu" statements.

If a binary item is read, system(0) is set to 10. The variable string consists of the hex fid in attribute 1 and the hex frame in attribute 2.

If the array is defined by a "file" statement, the "file.variable" is not used.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

{MAT}READ{U} array.variable FROM {file.variable,} id.expression {LOCKED statement.block} {THEN | ELSE statement.block}

Example

```
open 'customer' to cust.fv
dim customer.rec(20)
matread customer.rec from cust.fv,item.id
dim a(5)
mat a = "xyz":am":"abc"
matwrite a on "a"
This writes the dimensioned array "a" on the default file, using "a"
as the item-id. Although there are only five elements in the array,
each corresponds to two attributes (separated by attribute marks),
thus, ten total attributes are written.
dim customer.item(20)
matreadu customer.item from cust.fv,item.id locked
 crt item.id:' is locked by port ':system(0)
end then
 crt 'got it'
end else
 crt 'not on file'
end
```

MATREADU

see "matread".

MATWRITE

writes an item into the specified file.variable.

This statement may optionally be written "matwrite" or "write". If "mat" is omitted, a matrix write can be determined by the definition of the variable, "array.variable". If no "dim" statement appears, it is automatically treated as a "dynamic" array.

If the file.variable parameter is omitted, the default file.variable is used.

The "matwriteu" statement is identical to the "matwrite" statement, except the item remains locked. The "u" form keeps the item locked if it was previously locked by a "readu", "readvu", or "matreadu" statement within the same program.

The array is truncated to the last non-null array element. All trailing null elements are deleted. If the array is defined by a "file" statement, the "file.variable" is not used.

The "writex", "matwritex" and "matwritexu" statements all have the property of waiting until the actual disk update takes place before continuing execution of the program. They are used for "critical" write-through, such as error-logging.

Syntax

{MAT}WRITE{X}{U} array.variable ON {file.variable,} id.expression

Example

```
dim customer.item(30)
matread customer.item from customer.file,item.id else ..
customer.item(1)=newname
customer.item(30)=date()
matwrite customer.item on customer.file,item.id
```

MATWRITEU

see "matwrite".

MAXIMUM

returns the maximum of a list of numbers delimited by attribute, value, or subvalue marks. If different orders of marks are present, the "maximum" function will return the maximum of all fields regardless of delimiter.

Syntax

MAXIMUM(string.expression)

Example

```
equ am to char(254)
print maximum(3:am:3.2)
The result, "3.2", will be printed to the terminal.
```

MINIMUM

returns the minimum of a list of numbers delimited by attribute, value, or subvalue marks. If different orders of marks are present, the "minimum" function will return the minimum of all fields regardless of delimiter.

Syntax

MINIMUM(string.expression)

Example

```
equ am to char(254)
print minimum(3:am:3.2)
The result, "3", will be printed to the terminal.
```

MOD

returns just the remainder portion of the result of dividing one number by another.

There is no limitation on the numeric value of dividend or divisor. The "rem" and "mod" functions are identical.

Syntax

MOD(dividend, divisor)

Example

```
for i = 1 to lines
  print rec<i>
  if not(mod(i,60)) then print char(12):
next i
In this example, "mod" is used to find the end of a page.  Every 60
lines, a form feed, char(12), is printed.
```

NE

represents a "not equal to" condition within relational expressions.

Syntax

expression NE expression

Example

```
if ans ne "y" then print "Not equal to Yes" else print "Yes"
```

NEXT

occurs at the end of a "for..next" construct and causes the iteration counter to increment and branches to the corresponding "for" statement to decide whether to terminate.

The "for..next" construct may also be exited using the "exit" statement.

The "step" statement allows the increments of the loop to be changed. For example, "step 2" increments the variable by 2 when the "next" statement is encountered.

"step" can be a negative number, decrementing the variable at the "next" statement. A "step" of zero does not execute the "for..next" loop.

Syntax

FOR variable = expression TO expression {STEP expression}

statement{s}

NEXT variable

Example

```
for i = 1 to 10
 crt i:" ":
next i
print
1 2 3 4 5 6 7 8 9 10

numbers = ''
for i = 1 to 2
 for j = 1 to 5
   numbers<i,j> = j
 next j
next i
print numbers
1]2]3]4]5^1]2]3]4]5
for j = 10 to 1 step -1
 print j:" ":
next j
print
10 9 8 7 6 5 4 3 2 1
```

NOT

negates the effect of the normal outcome of "true" and "false" generated by a conditional expresssion.

Returns the logical inverse of the expression. If the expression evaluates to zero (0), one (1) is returned. If the expression evaluates to any non-zero value, zero is returned.

Syntax

NOT(logical.expression)

Example

```
if not(num(response)) then print "not numeric"
If "response" is not numeric, this prints "not numeric".
```

NULL

performs no operation, other than to provide an instruction where one is required.

This is usually used for program readability.

Example

```
if answer = "yes" then print "Are you sure?" else null
```

NUM

evaluates an expression and returns 1 (true) if it is a decimal numeric. Otherwise, it returns 0 (zero), if the expression contains any non-numeric characters.

The empty, or "null" string ('') is considered to be a numeric string.

The following strings are considered to be numeric, and return a "1" (true):

"." (period), "-" (minus), "+" (plus), "-." (minus/period), "+." (plus/period).

Syntax

NUM(expression)

Example

```
loop
  input number
until num(number) do repeat
This loop terminates when a numeric response is entered for the
variable "number".
```

OCCURS

searches a string for attributes or values that occur consecutively.

The string that is searched is specified by "string.expression". "num.expression" is a numeric expression that specifies the number of occurrences. This function returns all substrings that occur at least the number of times specified by "num.exprression", delimited by the same mark as in the string.

Syntax

OCCURS(string.expression, num.expression)

Example

```
In this example, the "occurs" function evaluates the following
string:
'22]11]11]11]22]11]11]22]11]11'
(bracket (]) represents a value mark)
for n = 1 to 5
    substr=occurs(string,n)
    print n 'r%2' : ' ': substr
    print dcount(substr,vm):' string(s) occuring ':n:' time(s)'
next n
This code outputs the following:
01 22]11]22]11]22]11
6 string(s) occuring 1 time(s)
02 11]11]11
3 string(s) occuring 2 time(s)
03 11
1 string(s) occuring 3 time(s)
04
0 string(s) occuring 4 time(s)
05
0 string(s) occuring 5 times(s)
The output for line 01 is each time "string" is found at least once
in a series. On line 02, the substring 11 is found to occur in a
series of 2 at least twice in the major string.  The string 11 is
also found in a series of 3 one time.  No 4 or 5 repetitive series
are found.
```

OCONV

The "oconv" function converts a value from its internal format to its external equivalent, according to the processing code being applied.

Syntax

OCONV(string.expression, conv.expression)

Example

```
print oconv(check.amount,"mr2")
This example converts "check.amount" to external format, using the
"mr2" conversion. (See the "m (mask)" processing code.)
print oconv(check.date,"d")
This example converts "check.date" to external format, using the "d"
conversion. (See the "d (date)" processing code.)
```

ON ... GOSUB

The "on .. gosub" statement transfers control to a local subroutine designated by a specific statement label according to the positional value of the "index.expression". The syntax also allows being specified as "on ... go sub ..." (it allows a space between "go" and "sub").

The "index.expression" produces a number that corresponds to the position of the corresponding statement label within a list of statement labels separated by commas (,).

The "index.expression" is truncated to an integer and the program executes the subroutine identified by the label number in the statement that corresponds to the truncated expression.

For example, if the expression evaluates to 1, the first subroutine is executed. If the "index.expression" evaluates to a number less than one, or to a number greater than the number of statement labels, no action is taken. Execution continues at the first executable statement following the "on ... gosub".

The local subroutine must be terminated with a "return" statement. At the end of the subroutine, the program returns to the statement following the "on...gosub" statement.

For ease of reading, the "on..gosub" statement.label list can be spread over multiple lines if each continued line is terminated with a comma (,).

Syntax

ON index.expression GOSUB statement.label{, statement.label...}

-or-

ON index.expression GOSUB statement.label,...,

statement.label,

statement.label

Example

```
loop
  print '1) doit  2) printit  3) deleteit'
  input response
until num(response) do repeat
if response >=1 and response <= 3 then
    on response gosub 8000,9000,9500
end
If "response" is 1, this branches to subroutine 8000. If it is 2, it
branches to subroutine 9000. If it is 3, it branches to subroutine
9500.
print 'a) doit  b) printit  c) deleteit'
input response,1
on index('abc',response,1) gosub doit,printit,deleteit
In this example, the result of the "index" function is used as the
"index.expression" to determine which subroutine is called.  If the
response is "a", then subroutine "doit" is called, and so on.
on response gosub 10,20,30,40,50,60
,70,80,90,100,110,120,130
When the value of the "index.expression" has a large number of
```

```
possible branches, readability is improved by using the multiple line
syntax.
```

ON ... GOTO

transfers control to the line within the FlashBASIC program which begins with the specified statement label, according to the positional value of the index expression referenced by "index.expression". The syntax also allows being specified as "on ... go to ..." (it allows a space between "go" and "to").

The "on ... goto" is identical to the "on ... gosub", except there is no "return" required.

Syntax

ON index.expression GOTO statement.label{, statement.label...}

-or-

ON index.expression GOTO statement.label,...,

statement.label,

statement.label

Example

```
on response goto 100,200,300,400,500
This is an indexed branch in the single-line format.
on response goto 100,200,300,
          400,500,600,
          700,800,900
This is an indexed branch in multi-line format.
```

OPEN

opens a specified filename and associates the file with the optional file.variable.

If "dict" is specified (or implied as an expression), the dictionary level of the specified filename is opened and assigned to the file.variable. The optional 'dict' may be expressed as part of the file name as 'dict file.name'.

If no file.variable is specified, the file is opened and assigned to the "default" file.variable.

If the file is opened successfully, the "then" clause, if present, is executed. If the file is not opened successfully, the "else" clause, if present, is executed. If the file can not be opened and no "then/else" clause is specified, the program terminates with error message 201.

See the "then/else construct" for an explanation on the use of "then" and "else" clause.

Syntax

open {"{dict}",}file.reference {to file.variable} {then | else statement.block}

Example

```
open 'customer' to customer.file else stop 201,'customer file cannot
be opened'
Opens the data area of the "customer" file to the file.variable
"customer.file". If the "customer" file does not exist, it stops and
prints error message 201.
open 'dict','invoice' to invoice.dict then
    print 'ok'
end else
    stop 201,'dict invoice cannot be opened'
end
Opens the dictionary of the "invoice" file. If successful, it prints
"ok". If not, it stops and prints error message 201, passing the
string "dict invoice". This results in the display:
```

```
"[201] 'dict invoices' is not a file name".
open "invoice.control"
readv next.invoice.id from "invoice.counter",1
This form, available only in D3 implementations, demonstrates how the
"to", "then" and "else" are all optional in an "open" statement. Note
the file.variable was left off of the read command.
```

operators

Operators are reserved characters (and combinations of adjacent characters): * ! & = # - + / ^ < > \ [ ] ( ) and may be referenced individually by their respective symbol.

Expressions are formed by combining operators with other expressions.

When an expression is encountered as part of a FlashBASIC program statement, it is evaluated by performing the operations specified by each of the operators on the adjacent operands.

Each operator has a precedence rating and, in any given expression, the highest precedence operation is performed first. (0 is the highest precedence and 8 is the lowest). If there are two or more operators with the same precedence or if an operator appears more than once, the leftmost operation is performed first.

OR

"Logical or" operator.

OUT

The "out" statement outputs a single ASCII character derived from a numeric expression in the range 0 to 255, which indicates its corresponding position in the ASCII table.

The expression must evaluate to a decimal number. It is evaluated and adjusted (mod(expression,256)) to a value between 0 and 255. The corresponding ASCII character is printed on the terminal screen.

Syntax

OUT num.expression

Example

```
out 7
This prints char(7) and rings the terminal bell.
```

PAGE

terminates the current page of output, prints the optional footing, positions to top of form, and prints the optional heading.

The optional expression may be used to change the current value of the page number.

Syntax

PAGE {num.expression}

Example

```
heading "this is a heading"
lines = dcount(item,char(254))
for i = 1 to lines
  l=item<i>
  if l[1,3]='.bp' then page else print l
next i
This example forces a new page if the attribute to be printed starts
with the string ".bp".
```

PERFORM

See: execute

PRECISION

defines the number of fractional decimal places that a numerical value can hold.

The default precision is four. "precision" may be set in the range of 0 to 9. Numbers with more decimal places than the precision are truncated.

Only one "precision" statement is allowed in a program and the "precision" statement must precede the use of any numeric data. Programs calling subroutines or entering other programs must have the same precision. (See "call" and "enter"). If the precisions do not match, the program aborts into the debugger. This restriction also applies to main programs which share data in named common space. However, violations in this case are not reported, and values of named common variables will be incorrect.

For certain "arithmetic functions", higher "precision" limits the magnitude of the values returned.

In D3, most functions have no limitation on their numeric range.

"precision" in the range of 1 to 9 handles numbers as 48-bit scaled binary numbers.

If the result of an "arithmetic.expression" which divides (/), multiplies (*), takes a remainder (\), or produces an exponent (^) exceeds the maximum magnitude of the 48-bit representation, the system creates an internal variable type which uses a precision of 18 digits to the right of the decimal point, and unlimited to the left. This feature assures maximum accuracy.

Operations which exceed the limits of the current "precision", round the results rather than truncate; (99999/100000 with "precision 4" results in a 1).

A "precision" of 0 forces 48-bit integer arithmetic for all operations. (99999/100000 results in a zero).

Syntax

PRECISION num.constant

Example

```
precision 2
This statement changes this program's arithmetic precision to 2. All
subroutines called from this program must have the same "precision"
statement.
precision 6
print 9999999/10000000
The result of the statement is rounded to 1 since the limits of
precision have been exceeded.  The expression (9999999/10000000) when
converted for print is rounded when it is translated to a string.
```

PRINT

directs output of an expression to the current output device.

If nothing follows the print statement, a blank line is output.

The "print.list", if present, consists of any number of expressions, including "@" functions, separated by commas or colons.

If a comma follows a print expression, the string generated is formatted right justified in a field of 15 blanks and aligned at tab positions that are pre-set at every 18 character positions.

A colon at the end of the print.list inhibits the output of a return and scroll following output of the last printed line.

Syntax

PRINT {expression{,expression..}{:}

Example

```
print 'hello'
This statement outputs the literal string "hello". The quotes delimit
the literal string and are not printed.
message = 'hello'
print message
This statement outputs the contents of the variable "message" to the
current active output device.
print message:' ':company.name
This statement outputs the results of a string expression made up of
the variables "message" and "company.name" and the literal string
space (" ").
print @(10,10):'the amount is ' : (unit.price * qty) 'r#12'
This statement positions the cursor at column 10, row 10 of the
terminal display, displays the literal string "the amount is ", and
outputs the result of the arithmetic expression "unit.price * qty"
right justified in a field of 12 characters.
```

PRINT ON

directs output to one of a number of open print files.

On D3, 100 print files may be opened at one time. The range of the "print.file.number", however, is between 0 and 32767.

The "print.file.number" has no connection to Spooler print file numbers. This number is logical and local to the current program and is used to group output.

Each logical "print.file.number" is assigned the next available spool job number, so it is possible to have the statement "print on 1 answer" output to job #3 and "print on 2 string" output to job #30.

Syntax

PRINT ON print.file.number print.expression

Example

```
print on 0 oconv(chk.dt,"d2/") "l#9":amt "r2*20"
```

PRINTCHAR

prints the first character of the string expression to the screen or printer.

Syntax

PRINTCHAR string.expression

PRINTER

controls the output from subsequent "print", "heading", "footing" or "page" statements.

"printer on" directs output to the system printer, via the Spooler.

"printer off" directs output from subsequent "print" statements to the terminal.

"printer close" indicates that the current (printer) output is completed. This closes all open Spooler entries generated by the current process, and releases control of the print job(s) to the

Spooler. "printer close" is implicit when the FlashBASIC program stops and returns to TCL, or the calling PROC or executing program.

"num.expression" evaluates as follows:

-1 "printer close"

0 "printer off"

>=1 "printer on"

See the "crt" statement for directing output to the terminal while "printer on" is in effect.

Syntax

PRINTER [OFF | ON | CLOSE | num.expression]

Example

```
print 'press <return> to begin printing'
printer on
print 'this is the first line'
The first string is displayed, print output is turned on, and the
second string is Sent to the spooler.
crt 'print? ':;input answer
printer (answer='y')
The printer is turned on if the variable "answer" is the letter "y".
printer on
print rec<5>
if rec<6>='' then
    printer off
    print 'error in attribute 6'
    print 'press <return> to continue ':
    input continue
    printer on
end
This section of program turns off the printer to display an error
message and prompt for operator intervension. When the printer is
turned back on, output is appended to the currently open spooler
file.
10 *
print 'do you want to print? ':
input ans
if ans # 'y' then stop
printer on
print 'the amount is ':amount
print ...
....
printer close
printer off
goto 10
In this example, a single section of a larger program sends output to
the Spooler, closes the job when finished, and continues execution of
the program.
```

PRINTER CLOSE

See: printer

PRINTER OFF

Directs "print" statement output to crt.

PRINTER ON

Directs "print" statement output to Spooler.

PROCREAD

reads the calling PROCs primary input buffer and assigns its contents to a specified variable.

When successful, the variable is treated as a string of characters delimited by spaces. The "field" function may be used to parse the variable. The "dcount" function may be used to determine the number of entries (number of spaces + 1) in the buffer variable.

The "else" condition is taken when the program has not been executed from a PROC.

Syntax

PROCREAD variable [THEN | ELSE statement.block]

Example

```
pq
ri
s1
oenter beginning date +
ip
s2
oenter ending date +
ip
hrun bp this.program
p
This is a sample PROC which prompts the operator for two dates. This
information can now be passed to the FlashBASIC program
"this.program".
this.program
001 procread buffer else
002   print 'this.program must be run from a PROC'
003   stop
004 end
005 start.date = field(buffer,' ',1)
006 end.date   = field(buffer,' ',2)
```

PROCWRITE

writes a string expression variable to the calling PROCs primary input buffer.

In order for PROC to be able to deal with the information in the variable, each space in the variable defines a new parameter in the PROCs input buffer.

Syntax

PROCWRITE variable

Example

```
pq
hrun bp this.program
p
if a1 = abort x
This PROC checks the primary input buffer to see if it has been
altered by the program "this.program".
read 'testrec' from test.file,test.id else
procwrite 'abort'
stop
end
This program writes an abort message to the controlling PROCs primary
input buffer.
```

PROGRAM

optionally used on the first line of a program to indicate that this is a program.

It is not needed, and is ignored by the compiler. It is supported strictly for compatibility with other implementations.

Syntax

PROGRAM name

Example

```
001 program tester
002 * this program tests things
....
```

PROMPT

indicates the single character to display during subsequent "input" statements which prompt for input from the keyboard.

The default prompt character is "?" (question mark). The prompt character remains defined until another "prompt" statement is executed.

Syntax

PROMPT character.expression

Example

```
prompt ""
This statement sets the prompt character to null so that no character
is output when an "input" statement is executed. This is useful when
working with serial devices.
```

PWR

raises a value contained in an expression to the power of the value of a second expression.

A "power.expression" of 0 (zero) always returns a "1" (one). A "num.expression" of 0 (zero) returns zero.

Syntax

PWR(num.expression, power.expression)

Example

```
print pwr(3,2)
This statement prints a 9, the result of 3 taken to the power of 2.
print 3^2
In this statement "3^2" is identical to pwr(3,2).
print pwr(xyz,.5)
This prints the square root of the value stored in the variable
"xyz". This is equivalent to:
print sqrt(xyz)
```

quotes

marks the beginning or ending of a literal string in FlashBASIC.

Generally, it does not matter which types of quotes are used on literals (strings). Some instructions, such as "heading" and "footing", impose certain restrictions on their use.

If a literal string is to contain a single-quote, it must be enclosed within double-quotes (") or backslashes (\).

Syntax

[\ | ' | "] string [\ | ' | "]

Example

```
if answer = 'quit' then stop
In this example, single quotes are used to determine if the contents
of "ANSWER" contains the literal string, "QUIT".
heading "'lc'Pick Systems'l'"
This example illustrates the use of two types of quotes, which happen
to be enforced by the "heading" statement. see "heading".
execute \list entity heading "'lc'Entity'l'"\
AQL sentences enforce the rule that the "heading" string must be
enclosed in double quotes. "heading" options must be enclosed in
single-quotes ('). The whole string is passed as a literal by
enclosing it in backslashes (\).
```

READ

reads an item as a dynamic array and stores the item as a string. Each attribute is automatically an element of the dynamic array variable and may be referenced with the appropriate intrinsic functions.

If the file.variable parameter is not specified, the default file.variable is used.

The "else" clause is taken when the item is not on file.

The "then" clause is taken when the item is read successfully.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

The "readu" statement is identical to the "read" statement, except that the item is "locked", preventing access to that item by any other process.

The "locked" statements are executed if the item exists and is already locked by another process or by the current process at a different level. "system(0)", in the "locked" clause, returns the port number which has the item locked. At the termination of the program, "readu" unlocks the item.

If a binary item is read, "system(0)" is set to 10. The variable string consists of the hex fid in attribute 1 and the hex frame in attribute 2.

This statement may optionally be written "matread" or "read". If "mat" is omitted, a matrix read can be determined by the definition of the variable, "array.variable". If no "dim" statement appears, it is automatically treated as a "dynamic" array.

Syntax

READ{} dynamic.array FROM {file.variable,} id.expression {LOCKED statement{s}} {THEN | ELSE statement.block}

Example

```
open 'customer' to customer.file else stop 201, 'customer'
item = ''
crt 'enter customer id ':
input item.id
read item from customer.file,item.id else stop 202,item.id
print 'the customer name is ':item<1>
This example opens the "customer" file and prompts the operator for
the customer's item-id.  If the read is scuccessful, the customer
name in attribute 1 is displayed.
file customer
crt 'enter customer id ':
input item.id
read customer from item.id else stop 202,item.id
print 'the customer name is ':customer(name)
This example used the "file" statement to bind the "customer" file to
this program. It prompts for the item-id, reads the item and prints
```

```
the name. The "file" statement allows dictionary-type references to
the read data item.
```

READNEXT

retrieves the next item-id from an active list and assigns it to a specified variable.

A list must be active before the "readnext" takes place. See the "select" and "execute" statements and the "system" function (11). The list may be generated within the program using either of these statements, or the list may be passed into the program from an external process, such as TCL or a PROC, which invokes a list producing verb immediately before running this program.

The "value.count" parameter indicates the position of the multi-value within an attribute. This is a by-product of an "exploded" sort, executed prior to the execution of the program. This allows multi-values to be retrieved in "exploded" sort sequence. If the list is not "exploded", the "value.count" value is always 1 (one).

The "else" condition is executed when there are no more items in the list.

If no "select.variable" is specified, the default "primary" or "secondary" select variable is used. The "secondary" specification uses the active "secondary" list. (See the "s" option on the list-producing verbs, such as "sselect").

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

READNEXT id.variable{,value.count} {secondary} {FROM select.variable} {THEN | ELSE statement.block}

Example

```
open 'customer' to customer.file else stop 201,'customer'
execute 'sselect customer with balance.due by name'
loop
  readnext item.id else exit
  read item from customer.file,item.id then
    print 'the name is ':item<1>
  end
repeat
The "readnext" command retrieves the next item-id from an active
list, but does not implicitly address the file.  The file to be
accessed must be opened before an attempt to use the item-id in a
"read" statement can be made.
execute "select md (s"
loop readnext id secondary else exit
  print id
repeat
This builds a "secondary" list and displays each item-id.
```

READT

reads a tape record and assigns the value returned to a specified variable.

The length of the tape record is specified by the most recently executed "t-att" command.

The "then" clause is taken if the operation is successful.

Either "else" or "onerr" can be specified, but not both. If the tape unit has not been attached or if an end-of-file (EOF) mark is read, the "else" or "onerr" clause, if present, is executed. In addition, the "onerr" statement can be used to check for end-of-tape (EOT), tape unit not ready, parity error, or block transfer error. See the system(0) function.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

READT variable [THEN | ELSE | ONERR statement.block]

Example

```
execute 't-att'
rewind else stop
eot=0
loop
  readt rec else eot=1
until eot do
  print trim(rec[1,25])
repeat
Reads a series of tape records and displays the first 25 bytes of
each record.  The "readt" else clause is usually taken when an
end-of-tape or end-of-data condition is reached.  The "system(0)"
function indicates whether there are any other unexpected problems.
```

READTL

reads a tape label and initializes the internal label storage area.

The "then" clause is taken if the operation is successful.

Either "else" or "onerr" can be specified, but not both. If the tape unit has not been attached or if an end-of-file (EOF) mark is read, the "else" or "onerr" clause, if present, is executed. In addition, the "onerr" statement can be used to check for end-of-tape (EOT), tape unit not ready, parity error, or block transfer error. See the system(0) function.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

READTL variable [THEN | ELSE | ONERR statement.block]

READTX

reads a tape record, converts it to hexidecimal, and assigns the value returned to a specified variable.

The length of the tape record is specified by the most recently executed "t-att" command.

The "then" clause is taken if the operation is successful.

Either "else" or "onerr" can be specified, but not both. If the tape unit has not been attached or if an end-of-file (EOF) mark is read, the "else" or "onerr" clause, if present, is executed. In addition, the "onerr" statement can be used to check for end-of-tape (EOT), tape unit not ready, parity error, or block transfer error. See the system(0) function.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

READTX variable [THEN | ELSE | ONERR statement.block]

READU

see "read" or "matread".

READV

reads an item from the optionally-specified file.variable and assigns the value contained in the attribute number referenced in the attribute expression to the specified variable.

The "readvu" statement is identical to the "readv" statement, except that the item is "locked", preventing access to that item by any other process.

The "else" clause is taken if the item is not on file. The "then" clause is taken when the item is read successfully.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

If the item is locked by the current process on the same level, the item lock still takes place. The "locked" statements are executed if the item is already locked by another process or by the current at a different level. If "locked" is not specified in a "readvu", the program waits for the lock to be released. system (0), in the "locked" clause returns the port number which has the item locked.

Syntax

READV{} variable FROM {file.variable,} id.expression, ac.expression {LOCKED statement.block} {THEN | ELSE statement.block}

Example

```
readv partlist from inventory,part.number,10 then
  cnt = dcount(partlist,char(253))
  for l = 1 to cnt
    print partlist<1,l>
  next l
end
Items in the "inventory" file have a list of composite parts in
attribute 10. In this example, only attribute 10 is read, and each
sub-part is displayed on a separate line.
readvu next.number from control.file,"nextnum",1 then
  print 'got the next number'
  writev next.number+1 on control.file,"nextnum",1
end
This example reads the first attribute from the "nextnum" item in the
"control.file", and locks the entire item.  The lock is cleared by
the "writev".
```

READVU

see "readv".

relational operators

used to compare both strings and/or numerics.

The relational operators are:

=, #, >=, <=, >, and <

These operators first attempt to convert both operands into numerics. If successful, a numeric comparison is performed at the current precision. If unable to convert BOTH operands into numerics, the operators convert both operands to strings and perform a left-to-right comparison.

The value returned will be non-zero (true) or zero (true).

Syntax

expression relational.operator expression

Example

```
1) print 3 = 2
The result is a 0.
2) print 2 < "dog"
The result is non-zero because 2 was converted into a string which
comes before the string "dog" in alphabetical order.
3) equ am to char(254)
x = "623abc"
print 623 = x[1,3]
The result is non-zero (true).  Although the first 3 characters of x
are actually a string of characters, the "=" operator was able to
successfully convert this into a number and do a numeric test for
equality.
```

RELEASE

releases (clears) items locked with a previous "matreadu", "readu" or "readvu" statement.

If the "file.variable" and "id.expression" are both omitted, all items currently locked by the current process are unlocked.

If the "id.expression" is specified without the "file.variable", the default file.variable is used.

Syntax

RELEASE {file.variable,} {id.expression}

Example

```
readu item from customer.file,item.id then
  input name
  if name # '' then
    item<1>=name
    item<20>=date()
    write item on customer.file,item.id
  end else
    release customer.file,item.id
  end
end
In this example, an item is read from the "customer.file" and locked.
If a new name is entered, the name attribute is changed, a date stamp
is placed in attribute 20 and the item is written, automatically
clearing the item lock. If no name is entered, no write occurs,
requiring the item lock to be cleared with the "release" statement.
```

REM

returns the remainder portion of the result of dividing one number by another.

There is no limitation on the numeric value of the dividend or divisor. The "rem" and "mod" functions are identical.

Syntax

REM(dividend, divisor)

Example

```
for i = 1 to lines
  print rec<i>
  if not(rem(i,60)) then print char(12):
next i
"rem" is used to find the end of a page.  Every 60 lines, a form
feed, char(12), is printed.
```

REM

designates a "remark" statement and all text which follows on the same line is ignored by the compiler.

The characters, "*" or "!" may be substituted for the "rem" statement.

The "remark" statement is used to explain or document the program, allowing the programmer to place comments anywhere in the program without affecting program execution. It is provided for compatibility and readability.

Syntax

REM text

Example

```
rem Last revised 1/1/89
! look for next value
* time to wrap up
```

REMOVE

removes a substring delimited by a system delimiter from a dynamic array.

Before the initial call, position.variable must be set to zero. The variable is updated with the end of the current substring at the end of every call.

The delimiter.variable is updated with the code indicating the delimiter found at the end of the substring returned. The code is one of the following:

0 = end of string

2 = attribute mark - ASCII char(254)

3 = value mark - ASCII char(253)

4 = subvalue mark - ASCII char(252)

5 = char(251)

6 = char(250)

Syntax

REMOVE variable FROM dynamic.array AT position.variable SETTING delimiter.variable

Example

```
x = "a":@am:"b":@vm:"c"
l = 0
remove xx from x at l setting d
print xx,l,d
remove xx from x at l setting d
print xx,l,d
remove xx from x at l setting d
print xx,l,d
This will print the following result:
a      2       2
b      3       3
c      5       0
```

REPEAT

see "loop".

REPLACE

inserts or replaces a specific attribute, value, or subvalue in the string referenced by "dynamic.array.expression" with the value referenced in "string.expression".

"-1" may be specified as the "ac.expression", "vc.expression", or "sc.expression". This appends "string.expression" as the last element in the respective location. If the last character of string.expression is not the correct separator, the appropriate "mark" character (attribute, value or subvalue) is inserted before the "expression" is added.

Alternate method, using dynamic array reference symbols:

dynamic.array.expression<ac.expression {,vc.expression {,sc.expression}> = string.expression

Syntax

REPLACE(dynamic.array.expression, ac.expression; string.expression)

REPLACE(dynamic.array.expression, ac.expression, vc.expression; string.expression)

REPLACE(dynamic.array.expression, ac.expression, vc.expression, sc.expression, string.expression)

Example

```
1) customer.item(1) = replace(customer.item(1),1,1,0,name)   or
customer.item(1) = replace(customer.item(1),1,1;name)
In this example, value 1 of attribute 1 in the first element of the
dimensioned array, "customer.item", is replaced with the data value in the
variable "name". This example could alternately be coded as:
  customer.item(1)<1,1> = name
2) string<1,-1> = value  or
string = replace(string,1,-1,0,value)   or
string = replace(string,1,-1;value)
These statements add a new value at the end of attribute 1 of "string".
```

REPLACE

maintains referential integrity between items that contain bi-directional bridges by eliminating duplicate item-ids.

The "item-id.new" replaces "item-id.old" in bridged-to items. "item-id.old" is deleted after replacement. In order for the data base to be consistent, there should be two-way bridges between the references.

Syntax

REPLACE file.variable,item-id.old WITH item-id.new

Example

```
 Assume the two files exist with the following data.  Both files
contain a bridge to attribute 1 of the other file.
  FN1           FN2
ID  Attr1    ID  Attr1
---------    ---------
b     1      1    b
c     2      2    c
d     3      3    d
Now assume the following FlashBASIC program is run:
open "fn1" to f1
replace f1,"b" with "c"
Now, the files contain the following data:
  FN1           FN2
ID  Attr1    ID  Attr1
---------    ---------
c     1]2    1    c
d     3      2    c
             3    d
```

RETURN

terminates an internal or external subroutine and returns execution control to the statement following the invoking "call" or "gosub" statement.

The "to" clause may only be used with an internal subroutine, and transfers control to the specified statement label. This is not considered a good programming practice.

Syntax

RETURN

RETURN {TO statement.label }

Example

```
input answer
if answer = 'y' then gosub 1000
print 'back again'
stop
1000 * subroutine
print 'ok'
return
In this example, if the answer is "y" then call internal subroutine
1000.  When 1000 is complete, control is "returned" to the line which
prints "back again".
```

RETURNING

specifies the variable which receives the results from an "execute" or "get" command.

Syntax

RETURNING variable

REWIND

rewinds the currently attached magnetic media unit to the "beginning" of the media.

If the tape is not attached or not on-line, the "else" or "onerr" clause, if present, is executed. Otherwise the "then" clause is executed. In addition, the "onerr" statement can be used to check for a "tape unit not ready" error.

Either "else" or "onerr" may be specified, but not both. See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

REWIND [THEN | ELSE | ONERR statement.block]

Example

```
execute 't-att'
rewind else
  crt 'cannot rewind the tape'
  stop
end
```

RND

generates a random number.

The generated number is in the range of 0 and the value specified in a numeric expression, minus one.

The range of the numeric expression may be up to 32,000.

Syntax

RND(num.expression)

Example

```
increment = rnd(8) + 1
The "rnd" function in this example generates a random number between
1 and 8. 1 is added to the result to ensure that the increment is
never zero.
```

ROLLBACK TRANSACTION

See: rollback work

ROLLBACK WORK

Rolls back (or undoes) a transaction

This operation atomically undoes all updates made since the last "begin work" command. All locks acquired within the transaction are released.

Syntax

ROLLBACK {WORK | TANSACTION} {THEN | ELSE statement.block}

Example

```
open "myfile" to f1
open "mylog" to f2
begin work
write total on f2,timedate();* create synchronized log
readu total from f1,"total"
if total = 0 then
  rollback work;* Cannot do anything
end else
 total = total - 1
 write total on f1,"total"
 commit work else print "Could not update"
end
```

ROOT

provides an interface to the b-tree indexes for subsequent references with the "key" statement.

This is somewhat like an "open", in that the statement establishes the root fid of the given index and assigns it to the "root.variable". Subsequent use of the "key" statement references the corresponding "root.variable". Any number of indexes may be "open" at once, provided that each has a unique "root.variable".

The "a.code" is a string expression containing the "a" (algebraic) processing code located in the file-defining item which identifies the index to use. The index must have previously been created with "create-index".

"root.variable" is for use by subsequent "key" statements.

The optional "then" clause is executed if the specified index is found. The "else" clause is executed if it is not found.

The "key" statement, in conjunction with the initializing "root" statement, provides the ability to sequentially cruise on any defined index for a particular file.

The "root" statement must precede the "key" statement.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

ROOT file.reference,a.code TO root.variable {THEN | ELSE statement.block}

Example

```
See "key".
```

RQM

Suspends processing for a specific amount of time.

SCAN

searches through a given "string.expression" for the first occurrence of up to three user-definable character(s) specified by "search.delimeter(s)" in addition to the system delimeters: subvalue, value, and attribute. If the delimeter is found, "scan" returns the numeric position at which the occurence was found. The user-defined characters must be separated by a system delimeter (' " \).

If the specified search delimeters or any system delimeters are not found, "0" (zero) is returned.

Syntax

SCAN(string.expression, search.delimeter(s))

Example

```
d.position = scan(\The "first" or 'second' name.\,\'\:AM:\"\)
This returns the value, "5", to the variable "d.position", since the
first occurrence of either \'\ or \"\ is found in the fifth position
of the string.
on scan("abcdefg","c":VM:"e":VM:"g") goto 10,20,30,40,50,60,70
This example exploits the fact that "scan" returns a number
indicating the position of the search character. For instance, if "c"
is found then it will goto statement label 30.
loop
 input ans,1
 until scan('abc',ans) do repeat
If "ans" is either an a, b, or c, the loop is terminated.  Notice
that a null value for "ans" results in a "0" from the "scan"
function.
```

screen

utilities to display formatted menus.

The screen utility consists of a series of FlashBASIC subroutines to handle formatted menus. All data is passed between the subroutines and the main program through a COMMON defined in the include "dm,bp,includes, screen.inc". The routines are described in more detail in the appropriate section.

screen.init Initialize the COMMON. This routine must be called before any other routine is called. After calling it, the main program may modify some the COMMON variables.

screen.display Display a screen and wait for a menu selection.

screen.input Display an input screen and wait for user data to be input.

screen.erase Erase a screen section.

screen.display

displays a screen and waits for a selection.

Input :

The COMMON variables defined in "dm,bp,includes screen.inc" must be set as follows:

screen — Array of up to 512 options. The options are displayed in a columnar format, prefixed by their number on 3 digits. The routine then waits for user input. The first 10 options can be selected by entering their number. Other options can be selected by typing the first letter of the option or by moving the cursor (CTRL-N: Next; CTRL-B: Back; CTRL-J: Move left; CTRL-K: Move right).

screen.help	Array of help messages. The message is displayed starting at the line specified by 'screen.msgline'. A value mark is a line separator. If non null, the help is displayed starting at the line specified by 'screen.msgline' when the cursor is moved to the corresponding option.
screen.size	Number of actual entries in the 'screen' array.
screen.title	Title. Displayed underlined.
screen.x	
screen.y	Coordinates of the upper left corner of the menu, starting at 0,0.
screen.active	Option number to set as 'active' (highlighted).
screen.lastline	Last line available to display the menu. If there are more options than there are available lines, the menu is displayed in columnar format. If there are more columns than possible to display on the 80 columns screen, only some columns are displayed.
screen.width	Width of an option. The default is 35 characters.
screen.doselect	Boolean. If true, the user can select more then one option before validating by hitting the <space> bar. An asterisk is displayed in front of the selected options. The list of selected option numbers is returned in 'screen.select'.
screen.msgline	Line number where the help messages can be displayed. The message area then extends up to the end of the screen.
screen.notice	If non null, contains the name of an external subroutine which is called periodically. The main program can provide a polling routine to display user information or perform some periodic task.

Output :

The following variables are then updated:

screen.active User selection:

n : Option n

Q : Quit or ESC

X : Exit to TCL

screen.select If 'screen.doselect' is true, contains the list of option numbers, separated by an attribute mark, selected by the user.

screen.maxsize Last line actually displayed on screen. In most cases, it is screen.y+screen.size, but, if there are too many options to fit in one column, this variable will reflect the actual last line on screen. This is used by screen.erase.

Syntax

call "dm,bp, screen.display"

Example

```
Example
include dm,bp,includes screen.inc
* Init the screen
* ---------------
call "dm,bp, screen.init"
```

```
* Clear the whole screen
* ----------------------
crt @(-1)
* Build the menu
* --------------
screen.x=0 ; screen.y=0      ;* Top left corner
screen(1)="User"             ;* Option 1
screen(2)="Account"
screen.help(1)="User name"   ;* Help messages
screen.help(2)="Account name"
screen.size=2                ;* Number of options
screen.active=1              ;* Option 1 is default
screen.msgline=23            ;* Help line
* Display the menu
* ----------------
call "dm,bp, screen.display"
begin case
  case screen.active='x'
    * Back to TCL
    stop
  case screen.active='q'
    * User typed 'q' or ESC
    ....
  case 1
    * Selected an option from 1 to screen.size
    ...

end case
```

screen.erase

erases a formatted screen.

Input :

The following variables in the COMMON variables defined in the include "dm,bp,includes screen.inc" must be set:

screen.x

screen.y Coordinates of top left corner.

screen.maxsize Last line used on the screen. This variable was set automatically by a call to 'screen.display' or 'screen.input'.

Syntax

call "dm,bp, screen.erase"

screen.init

initializes the screen COMMON workspace. This call should be done prior to any other call the screen subroutines.

Syntax

call "dm,bp, screen.init"

screen.input

displays a formated screen and waits for input from a user. Multiple fields can be entered by the user before validation. Type checking is done by the subroutine, and the caller may provide a custom external routine to perform additional controls.

Input :

The COMMON variables defined in "dm,bp,includes screen.inc" must be set as follows:

screen Array of up to 512 options. If multivalued, the options are presented in a multicolumnar format, with the input field underneath the headers. Data is entered and validated when the cursor is moved out of the field by: CTRL-N: Next; CTRL-B: Back; CTRL-J: Move left; CTRL-K: Move right. CTRL-X erases an input field.

screen.data Array of data. If non null, the data is displayed and can be modified by the user.

screen.type Array of type controls. Controls user input. If the corresponding 'screen' element is multivalued, the control applies to all multivalues. See the section 'Input Control' below.

screen.help Array of help messages. The message is displayed starting at the line specified by 'screen.msgline'. A value mark is a line separator. If non null, the help is displayed starting at the line specified by 'screen.msgline' when the cursor is moved to the corresponding option.

screen.size Number of actual entries in the 'screen' array.

screen.title Title. Displayed underlined.

screen.x

screen.y Coordinates of the upper left corner of the menu, starting at 0,0.

screen.active Option number to set as 'active' (highlighted).

screen.lastline Last line available to display the menu. If there are more options than there are available lines, the menu is displayed in columnar format. If there are more columns than possible to display on the 80 columns screen, only some columns are displayed.

screen.width Width of an option. The default is 35 characters.

screen.msgline Line number where the help messages can be displayed. The message area then extends up to the end of the screen.

screen.notice If non null, contains the name of an external subroutine which is called periodically. The main program can provide a polling routine to display user information or perform some periodic task.

screen.control If non null, contains the name of an external subroutine which performs custom controls on user input. This routine must either be cataloged in the current account, or, better, be an explicit path name, like 'accounting,bp, check.input'. See the section 'Input Control Subroutine' below

Output :

The following variables are then updated:

screen.active User selection:

Q : Quit or ESC

Y : Validated

screen.maxsize Last line actually displayed on screen. In most cases, it is screen.y+screen.size, but, if there are too many options to fit in one column, this variable will reflect the actual last line on screen. This is used by screen.erase.

Input Control :

The 'screen.type' array elements contains the following codes:

<null> Optional field. May be left empty. No control.

t Required field. Any text.

x Display only. The user is not allowed to modify the data. The cursor cannot be moved into the corresponding field.

l Explicit list. The user can enter only one of an explicit list of elements. Each element is separated by a tilde. For example: 'l~yes~no~'. Note a trailing tilde is required.

y Confirmation field. The user can only select [y{es}|n{o}|q{uit}]. If 'y{es}' is entered, the routine terminate.

n Numeric field. Followed by the min and max legal values (inclusive), separated by a dash. For example: 'n0-1024'.

<number> Custom control code. This strictly positive number is passed as the first argument to the user provided external routine specified in 'screen.control'.

Input Control Subroutine :

The user can specify an external subroutine in 'screen.control' which is called when the control code in a 'screen.type' array element is a number. This number is passed as the first argument of the subroutine, along with other information. The subroutine can then either validate the subroutine, and/or modify the user input. The user must provide a subroutine with the following arguments:

Input:

1st arg: Control type number

2nd arg: User input data

3rd arg: Data field we are leaving (the one to control)

4th arg: Data field we are entering (where the cursor now is)

Output:

1st arg: Set to 0 if data is invalid, or 1 if ok.

2nd arg: Validated data, if 1st arg=1

3rd arg: Unchanged

4th arg: Unchanged

Syntax

call "dm,bp, screen.input"

Example

```
Example :
The following example displays a screen like:
User    : ..........
Order #1          Order #2
.................  .................

include dm,bp,includes screen.inc
* Init the screen
* ---------------
call "dm,bp, screen.init"
* Clear the whole screen
* ----------------------
crt @(-1)
* Build the menu
* --------------
```

```
screen.x=0 ; screen.y=0      ;* Top left corner
screen(1)="User"             ;* Option 1
screen(2)=""
screen(2)<1,-1>="Order #1  " ;* 1st column
screen(2)<1,-1>="Order #2  " ;* 2nd column
screen.type(1)='t'           ;* Mandatory text
screen.type(2)=1             ;* User provided control
mat screen.data=''           ;* Clear result
screen.help(1)="User name"   ;* Help messages
screen.help(2)="Account number and val date"
screen.size=2                ;* Number of options
screen.active=1              ;* Option 1 is default
screen.msgline=23            ;* Help line
screen.control="account,bp, check.input"
* Display the screen and input data
* --------------------------------
call "dm,bp, screen.input"
begin case
  case screen.active='x'
    * Back to TCL
    stop
  case screen.active='q'
    * User typed 'q' or ESC
    ....
  case screen.active='y'
    * OK. Validated
    ....

end case
```

SELECT

creates an "active" list of item-id's, allowing sequential access to each item in the file by use of the "readnext" statement.

Unlike the AQL "select" or "sselect" verbs, the FlashBASIC "select" statement can NOT have selection criteria or perform a sort. The FlashBASIC "select" passes through every item in the file in "hashed" order.

If the file.variable parameter is not specified, the default file.variable is used. When used with the "to" clause, the item list is assigned to the specified select variable.

If an external list is already active when the program is executed, or the program performs an "execute" of an AQL "select", "sselect", "qselect", or "get-list", the active list is returned by the FlashBASIC "select", irrespective of the passed file.variable.

Syntax

SELECT

SELECT variable {TO list.variable}

select file.variable {to list.variable}

Example

```
open 'customer' to customer.file else stop 201,'customer'
select customer.file
eof=0
loop
  readnext id else eof = 1
until eof do
  print id:" exists"
repeat
```

```
The "customer" file is opened and every item is selected.
select customer.file to customer.list
eof=0
loop
  readnext id from customer.list else eof=1
until eof do
  print id:" exists"
repeat
The customer.file is selected and assigned to the list.variable
"customer.list".
string = '1001':char(254):'1002':char(254):'1003'
select string to list
eol=0
loop
 readnext id from list else exit
 print id:" exists"
repeat
The array.variable, "string", is treated as an active list by
assigning it to the list variable, "list".
open 'md' to md.file else stop 201,'md'
execute 'select md with a1 = "d]"'
select md.file to md.list
loop
  readnext file.name from md.list then
    open file.name to temp.file then
      execute 'select ':file.name
      select temp.file to temp.list
      loop
        readnext temp.id else exit
        print temp.id:" in ":file.name:" exists"
      while 1 do repeat
    end
  end else exit
until 1 do repeat
This example demonstrates multiple active lists in the same program.
This example first selects all local file pointers from the master
dictionary and assigns them to the "md.list" list variable using
"select". Each file name is used to select all items in the data
section.
```

SEND

sends output to a specified port.

The port to which the output is being sent must be attached with the "dev-att" command.

In the "send" statement, the first expression is evaluated as a string. This string may contain any data formatting expressions which work in the "print" or "crt" statements including cursor positioning, "@(column,row)", and special functions, "@(-n)". The actual codes that are generated is determined by the terminal characteristics of the process executing the "send", NOT the port to which the characters are being passed. The correct terminal type must be established with the "term" command.

The optional ":" (colon) supresses a terminating carriage return, line feed.

The "port.number" expression must evaluate to a valid port number. If the port number is invalid, and there is no "else" clause, the program aborts into the FlashBASIC debugger. If there is an "else" clause, it is executed when the port.number is invalid.

The "send" statement is typically used after a port has been attached by the sending process. Attachment ensures that the process has exclusive use of the port. Data may be sent to any port

that is linked (but not attached) to another process. If the port is attached to another process, the "else" clause is executed.

Both the "then" and "else" clauses are optional, but, at least one must be specified.

The "sendx" statement converts the exploded ASCII hexadecimal string results of "string.expression" to its binary equivalent and then transmits it to the specified port. The conversion process terminates when the first non-hexadecimal character is encountered. "sendx" supresses the output of a carriage return, line feed pair. "sendx" does not allow the ":" (colon) to exist as part of the statement.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

SEND{X } string.expression{:} TO port.number {THEN | ELSE statement.block}

Example

```
execute "dev-att 10"
send 'begin transmission' to 10 else print 'port in use'
execute "dev-det 10"
This sends the string, 'begin transmission', to port 10. If port 10 is already
attached by another process, the "else" clause is executed.
send msg: to device1 then print 'message sent'
The string in "msg" is sent without a carriage return and line feed to the
device addressed by "device1". If successful, the "then" clause is executed.
sendx msg to device1 then print 'hex message converted to binary and sent'
The binary equivalent of the ASCII hexadecimal string in "msg" is sent to the
port addressed by "device1" and if successful, prints a message.
execute "dev-att 10"
string = "hello there"
send @(10,9) : string : to 10 then crt "string sent" else crt "string
not sent"
This sends the string, "hello there", to port 10 at position column position
10 on row 9. The term type of the SENDER must match with the terminal of the
RECEIVER for this to work correctly.
```

SENTENCE

This function is identical to performing a TCLREAD statement.

Syntax

SENTENCE()

SEQ

converts any ASCII character to its corresponding numeric equivalent in the range 0 to 255.

This "seq" function is the opposite of the "char" function.

Any expression can be used as an argument, but all characters are ignored beyond position 1.

The "seq" of a null string ends up using the segment mark (x'ff') end-of-string delimiter as the argument resulting in a value of 255.

Syntax

SEQ(ascii.character.expression)

Example

```
input c,1
if seq(c)<32 then stop
This example stops when any control character is input.
```

SETTING

defines the variable to receive the index results of a "get" or "locate" command.

Syntax

SETTING variable

SIN

calculates the sine of an angle specified in degrees.

The result of this function is a fraction in the range -1 to 1. If the expression is less than 0 or greater than 360 degrees, mod(expression,360) is used to adjust it to this range before the sine is calculated.

Note: This function is only accurate to 3 digits.

Syntax

SIN(num.expression)

SLEEP

places a process to sleep for a specific number of seconds, or, until a specific time.

The "sleep" time may be specified as a numeric expression, which indicates the number of seconds to "sleep", or as a "time.expression" in "military time" format (hh:mm:ss). If no argument is indicated, the process sleeps for one second. "rqm" and "sleep" are identical.

Releases 6.1.0 and above allow the specification of fractional seconds in the numeric expression. Also, a numeric expression of zero will cause the process to relinquish its time slice.

Syntax

SLEEP {num.expression}

SLEEP {"time.expression"}

Example

```
sleep 3600
This puts the process to "sleep" for one hour.
until.wakeup.time = "08:00"
sleep until.wakeup.time
This puts the process to sleep, leaving a "wakeup call" for 8:00 a.m.
```

SORT

sorts an attribute or value mark delimited string.expression in ascending order.

If the expression contains both attribute and value marks, this function replaces all marks with the type first encountered.

A second sort function is available that works in a manner similar to the "locate" statement. On this expanded version, the user may specify in which attribute, and or value the list resides as well as the sort sequence to use.

Syntax

SORT(string.expression)

SORT(string.expression, ac.expression, vc.expression, sequence.expression)

Example

```
equ vm to char(253)
list='zebra':vm:'albatross':vm:'gooney bird':vm:'elephant'
newlist=sort(list)
```

```
The variable "newlist" contains the string:
albatross]elephant]gooney bird]zebra
The "]" (bracket) represents a value mark.
```

SOUNDEX

returns the four-digit soundex code for a phonetic string expression.

The soundex code consists of the first letter of the word, plus values for the next three additional consonants. If there are less than three consonants, zeros are placed in the code. Soundex codes are values given to consonants. Words with a similar arrangement of consonants have similar soundex codes, regardless of the actual spelling. Also, similar sounding consonants may have the same soundex code.

The "code" option provides compatibility with the R83 "U00B9" user exit, where:

code=0 Original census soundex; R83 compatible (default)

code=1 English-language soundex

Syntax

SOUNDEX(string.expression{,code})

Example

```
crt soundex("PICK")
This outputs "P2".
```

SPACE

generates a string of spaces whose length is equal to the value of the numeric expression.

Syntax

SPACE(num.expression)

Example

```
crt "name" : space(22) : "phone number"
```

special characters

defines reserved characters.

Each of the following characters are reserved for special purposes:

: ; @ * ! & = # - + / ^ < > ' " \ [ ] ( )

They may be referenced individually by their respective symbol.

Each of these characters have special functions and they can NOT be used in variable definitions.

SPOOLER

a function provided purely for compatibility reasons. It always returns 0 on D^3.

Syntax

SPOOLER(string, string)

SPOOLQ

enables or disables the spooler entry number message

In the "expression" form, the entry number printing is disabled when the expression evaluates to false. It is enabled when the expression evaluates to true. In an expression, 0 = "spoolq off" and non-zero = "spoolq on".

Syntax
SPOOLQ [OFF | ON | expression]

SQRT

calculates the square root of a given numeric expression.

If the expression is negative, the function returns 0 and prints an error message.

Precision is kept for up to 9 digits.

Syntax
SQRT(num.expression)

Example
```
print sqrt(25)
```

SQUOTE

extracts a single-quoted string from string.expression.

The "squote" function returns the first substring found which is surrounded by single-quotes. If no substring is found matching this criteria, then a null string is returned.

Syntax
SQUOTE(string.expression)

STATUS

returns the value of system(0).

Syntax
STATUS()

STOP

stops program execution and returns to the invoking the process.

The parameter{s} are passed to the error message handler and the message stored in "error.message.number" is displayed.

The "stop" statement is similiar to the "abort" statement, except the "stop" statement terminates execution of the program without going into the FlashBASIC debugger. Control returns to the calling program, PROC, or Macro.

Syntax
STOP

STOP error.message.number, "parameter"{, "parameter" .. }

Example
```
>open "invoices" to inv.file else stop 201,"invoices"
The "stop" statement above stops program execution if "invoices" can
not be opened, and prints the designated error message with a passed
variable parameter. If the "invoices" file does not exist, the
following message displays:
[201] 'invoices' is not a file name
read cust.item from cust.file,id else stop 202,id
If the item referenced by the variable "id" is not on file, the
program stops and displays the following:
[202] 'id' not on file.
The actual item-id is displayed in the error message.
```

STR

repeates a "string.expression" the number of times specified in "num.expression"

See the "system" function for retrieving the current device width to substitute as the numeric expression.

Syntax

STR(string.expression, num.expression)

Example

```
print str("*",79)
This prints 79 "*" (asterisks) at the current cursor (or print head)
position.
print str("=",system(2))
This prints a string of "=" (equal signs). The number of "="'s that
it prints is determined by system(2), which contains the current
device output width most recently designated with a "term" command.
```

SUBROUTINE

defines program as an external subroutine.

FlashBASIC provides the ability to "call subroutines". The "subroutine" statement must appear on the first line of an external subroutine invoked by a "call" statement.

Arguments defined in the optional "argument.list" are delimited by "," (commas). The "argument.list" must have the same number of arguments as are in the "call" statement.

If a subroutine is called from the Update processor, in either a file-defining item, or an attribute-defining item, no "argument.list" may be specified by the "call" statement. If called from a file-defining item, the "item" being filed is automatically passed as a parameter in a call. If called from an attribute-defining item, the current value of the attribute being processed is automatically passed. Note that the subroutine will be called once for every value in the attribute.

A subroutine exits with a "return" statement.

All external subroutines must be cataloged prior to execution.

Syntax

SUBROUTINE {(argument.list)}

SUBROUTINE subroutine.name{(argument.list)}

Example

```
*program main
10 input start.date
call validate.date(start.date,ok)
if not(ok) then goto 10
This example prompts for a date and calls the external subroutine
"validate.date" to make sure the date is legal.  Here is the
subroutine:
subroutine validate.date(pdate,ok)
*    may need rework for years crossing 2000
if pdate matches "1n0n1x1n0n1x2n0n" then
   if iconv(pdate,'d') # "" then
      ok=1
   end else
      ok=0
   end
end else
```

```
   ok=0
end
return
The variable "ok" is set to 1 if the date is valid.  Otherwise, "ok"
is set to zero (false).
```

SUM

returns the sum of a list of numbers delimited by attribute, value, or subvalue marks. If different orders of marks are present, the "sum" function computes the sum of each sublist.

Syntax

SUM(string.expression)

Example

```
equ am to char(254)
print sum(3:am:3.2)
The result, "6.2", will be printed to the terminal.
equ am to char(254), vm to char(253)
print sum(3:vm:3.2:am:6:vm:2)
The result, "6.2^8", prints because "sum" senses that there were two
lists present, separated by an attribute mark.
```

SUMMATION

returns the sum of a list of numbers delimited by attribute, value, or subvalue marks. If different orders of marks are present, the "summation" function will add the contents of all fields regardless of delimiter.

Syntax

SUMMATION(string.expression)

Example

```
$options ext
equ am to char(254)
print summation(3:am:3.2)
The result, "6.2", will be printed to the terminal.
```

SWAP

searches a "string.expression" for the "search.string" and then, if at the "start" occurrence, will replace "search.string" with "replacement.string" for "occurrences" occurrences. If "start" or "occurrences" are omitted, they will default to 1 and 0 respectively. An "occurrence" setting of 0 will replace all occurrences of "search.string" with "replacement.string".

Syntax

SWAP(string.expression, search.string, replacement.string {,occurances {,start}})

Example

```
nstring = swap("The new time","new","old")
The assigns the value "The old time" to nstring.
 print swap("---","-","+",2,1)
 This will output "-+-". The "swap" function will replace one
occurrence of "-" starting at the second occurrence.
```

SYSTEM

provides an interface to a number of system variables, depending upon the requested numeric argument, as shown below:

0 Context-oriented status information.

After a tape-handling error, system(0) returns:

1 if not attached.

2 null variable.

3 attempt to write null string.

5 EOT encountered.

6 tape write protected.

7 tape unit not ready.

8 unrecoverable parity error.

9 block transfer error.

10 binary item read (see note below).

11 record truncated.

12 unrecoverable write error.

13 unrecoverable read error.

14 no tape media inserted.

15 tape subsystem not ready.

After a "readu", "readvu" and "matreadu" statement which contains a "locked" clause, system(0) returns the port.number that has the item locked.

Value 10 actually has nothing to do with tape and is relevent to all "read"-type statements. It is set when an "indirect" (or "pointer") item is read.

On D3 Unix implementations, system(41) returns the value of "errno" after a C function call.

All statements which allow the "onerr" clause provide a "hook" to system(0). See "onerr" for examples.

1 Returns 1 if the system printer is "on", meaning that a "printer on" statement has been issued or that the program was activated with a (p) option. Otherwise, a 0 (zero) is returned, meaning that output is being directed to the terminal.

2 Returns the current output device page width as defined by the "term" command.

3 Returns the current output device page length as defined by the "term" command.

4 Returns the number of lines remaining to print on the current page, based on the current terminal characteristics previously defined with the "term" command. Note: use of the @(row,col) function will not change this value.

5 Returns the current page number. Note: use of the @(row,col) function will not change this value.

6 Returns the current line number (Not the "port" number -- the actual print line number). Note: use of the @(row,col) function will not change this value.

7 Returns the terminal "type" code, as defined by the "term" command.

8 Returns the block size at which the tape was last attached.

9 Returns the current cpu millisecond count.

10 Checks the current stack (ston) condition. Returns 1 (one) if the stack is on, or 0 (zero), if not.

11 Checks for an externally-generated active list.

It returns "0" if there is no active list, or "n" (the actual number of items selected), if an active list exists.

This has nothing to do with the internal (FlashBASIC) "select" statement, which requires a "readnext" to determine if any items were selected. See the "readnext" and "select" statements for additional warnings on using system(11).

12 Returns the system time in milliseconds.

13 Forces an rqm (terminates timeslice) and returns a 1. Deactivates the process in the scheduling queue until its next turn.

14 Returns the number of bytes in the type-ahead input buffer of the current process.

15 Returns the TCL verb option{s} that are in effect.

16 Returns the current process level (push-level).

17 Returns the message numbers (item-ids) returned by the previous "execute" statement, separated by attribute marks.

18 Returns the number of ports on the system.

19 Returns a unique item-id consisting of the current system date in internal format, followed immediately by the current system time in seconds. If more than one item-id is generated in a second, an alpha character is appended to the item-id.

20 Returns the most recent Spooler entry number generated by the current process.

21 Returns today's date in internal format with a sequential numeric suffix.

22 Returns the port.number of the current process.

23 Returns the current process privilege level.

24 Returns 1 if the current process is a phantom. Otherwise it returns 0 (zero).

25 Checks for an active secondary list. Returns the number of items in the secondary list, or 0 (zero) if no secondary list is active.

26 Returns 1 if "capturing on" is in effect. Otherwise, it returns 0 (zero).

27 Returns 1 if TCL case-sensitivity is in effect for the current process. Otherwise, it returns 0 (zero). (see "casing").

28 Returns 1 if the current FlashBASIC program is data case-sensitive, or 0, if not. The first time a FlashBASIC program is entered, the FlashBASIC case flag is set to the current process' case flag. Any subsequent "casing" statement modifies the FlashBASIC flag but not the current process' flag. Only the current process' flag is passed between levels. (see "basic" and "compile" commands).

29 Returns 1 if R83-format 'who' is in effect (port.number, account name, user-id) or 0 (zero) if "D3-format" is in effect (port.number, user-id, account name). See the "who" verb for options.

30 Returns the port.number of the process which has the item locked. (see "locked").

31 Returns the current Spooler form queue number most recently assigned with an "sp-assign".

32 Returns the data frame size for the system.

33 Returns the name of the FlashBASIC program which called the subroutine.

34 Is currently not supported and should not be used.

35 Returns the total number of physical ports on the system, excluding phantom ports.

36 Returns a non-zero if running a FlashBASIC program, otherwise, it returns a 0 (zero).

37 Custom Interrupt Handling in FlashBASIC. A generalized method is provided by which FlashBASIC applications may handle the break key or any other type of interrupt in a special manner. This function returns the value of an internal location indicating the last interrupt processed and then clears that location. Note that although the only way to poll this value is with the system(37) function in FlashBASIC, the internal value is ALWAYS set, even if the application is not in FlashBASIC. The following numbers are valid return codes:

00 No interrupt has occured

03 Another process sent a message to the current line

10 The break key was hit

12 Escape level push

13 Another process TCL'd the current line

Other values are undefined and should not be relied upon.

The system(37) function has only one limitation in that if a process is waiting at input, the user must type a key before the application can poll the interrupt status. All other operations, including sleep, will complete immediately after a break allowing the program to detect that interrupt in real time. This feature has several applications. See the 'Example' section below.

38 Returns the system identification and options. A string with four attributes as follows is returned:

host ^ chip ^ imp ^ opt

host : Underlying host operating system (decimal).

chip : Processor chip code (decimal).

imp : Implementation code (decimal).

opt : Option string (one character per option)

The values of each field are defined in the include "dm,bp,includes sysid.inc" which should be included in programs using this function.

39 Returns a non-zero (usually 1) if the spelling checker is enabled, or 0 if it is not.

40 Returns the current tape reel number, or -1 if not attached.

41 Returns the Unix Errno variable.

100 Returns the system identification:

system;filename:name;release; version;hardware; date

system : The system name, which may contain spaces.

filename : Configuration file name.

name : Contents of the 'name' statement in the configuration file.

release : The current release level.

version : The current version level.

hardware : The type of hardware.

date : The date of the release.

On D3 Unix implementations, the format returned by system(100) is as follows:

D3 Unix: system; o/s name; filename:name; release; version; hardware; boot monitor version; date

system The system name, which may contain spaces.

o /s name "SCO", "AIX", etc.

filename Configuration file name.

name Contents of the 'name' statement in the configuration file.

release Unix system release.

version Unix system version.

hardware Machine hardware name, or serial number.

boot monitor version The release level of the monitor.

date Boot monitor date.

Syntax

SYSTEM(num.expression)

Example

```
1)  system(37):
When doing an "execute" of a Unix shell command, the Unix command will simply
abort if it gets a break signal (the normal action in Unix). The following
code will sense that a break has occured and reexecute the Unix instruction,
giving the appearance of a standard Pick break-and-continue capability:
do.unix:
x = system(37); * Clear the interrupt value
execute "!exec find / -name ap - print"
if system(37) = 10 then goto do.unix; * Break - Return, reexecute
Another useful application is for a FlashBASIC program to mimic the behavior
of the Update processor's interrupt handling and redraw the screen background
if it senses an interrupt. For example:
gosub refresh
open "data"
select
loop
readnext id else exit
read rec from id
name = rec<1>
balance = rec<2>
balance = balance + 50
print @(20,2):name
print @(20,3):balance
if system(37) then gosub refresh; * Screen might be hosed
repeat
stop
refresh:
* Draw the background
print @(-1)
print @(10,1):"Processing Accounts"
print @(10,2):"Name:"
print @(10,3):"Balance:"
return
By using the system(37) in conjunction with the "break off" statement, a
FlashBASIC program can implement its own signal handler. For example, assume a
user wishes to update many items in a file, but he or she wants this done
atomically, i.e. either all the items in the file must be updated or none of
the items must be updated. This is a simple form of transaction bracketing and
```

can be implemented on previous releases by simply turning off the break key. However, this method has the disadvantage of not being able to interrupt the process. The system(37) capability provides the ability to offer a "controlled" interrupt allowing the user to continue, or back out of all changes made so far. This routine could be expanded to allow other operations provided it didn't turn the break key on (which would cause the system to catch the break and push a level, allowing the user to type "end"). The example is as follows:

```
break off
x = system(37);* clear previous signals
max = 2000
dim xx(max)
for i = 1 to max
readu xx(i) from i
writeu "abc" on i
if system(37) = 10 then gosub interrupt; * break key hit
next i
* Release the locks
for i = 1 to max
release i
next i
stop
interrupt:
print "Interrupted"
print "Processed ":i:" items"
print "C)ontinue or Q)uit without modifying file:":
in x
print
x = char(x)
if x = "q" then
* Change the items back to the way they were
max = i
for i = 1 to max
* note that write will release the locks along the way
write xx(i) on i
next i
stop
end
return
```

Finally, system(37) increases the capabilities of D3 in the area of interprocess communications. For example, a batch job can poll system(37) repeatedly and when it senses an interrupt, it can either simply send a message indicating its progress, or do some other, more complex action by reading a command item from a file. Note that using the system(37) is much more efficient that reading commands and writing status output for every item. An example is as follows:

```
xx = system(37);* clear status
open "data"
for i = 1 to 100000
* In a real program, there should be more work done in the main loop.
* Otherwise, the system(37) becomes comparatively more expensive.
write "abc" on i
if system(37) then execute "msg !0 Processed ":i:" items."
next i
stop
* "z" the preceeding program and then type the following
* tcl (line#) (user) who
*
* The actual command executed is unimportant.  The who command is a
*   good candidate since its cheap and has no side effects.
*
* Note that the msg does NOT work with phantoms, but the "tcl"
```

```
command
*   will.
*
2)  system(38):
include dm,bp,includes sysid.inc
imp=system(38)
if imp<sys$host>=sys$unix then
* Running on Unix implementation
n=%open( "myfile, O$RDONLY)
....
end
3)  system(100):
crt system(100)
D3 Unix: RS6000;AIX;pick0:PROD0;2;3;000047311000;6.0.0.M0;27 May 1992
```

TA

toggles or resets the type-ahead buffer.

The FlashBASIC "ta on/off" statement is equivalent to the TCL command "type-ahead."

The FlashBASIC "ta clear" statement is equivalent to the BASIC statement "inputclear."

Syntax

TA [CLEAR | OFF | ON | expression]

TAN

calculates the trigonometric tangent of the angle specified in degrees.

The numeric expression is normalized to the range of 0 - 360 degrees.

Syntax

TAN(num.expression)

Example

```
sin.10 = sin(10)
```

TCL

executes any valid TCL statement as a subroutine.

After execution, the FlashBASIC program continues with the next statement.

Input may be passed to the TCL statements using the "data" statement prior to the "TCL" statement.

The "to" modifier passes the last message number and arguments of the TCL statement to the specified dimensioned array variable.

After the "TCL" statement is completed, the data queue (stack) is reset. Multiple "data" statements may be passed when they are separated by value marks. Multiple TCL statements may be passed when they are separated by attribute marks.

If an active list is generated by the TCL command, it is passed back to the FlashBASIC Program. The list can be used by the "readnext" statement or it can be assigned to a specific variable using the "select to" command for later use with a "readnext" statement.

If the "off" statement is issued as the TCL statement, control does NOT return to the FlashBASIC program.

Syntax

TCL tcl.command {TO dimensioned.array.variable)

Example

```
dim results(3)
tcl 'count md if a1 "pq"' to results
This example executes an AQL "count" statement and returns the
error message item-id's to the dimensioned array, "results".  The
output of the "tcl" statement looks like this:
[407] 26 items counted out of 897 items.
The contents of the dimensioned array "results" looks like this:
results(1) = 407
results(2) = 26
results(3) = 897
```

TCLREAD

loads the TCL command used to activate the program into a variable.

The program must be the program called from TCL. "tclread" will not work when called from a subroutine or from "enter" or "TCL" statements. This statement allows parameters to be passed from TCL to a program directly without processing. Individual parameters can be extracted using the "field" function.

Syntax

TCLREAD variable

Example

```
"tclread" is used to extract the file name and item-id from the "tcl" command
line.  The name of the program is "show".  Here is a sample TCL command using
the "show" program:
show bp testprog
program show
tclread sentence
filename=field(sentence,' ',2)
itenname=field(sentence,' ',3)
```

THEN

specifies which statements to execute when the conditional statement evaluates to true.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

THEN statement.block

TIME

returns the current system time in its internal format, representing the number of seconds past midnight.

The parentheses following the function are required, and never contain any arguments.

Example

```
print 'the time is ':oconv(time(),'mth')
This example outputs the current system time in external format.
start.time=time()
loop until (time()-start.time)>30 or system(14) do
   rqm
repeat
This is an example of a loop which terminates under two conditions. Either a
key is entered on the keyboard (system(14)) or 30 seconds has passed without
any keystrokes.  If no keys are waiting and the timeout is not over, sleep for
a second and check again.
```

TIMEDATE

returns the current system time and date in external format (hh:mm:ss dd mmm yyyy).

The parentheses following the function are required, and never contain any arguments.

Example

```
print @(0,0):'main menu':@(0,50):timedate():
This example shows "timedate()" used for a menu heading.
```

TRANSACTION

enables or disables participation in a transaction.

This function should not be used directly in the context of a user-written BASIC program as it allows violating the theoretical structure of a transaction.

This functionality is intended for languages built on top of BASIC (like some SQL engines or 4GL languages) who need to use temporary files for internal reasons without affecting a user-level transaction.

Note that the word "onoff" is required if an expression is used with this statement.

Syntax

TRANSACTION [OFF | ON | {onoff expr}]

TRANSACTION ABORT

See: rollback

TRANSACTION CACHE

enables or disables the transaction read cache.

Normally, all read operations within a transaction first query the list of uncommitted updates. This is to allow the re-reading of an item already written.

Users can turn this cache off if they are confident that the transaction does not attempt this activity. Turning the cache off can improve the performance of reads within a transaction especially if that transaction is a long one.

Syntax

TRANSACTION CACHE [ON | OFF | expression]

TRANSACTION COMMIT

See: commit

TRANSACTION FLUSH

enables or disables the transaction flush mechanism.

Normally, all updates processed within a transaction are force-flushed to a reliable medium (disk). This is done to assure ACID compliance - i.e. the ability to recover a transaction if the machine should halt unexpectedly.

If ACID ("Atomic", "Compliant", "Isolated", Durable") compliance is not necessary, then this flush mechanism can be disabled providing signicantly higher performance.

Syntax

TRANSACTION FLUSH [ON | OFF | expression]

TRANSACTION ROLLBACK

See: rollback

TRANSACTION START

See: begin work

TRIM

removes leading, trailing, and/or redundant characters from a string.

If only one parameter is specified, then "trim" removes all leading and trailing blanks from a string expression, and compresses multiple embedded blanks to one embedded blank.

If the "trimchar" character is specified, then that character substitutes for the space. If the "type" parameter is missing, then type R is assumed.

If the "type" parameter is specified, then the trim behavior can be controlled more specifically. Available values are:

L trim leading

T trim trailing

B trim leading and trailing

A trim all

R trim redundant

Syntax

TRIM(string.expression, {trimchar {,type } })

Example

```
string = trim("   cat   dog   bird   ")
After trimming, "string" contains : "cat dog bird".
readt record else stop
name = trim(record[1,25])
address = trim(record[26,40])
In this case, the "trim" function is used to convert fixed length fields to
variable length fields by removing trailing and leading blanks.
The following program:
line = '--a--b--c--'
print trim(line,'-')
print trim(line,'-','L')
print trim(line,'-','T')
print trim(line,'-','B')
print trim(line,'-','A')
print trim(line,'-','R')
Would produce the following output:
-a-b-c
a--b--c--
--a--b--c
a--b--c
abc
-a-b-c-
```

TRIMB

removes trailing spaces from a string.

Syntax

TRIMB(string.expression)

TRIMF

removes leading spaces from a string.

Syntax

TRIMF(string.expression)

u$pl.mon.data

The "u$pl.mon.data" user exit returns the following monitor information:

result<1>: address of extended memory table (hex)

result<2>: segment of 1st device driver (hex)

result<3>: offset of 1st device driver (hex)

result<4>: address of top of pib links (hex)

result<5>: pib size (decimal

result<6>: offset of $sio.gen.driver (hex)

result<7>: address of monitor tape buffer (hex)

result<8>: address of tape device code branch table (hex)

Syntax

result = oconv("","u$pl.mon.data")

u$uex.get.buf

The "u$uex.get.buf" user exit gets a block of consecutive frames in memory as close to the specified "max.frames" as possible.

"result" is dynamic array which contains the following information:

result<1> = segment:offset

segment: segment (hex) of requested buffer (0 for ap/prot)

offset : offset (hex) of requested buffer (0 for ap/native)

result<2> = number.of.frames

Syntax

result = oconv(max.frames,"u$uex.get.buf")

Example

```
tape.buf = oconv(128,'u$uex.get.buf')
```

u$uex.rel.buf

The "u$uex.rel.buf" user exit released a block of consecutive frames in memory previously allocated by "u$uex.get.buf".

"block.info" is a dynamic array which contains the following information:

block.info<1> = base address of frames to be released (address is composed of segment:offset)

block.info<2> = number of frames to be released

"result" is a status code indicating:

'0' = successful

'1' = illegal base address segment:offset

'2' = result<2>: number of frames not released

Syntax

result = oconv(block.info,"u$uex.rel.buf")

Example

```
code = oconv("6000:0"<am>"128","u$uex.rel.buf")
```

u0003

returns the TCL command invoking the FlashBASIC program.

The same functionality can be achieved with the "tclread" statement.

Syntax

result = oconv('', 'u0003')

u0004

Returns term settings in dynamic array format:

1. terminal type
2. terminal width
3. printer width
4. terminal depth
5. printer depth
6. lineskip
7. linefeed delay
8. formfeed delay
9. back space
10. <null>

Syntax

result = oconv("","u0004")

Example

```
result = oconv("","u0004")
print result
wy-50^79^80^24^59^0^1^1^8^
```

u0005

clears the type-ahead buffer.

Syntax

result = oconv('', 'u0005')

u000e

gets the name of the current account.

Syntax

account.name = oconv('', 'u000e')

u0010

Creates a phantom process. Logon.data is a dynamic array, which consists of:

user.id] user.password ^ md.id] md.password ^ command ^ options ^ parent.pib# ^ child.pib#

"]" denotes a value mark (char(253)); "^" denotes an attribute mark (char(254)).

Syntax

result = oconv(logon.data, "u0010")

Options

s do not create hold file

m return any messages to parent.pib

u0011

gets the port number of the scheduler process.

Syntax

port.number = oconv('', 'u0011')

u001c

gets the number of attributes (elements) after a "matread".

Syntax

attributes = oconv('', 'u001c')

u001f

returns the last "shutdown" status.

Syntax

status = oconv('', 'u001f')

u0033

converts a decimal number into a Roman numeral.

Syntax

roman.numeral = oconv(decimal.number, 'u0033')

u0039

reads an absolute disk sector.

"read.command" is a dynamic array which contains the following information:

Read.command<1> = drive.number, <vm>, cylinder.number, <vm>, head.number, <vm>, sector.number

Drive.number may be:

1 = drive A:

2 = drive B:

3 = drive C:

4 = drive D:

read.command<2> = mode, <vm>, table1, <vm>, table2, <vm>, #bytes

Mode may be:

1 = convert ASCII to HEX

2 = transform and prefix character

3 = translate and delete character

4 = transform, prefix and drop Most Significant Bit

5 = transform, delete and drop Most Significant Bit

<value mark>

Table1 contains one or more hex characters to scan for on input. If there is more than one entry, then they are separated by <subvalue marks>.

<value mark>

Table2 contains one or more hex characters to replace the corresponding characters from table1. If there is more than one entry, then they are separated by <subvalue marks>.

<value mark>

#bytes is the number of bytes per sector.

Syntax

result = oconv(read.command, 'u0039')

u0065

returns the estimated number of items in the most recently referenced file.

This user exit gives an immediate estimate of the number of items in the file. This estimate is automatically updated by any simple AQL count or select which traverses the entire file, or by the file-save. Normal inserts and deletes also update this estimate although complete accuracy is not assured since this update is not done under lock.

If the internal estimate has never been updated, then a -1 is returned.

To use this user exit, a file reference to the desired file must be accomplished first. A dummy read will accomplish this.

Syntax

oconv("", 'u0065')

Example

```
open "myfile"
.
.
read xx from "";* just reference file
print oconv("","u65")
```

u0079

returns the "where" status for the specified PCB (Process Control Block) frame-id, and its return stack information.

Syntax

result = oconv(pcb.number, 'u0079')

Example

```
b = oconv('02c0', 'u0079')
02c0 is the pcb (which happens to be port 0) to examine.
```

u007A

updates a binary item from a FlashBASIC program.

The file in which the item is stored is referenced by the last open or last file access preceding the call. See the examples below.

"item.id"

Item-id of the binary item.

"item.body"

Since a FlashBASIC program cannot handle segment marks (decimal 255) characters, a special encoding scheme is used. A segment mark is replaced by the special two character sequence 'DLE _', where DLE is a 'data link escape' character (decimal 16) and '_' the underscore character. The following special sequences are replaced at file time:

DLE DLE Replaced by one DLE.

DLE _ Replaced by one segment mark.

DLE XX Replaced by one or more segment marks, where 'XX' is the number of segment marks from 1 to 126, plus 128 (hex x'80'). For example, DLE x'81' is replaced by one segment mark, DLE x'A3' is replaced by x'A3'-x'80'=x'23'=35 segment marks.

"size"

Item body size, in decimal. The size does not include the special DLE escape sequences, but it should includes the number of segment marks after decompression. In other words, it is the size as if the item body was not encoded with DLE sequences. If the size is not large enough, additional overflow is allocated to hold the binary item, but this space is not contiguous in the Virtual space, which may have some impact on performance while accessing the binary item.

"code"

Return code:

>0 Number of frames allocated to hold the binary item.

-1 Not enough overflow.

-2 Illegal DLE sequence.

-3 Illegal file reference.

Syntax

code = oconv(size^item.id^item.body, "u007A")

Example

```
equ DLE to char(16)
x=char(01):DLE:"_"
open "dict","object" to fd
code=oconv(2:AM:'test':AM:x, "u007a")
Write a binary item 'test' containing the TWO characters (in hex)
x'01ff' in the dict of the file 'object'. 'code' returns 1 frame allocated.

equ DLE to char(16)
x=DLE:char(128+32):DLE:DLE
read dummy from fd,'' else dummy=''
code=oconv(33:AM:'test':AM:x, "u007a")
Write a binary item 'test' containing 32 consecutive segment marks, followed
by one DLE, in a file described by the file descriptor 'fd'. Note how the read
of a dummy (possibly non existent) item is used to 'reference' the file.
```

u0089

converts a hexadecimal number (in a string) to a binary number (in a string).

Syntax

binary.number = oconv(hex.number, 'u0089')

u0091

reads and writes D3 shell variables.

The first syntax form returns a list of all active user shell variables (not including their values).

The second syntax form returns the value of a specific variable.

The third syntax form sets the value of a specific variable. Note that unlike the retrieval forms, the user should avoid the "@$" prefix before the variable name. The "^" character indicates an attribute mark.

The fourth syntax form is used to delete a shell variable. As with the third syntax, the user should avoid the "@$" before the variable name. The "]" character indicates a value mark.

This user exit is used by the "set", "unset", and "penv" verbs.

Syntax

oconv("", 'u0091')

oconv("variable.name", "u0091")

oconv("variable.name^variable.value", "u0091")

oconv("variable.name]", "u0091")

u009d

encodes or decodes a string of characters.

ICONV causes the string to be "encoded", while OCONV causes an encoded string to be "decoded".

"seed"

is any non-null string of characters which will be used in the encryption algorithm.

"am"

indicates an attribute mark (char(254)).

Syntax

coded.string = iconv(seed:am:string, 'u009d')

uncoded.string = oconv(seed:am:string, 'u009d')

u009e

Replaces all occurrences of str1 with str2 in str3.

Syntax

xx = oconv("u":am:str1:am:str2:am:str3, "u9e")

Example

```
xx=oconv("u":am:"jan":am:"feb":am:"The month is jan","u9e")
The value of xx will be "The month is feb".
```

u00b9

returns a soundex value for a specified string.

Syntax

soundex = oconv(string, 'u00b9')

Example

```
string = 'DEMONS'
soundex = oconv(string, 'u00b9')
This returns "D552" to the variable, "soundex".
```

u00ba

performs BIOS "INT 10" calls to change the video screen attributes on the console.

"registers" is a dynamic array containing the values for the A, B, C, and D registers in the following form:

registers<1> = AX register (AH:AL)

registers<2> = BX register (BH:BL)

registers<3> = CX register (CH:CL)

registers<4> = DX register (DH:DL)

The functions available through the INT 10 call are:

00 = Determine or Set Video State

01 = Set Cursor Type

02 = Set Cursor Position

03 = Read Cursor Position

04 = Read Light Pen

05 = Select Active Page

06 = Scroll Page Up

07 = Scroll Page Down

08 = Read Character Attribute

09 = Write Character and Attribute

0A = Write Character

0B = Set Color Palette

0C = Write Dot

0D = Read Dot

0E = Write TTY

0F = Return Current Video State

10 = Set Palette Registers

11 = Character Generator Routine (EGA and above)

12 = Alternate Select (EGA and above)

13 = Write String

14 = Load LCD Character Font

15 = Return Physical Display Parameters

1A = Display Combination Code

1B = Functionality/State Information

1C = Save/Restore Video State

and others...

More information about IBM's INT-10 function can be found in the IBM Technical Reference Manuals on PC's

Syntax

result = iconv(registers, 'u00ba')

u014a

reads an absolute disk sector.

"read.command" is a dynamic array which contains the following information:

Read.command<1> = drive.number, <vm>, block.number

Drive.number may be:

1 = drive A:

2 = drive B:

3 = drive C:

4 = drive D:

<value mark>

Block.number is the logical disk block number to read.

read.command<2> = mode, <vm>, table1, <vm>, table2

Mode may be:

1 = convert ASCII to HEX

2 = transform and prefix character

3 = translate and delete character

4 = transform, prefix and drop Most Significant Bit

5 = transform, delete and drop Most Significant Bit

<value mark>

Table1 contains one or more hex characters to scan for on input. If there is more than one entry, then they are separated by <subvalue marks>.

<value mark>

Table2 contains one or more hex characters to replace the corresponding characters from table1. If there is more than one entry, then they are separated by <subvalue marks>.

"result" is a status code indicating:

'00' = successful,

'xx' = failed, where 'xx' is any non-zero code.

Syntax

result = oconv(read.command, 'u014a')

u014b

writes an absolute disk sector.

"write.command" is a dynamic array which contains the following information:

Write.command<1> = drive.number, <vm>, block.number

Drive.number may be:

1 = drive A:

2 = drive B:

3 = drive C:

4 = drive D:

<value mark>

Block.number is the logical disk block number to write.

Write.command<2> = mode

Mode may be:

1 = convert HEX to ASCII

Write.command<3> = 'u014b'

Write.command<4> = A sector's worth of data to write, in 1024 2-digit hex characters, which corresponds to a 512-byte sector.

"result" is a status code indicating:

'00' = successful,

'xx' = failed, where 'xx' is any non-zero code.

Syntax

result = oconv(write.command, 'u014b')

u017e

allows output of any character to the terminal or spooler.

Syntax

variable = oconv(string, 'u017e')

Example

```
printer on
x = oconv(char(255),'u017e')
 Output a character xFF to the spooler.
 x = oconv('abc','u017e')
 Output the string "abc" to the terminal.
```

u01b6

Converts a numeric expression to the number of units as defined by the set-units, set-dozens, set-decimal or set-thousands verbs.

Syntax

x = oconv(numeric.expression, "u01b6")

Example

```
:set-dozens
:u bp test
01 print oconv(24,"u01b6")
This program will print the value "2", as 24 = 2 dozen.
```

u0209

returns only alphabetic and numeric characters.

Syntax

result = oconv(var, 'u0209')

u0358

allows character input combined with an 'End Of Input' (EOI) character.

"length" is the maximum length of the input, not including the 'End Of Input' delimiter. "length" cannot be over 500. The 'EOI' character may be: x'0A', x'0E',x'17', x'0C', x'0D', x'1A', x'0B'

Editing Functions: The "left-arrow" key moves the cursor one position to the left while erasing the right side of the input field.

When the cursor reaches the left side of the input field, it remains on that first position.

When the cursor reaches the right side of input field, it remains on the last position and a 'bell' is sent. The user may then change the last character or move to the left.

Control characters other than the EOI are ignored.

Depending on the EOI character, 'variable' may be tested to take different branches. The EOI character is stored as a second attribute of the variable.

Syntax

variable = iconv(length, "u0358")

Example

```
var=iconv(lg,"U0358")
carvalid=seq(extract(var,2))
if carvalid=14 then ...
if carvalid=15 then ...
```

u1072

returns a dynamic array, sorted in ascending order.

Each element sorted must be separated by an attribute mark.

Syntax

sorted.dynamic.array = oconv(dynamic.array, 'u1072')

Example

```
equ am to char(254)
a = 'abcd': am: 'zxy': am: 'hijk': am: 1234: am: 789
b = oconv(a, 'u1072')
b will now contain:
1234 :am: 789 :am: abcd :am: hijk :am: zxy
```

u10b9

converts American currency punctuation ($1,000.00) to International punctuation (#1.000,00)

The "$" (dollar sign) is converted to "char". If "char" is not specified, then "#" (pound sign) is assumed.

The "," (commas) are converted to "." (periods).

The "." (decimal points) are converted to "," (commas).

Syntax

intl.format = oconv(american.format, 'u10b9{; char'})

u11b6

Converts a unit expression to a numeric value as defined by the set-units, set-dozens, set-decimal or set-thousands verbs.

Syntax

x = oconv(numeric.expression, "u11b6")

Example

```
:set-dozens
:u bp test
01 print oconv(5,"u11b6")
This program will print the value "60", as 60 = 5 dozen.
```

u1209

breaks a single line of text into multiple lines of text with each line being no more than "length" characters.

This function is similar to the "t" justification in AQL.

result = oconv(var, 'u1209,':length)

u1f

returns a string of digits indicating the boot status. If the "n" constant is not specified as the input string, then the boot status is cleared. If the first digit is one, then normal monitor shutdown was started. If the second digit is one, then normal monitor shutdown completed. If the third digit is one, then a full file restore has just completed.

Syntax

oconv('{n}', 'u1f')

u20b9

converts International currency punctuation (#1.000,00) to American punctuation ($1,000.00)

The "char" is converted to a "$" (dollar sign). If "char" is not specified, then "#" (pound sign) is assumed.

The "." (periods) are converted to "," (commas).

The "," (commas) are converted to "." (decimal points).

Syntax

american.format = oconv(intl.format, 'u20b9')

american.format = oconv(intl.format, 'u20b9;char')

u2117

returns information about the current process.

The information is returned in a dynamic array.

result<1> contains the PCB (Process Control Block) in hex, and the PIB status.

result<2> contains the process' Return Stack, separated by value marks.

result<3> contains:

0 If it was the last process,

1 If it is the current process,

2 If none of the above.

If the process is not logged on, then result<1> and <2> are empty.

Syntax

result = oconv(0, 'u2117')

u21a3

opens a file, or reads the next attribute from a file.

If used in an "iconv", it opens a file.

If the OPEN was successful, "result" contains a '1'.

If used in an "oconv", it reads the next attribute.

Syntax

result = iconv(file.reference, 'u21a3')

variable = oconv('', 'u21a3')

Example

```
result = iconv('bp fdisk', u21a3')
variable = oconv('', 'u21a3')
Opens the file "bp, fdisk", and reads the next attribute into "variable".
```

u21b6

Returns a numeric value representative of the current unit setting defined by the set-units, set-dozens, set-decimal or set-thousands verbs:

0 units

1 dozens

2 decimal

3 thousands

Syntax

x = oconv('', "u21b6")

Example

```
:set-dozens
:u bp test
01 print oconv(5,"u21b6")
This program will print the value "1".
```

u222d

Retrieves the CC pointer item information from binary items.

Attribute one is always "CC"

Attribute two is the start frame number.

Attribute three is the number of frames that the binary item resides in.

Attribute four is null (for R83 backward compatibility)

Attribute five is the time and date stamp in external format, separated by two spaces (for R83 backward compatibility)

A null result is returned on any error, such as the file could not be opened, the item did not exist or the item was not binary and there simply was no CC pointer information.

Syntax

result = oconv("dict file.name item.name", "u222d")

u28

returns the number of saves that have occured since the last full restore.

It is used by the "file stats" portion of the "file-save" program, and is passed into the AQL selection processor to eliminate files that were NOT saved on the most recent file-save. It is also used by the "list-file-stats" program.

Syntax

result = oconv('', 'u28')

Example

```
statno = oconv('','u28')
execute 'list stat-file with stat# "':statno:'" and with reel# # "1" '
```

u3060

encrypts a character string. This is the same user exit called by the "password" program.

The result is always an 8-character hex string.

Syntax

result = oconv(variable, 'u3060')

Example

```
a = "FRED"
b = oconv(a, 'u3060')
(b = "8F7D5A29")
a = "fred"
b = oconv(a, 'u3060')
(b = "D3575E89")
```

u3079

returns the PCB fid of the current process, or returns the port number of the current process.

If "iconv," returns the PCB (Primary Control Block) fid in hex of the current process.

If "oconv," returns the port number of the current process.

If another port is specified, then the port number must be in hex.

Syntax

pcb.fid = iconv(port.number, 'u3079')

port.number = oconv(pcb.fid, 'u3079')

Example

```
a = iconv(0, 'u3079')
"a" will be the PCB for port 0 (in hex).
a = oconv('02c0', 'u3079')
"a" will be Port 0 for a system with 704 ABS frames.
```

u307a

suspends processing until a specified time.

This user exit is identical to the FlashBASIC statement "sleep".

The "time.of.day" may be specified in military time format (hh:mm:ss). If no argument is indicated, the process sleeps for one second. "rqm" and "sleep" are identical.

Syntax

dummy = oconv(time.of.day, 'u307a')

Example

```
a = oconv('19:00', 'u307a')
Sleeps until 7:00pm.
```

u3090

aborts a FlashBASIC program.

Syntax

dummy = oconv('', 'u3090')

u313c

returns the diskette drive type and density selected.

The results are placed in a dynamic array.

result<1> contains the diskette density.

1 = 360k,

2 = 1.2mb,

3 = 720k,

4 = 1.44mb.

result<2> contains the diskette drive.

a = A:

b = B:

Syntax

result = oconv('', 'u313c')

Example

```
result = oconv('', 'u313c')
result = 2 :am: a
Diskette drive (A) is hi-density 1.2mb.
```

u352e

retrieves statistical information about the use of buffers by the D^3 monitor.

This user exits is for internal usage only.

Syntax

result = oconv('', 'u352e')

u3b

returns information about the port.

"line.status"

is the port information.

"code"

is a single character which describes the type of information requested, followed by the port.number to examine.

"a" returns the ABS base fid.

"c" returns the CPU units currently used.

"h" returns the number of pages printed.

"l" returns the current return stack.

"p" returns decimal PCB fid.

"s" returns the hex PIB status code.

Syntax

line.status = oconv(code, 'u3b')

Example

```
result = oconv( "c":46, 'u3b' )
print result
188
The current CPU charges for port 46 is 188 units.
```

u3f

converts an ASCII hex-format string to binary, and sends it to the printer.

Syntax

results = oconv(string.expression, 'u3f')

Example

```
stuff = "0D0A0000"
x = oconv(stuff,'u3f')
Sends the hex string "0D0A0000" to the printer (carriage return,
linefeed, and 2 nulls for linefeed delay.)
```

u4070

directs all subsequent output to the terminal.

This is the same as the FlashBASIC "printer off" command.

u407a

places a port to "sleep" for a specified number of seconds.

This is the same as the FlashBASIC "sleep" statement.

"dummy"

is a dummy variable. No data is returned.

"seconds"

is the number of seconds for the process to sleep.

Syntax

dummy = oconv(seconds, 'u407a')

u4117

returns the PCB (Process Control Block) in hex for the current process.

Syntax
pcb.fid = oconv('', 'u4117')

u4209

parses the input into discreet words.

Each word of of the output is separated by attribute marks.

Syntax
result = oconv(var, 'u4209')

u50bb

returns the port number, user-id and current account for this process, just like the TCL "who" command.

A specific portion of the who information may be requested. If the conversion value begins with the character "a", "u", or "p", then only the account, user, or pib will be returned respectively.

The data for another pib may be obtained by putting a "%" character followed by the pib number in the conversion value.

Syntax
result = oconv('{[a|u|p]}{%pib}', 'u50bb')

Example
```
a = oconv('', 'u50bb')
 In D3, "a" contains "0 bob dm" if the current user-id is "bob" and
the current account name is "dm".
a = oconv('a','u50bb')
"a" contains the current account.
a = oconv('u%3','u50bb')
"a" contains the user ID for pib 3.
```

u5117

returns the PCB (Process Control Block) in hex for port 0.

Syntax
pcb.fid = oconv('', 'u5117')

u60ba

flushes all memory to disk, and shuts down the D^3 System.

Syntax
result = oconv('', 'u60ba')

u60bb

returns the user-id and account name of the current process.

Syntax
result = oconv('', 'u60bb')

Example
```
a = oconv('', 'u60bb')
In D3, "a" contains "bob dm" if the current user-id is "bob" and the
current account name is "dm".
```

u6193

removes all but the last three characters from the data passed from FlashBASIC.

Syntax

results = oconv(string.expression, 'u6193')

Example

```
string = "abcdefghixyz"
string = oconv(string,'u6193')
In this example, the results of the conversion call would be "xyz"
```

u63

sets/clears privilege bits.

The bit to be set or cleared is indicated by 'value' which must be one of the following:

0 Entire table

-1 Inhibit (like break on/off, but works like a bit)

-2 TCL restart

-3 Break/End restart

if iconv is used then the value is set. If oconv is used, then the value is cleared. The 'result' is the value of the bit before it was changed except when used with the 'value' 0 in which case, the result is undefined.

This user exit is similar to the bitset and bitreset statements on some licensee platforms.

Syntax

result = oconv(value, 'u63')

result = iconv(value, 'u63')

u7000

returns the size of an item.

Syntax

item.size = oconv('file.name item-id', 'u7000')

u70ba

forces the system to restart.

Syntax

dummy = oconv('', 'u70ba')

u7193

converts the American date format (mm/dd/yy) to international format (dd/mm/yy).

Syntax

results = oconv(string.expression, 'u7193')

Example

```
date = "11/06/92"
intl.date = oconv(date,'u7193')
At the end of this operation, "intl.date" contains "06/11/92"
```

u8194

returns the number of items selected by the most previous select.

Syntax
result = oconv(var, 'u819f')

u90

returns the file-of-files number for the file most previously referenced in the BASIC program.

This user exit is used by the "get-fof" command.

Syntax
oconv("", 'u90')

u90e3

flips the parity bit on the input.

This is useful for reading or writing some licencee compatible tapes.

Syntax
result = oconv(var, 'u90e3')

u9116

logs on a given port.

This is the same as "logon".

Syntax
dummy = oconv('port.number, user.id {,user.password}, account.name {,account.password}', 'u9116')

Example
```
a = oconv( '3, SYSPROG', 'u9116')
Logs on port 3 to SYSPROG
```

u92

follows the links of a frame.

The format of "string" is as follows:

decimal.fid {, loop.count }

If loop.count is not specified, then the user exit will traverse links until it encounters a 0 value.

The iconv version scans forwards, while the oconv version scans backwards.

Syntax
iconv(string, 'u92')

oconv(string, 'u92')

ua116

logs off a specified port.

Syntax
dummy = oconv('port.number', 'ua116')

Example
```
a = oconv('6', 'ua116')
Logs off port 6.
```

ub070

returns the absolute value of a variable, stripping the leading "-", if any.

Syntax

result = oconv(var, 'ub070')

ub0ba

"peeks" (reads) one byte of memory.

Syntax

byte = oconv(memory.address, 'ub0ba')

Example

```
memory.address<1> = '0000'   ;* This is the memory segment.
memory.address<2> = '09e6'   ;* This is the offset.
memory.address<3> = '00'     ;* This is a dummy.
byte = oconv(memory.address, 'ub0ba')
"byte" will contain the contents of memory location 0000:09e6.
Addresses worth viewing/changing:
0x0816     : CPU type (1=8086, 2=80286, 3=80386)
0x0817     : # of user system
0x0830-0831: # of disk cylinders
0x0832     : # of disk heads
0x0833     : # of disk sectors
0x085B     : Maxfid
0x0B69     : # of serial ports
0x0BE4     : # of parallel printers
0x09E6     : Various info.
bit0 = Overlapping disk and tape I/O
bit1 = Forced tape recognition
bit2 = Spinel card
0x09E8     : Wait-And-Post flag (00=disabled, FF=enabled)
0x0BEE-0BEF: Tape I/O Address (Low Byte, High Byte)
0x9B40-9B41: Points to SIO.RING (IRQ4)
0x9B42-9B43: Points to SIO.RING (IRQ3)
0x9B44-9B45: IRQ4 Serial Card
0x9B46-9B47: IRQ3 Serial Card
Serial Card Addresses:
0x9257 = Arnet card
0xB657 = No serial cards
0xB957 = IBM COM1:-type card
0xBC57 = Digigraphics / Monolith card
0xBF57 = AST card
0xC257 = Arnet card
0xC557 = ALR card
0xC857 = Digiboard card
0xCB57 = AMI card
0xCE57 = Hostess card
0xD157 = Spinel card
0x9B80-9B2F: SIO.RING UART starting address
(9B00-9B01 should be FF FF)
0x9BB0-9B2F: PIB.RING starting addresses
```

uc0ba

writes ("pokes") a value into a memory address.

Syntax

dummy = oconv(memory.location, 'uc0ba')

Example

```
memory.location<1> = '0000'   ;* This is the memory segment
memory.location<2> = '09e6'   ;* This is the offset
memory.location<3> = '01'     ;* This is the value to write
(poke)
dummy = oconv(memory.location, 'uc0ba')
The memory location 0000:09e6 now contains the value 0x0001.
```

ud0ba

reads a character byte from a serial port.

"char"

is the byte that is returned.

"port.number"

is a dynamic array:

port.number<1>

4-digit hexadecimal port address.

port.number<2>

a dummy value.

Syntax

char = oconv(port.number, 'ud0ba')

ue0ba

writes a character byte out to a serial port.

"port.number"

is a dynamic array:

port.number<1>

4-digit hexadecimal port address.

port.number<2>

character byte to transmit.

Syntax

dummy = oconv(port.number, 'ue0ba')

uf070

starts and stops multi-user mode, and returns information about the number of serial ports, pibs, phantom pibs, licensed users, users logged-on, and the maximum number of users that can logon.

Setting the system in single user mode simply prevents more users from being able to log on. Users currently logged on are not affected.

"function"

"S" to start single user mode,

"M" to start multi-user mode,

"" to only retrieve the current settings.

The returned string consists of 6 values in the following order:

1 SIO The number of serial ports, and an initial character ("s" or "m")) indicating if the system is currently in single or multi-user mode. (The number of serial ports is not returned on Unix-based systems.)

2 PIBS The number of pibs available for serial devices, in hex.

3 PHANTOMS The number of pibs available for phantom jobs, in hex.

4 ULIC The number of licensed users, in hex.

5 ULOG The number of users currently logged on, in hex.

6 UCEIL The maximum number of concurrent users permitted, in hex.

Syntax

var = oconv(function, 'uf070')

Example

```
x = oconv("", "UF070")
print "The current number of users is: ":xtd(x<5>)
```

UNLOCK

resets an execution lock, in the range 0 to 63, previously set with a "lock" statement.

If no "lock.number" is specified, all locks set by the program are unlocked.

Syntax

UNLOCK

UNLOCK lock.number.expression

Example

```
again: lock 12 else rqm; goto again
read rec from file,id else print id:" not found"; rec = "X"
write rec on file,id
unlock 12
This conditionally sets execution lock 12 just before a segment of
code which performs file retrieval and update. When complete, lock 12
is cleared by the "unlock" statement.
```

UNTIL

specifies a termination condition for a "for .. next" or "loop" constructs.

user exits, FlashBASIC

allows calling direct system code from a FlashBASIC oconv/iconv command. Many of the functions performed by user exits may be achieved by using the "system" function or the "execute" statement with the appropriate TCL command or FlashBASIC statement.

WAITING

see "get".

WEOF

writes an "end of file" mark to the currently attached magnetic media.

The "then" clause is executed if the operation was successful. If the operation was unsuccessful, the "else" or "onerr" clause is executed.

The "onerr" clause may be used to check for tape error conditions by interrogating the "system(0)" function. Either "else" or "onerr" may be specified, but not both.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Syntax

WEOF variable [THEN | ELSE | ONERR statement.block]

Example

```
In this "subroutine", a series of tapes files are written using items in a
previously formatted disk file. The items represent tape blocks and the ids
are numbered 1 though M, where M is the block count. When a null item is
encountered, the logic assumes that an "eof" is to be written to tape. Two
"eof"s indicate an end-of-data condition.
subroutine outtape
execute 't-att ':blocksize
if system(0) = "1" then ; * tape not attached
  err=system(0)
  return
end
ctr=1; eot=0 ; err=0
loop
  read block from tape.hold.file,ctr then
    if block='' then  ;* a null block in the file
                      ;* write an eof.
      weof else err=system(0)
    end else
      writet block else err=system(0)
    end
  end else eot=1
until eot or err do
 ctr=ctr+1
repeat
if eot then            ;* at end-of-tape
  weof then            ; * write 2 eof's
    weof else err=system(0)
  end else err=system(0)
end
return
```

WHILE

specifies a continuation condition in a "for...next" or "loop" construct.

WRITE

writes the item from the dynamic array specified in "dynamic.array.variable" into the specified file, using the item-id specified in the "id.expression".

If the file.variable is not specified, the default file.variable is used.

The "writeu" statement writes a dynamic array into the specified file.variable and keeps the update lock set. "writeu" additionally keeps items locked that were locked by a previous "readu", "readvu", or "matreadu" statement.

The "writex", "matwritex" and "matwritexu" statements all have the property of waiting until the actual disk update takes place before continuing execution of the program. They are used for "critical" write-through, such as error-logging.

An optional "on.error.clause" may be specified. This consists of the words "on error" followed by a statement list. The clause is taken if the update fails because the data source is unavailable (as can be the case if the file is remote), or if a callx correlative applied to the file fails because of an "inputerr" statement.

Syntax

WRITE{X}{U} dynamic.array.variable ON {file.variable,} id.expression {on.error.clause}

Example

```
write item on customer.file,item.id
This statement unconditionally writes the contents of the array.variable
"item" on the "customer" file with a specified "item.id". If the item doesn't
exist, it adds it. If the item exists, it overwrites it.
readu item from customer.file,item.id else stop
...
begin case
case action = 'fs'
writeu item on customer.file,item.id
case action = 'fi'
write item on customer.file,item.id
return
case action = 'ex'
release customer.file,item.id
return
end case
...
Since a "writeu" does not clear the item lock, a "release" or an
explicit "write" is required to clear the lock before continuing.
```

WRITET

writes a tape record to the attached magnetic media from the specified variable.

If the size of the expression is less than the attached tape block size, it is padded to fill the block size. If it is greater than the block size, it is truncated.

The "then" clause is executed if the "writet" is successful. The "else" or "onerr" clause is executed if the tape unit is not attached or the string value of expression is an empty string (").

Either "else" or "onerr" may be specified, but not both. In addition, the "onerr" clause may be used to check for tape error conditions by interrogating the "system(0)" function.

See the "then/else construct" for an explanation on the use of "then" and "else" clauses in statements that allow or require them.

Note: To ensure the block is written write a file mark. See "weof" statement. The "weof" will flush the buffer to tape and write an end-of-file mark on the tape.

Syntax

WRITET expression [THEN | ELSE | ONERR statement.block]

Example

```
record = name 'l#20' : address 'l#15'
writet record else stop
```

WRITEU

see "write" or "matwrite".

WRITEV

writes the value of an expression into the attribute designated in the attribute expression parameter, using the item-id specified in the item-id expression parameter.

This statement causes a "read" to occur prior to the "writev".

If the file.variable parameter is omitted, the default file.variable is used.

"-1" may be used as the value of the "ac.expression". This appends a new attribute to the end of the item.

Syntax

WRITEV expression ON {file.variable ,} id.expression, ac.expression

Example

```
writev time() on control.file,'control.list',3
```

WRITEVU

writes the value of an expression into the attribute designated in the attribute expression parameter, using the item-id specified in the item-id expression parameter, and keeps the update lock set.

If the file.variable parameter is omitted, the default file.variable is used. This statement causes a "read" to occur prior to the "writev". The "writevu" statement is identical to the "writev" statement, except that the item remains locked.

Syntax

WRITEVU expression ON {file.variable, } id.expression ,ac.expression

XTD

converts an expression from its hexadecimal format into its equivalent decimal format.

Syntax

XTD(string.expression)

Example

```
number = xtd("a")
This assigns "10" to "number".
```

[

substring extraction, assignment, and field store.

Example

```
if response[1,1] = "y" then printer on
```

\

marks the beginning or ending of a literal string in FlashBASIC or is the "remainder" operator.

Generally, it does not matter which types of quotes are used on literals (strings). Some instructions, such as "heading" and "footing" do impose certain restrictions on their use.

Syntax

\literal string\

num.expression \ num.expression

Example

```
1)  if answer = \quit\ then stop
The backslash can be used anywhere a single-quote or double-quote is used.
2) execute \list md heading "'lc'md listing'l'\
Typically, the backslash is used when executing a TCL string expression in
FlashBASIC. The AQL "heading" statement requires the use of both single-
quotes, as options, and double-quotes.
```

****

"\" calculates the remainder of dividing the dividend by the divisor (otherwise known as the modulo).

There is no limitation on the numeric value of the dividend or divisor.

Syntax

dividend \ divisor

Example

```
x = 21
y = 8
z = x \ y
print z

In this example, "5" (the remainder of 21/8) would be printed.
```

\=

is an assignment operator which divides a variable by a given numeric expression and assigns the remainder (modulo) to the variable.

Syntax

variable \= num.expression

Example

```
x \= 2
This divides the current value of "x" by "2" and assigns the remainder to x".
This is equivalent to: x = x \ 2
```

]

Substring, or text string extraction.

^

an alternate means of indicating an exponentiation operation on a specific numeric expression.

Example

```
1)  a = x ^ 3
This example calculates the value of "a" as the cube of the value of "x".
2)  a = pwr(x,3)
This example is identical in output to the previous example.
```

FlashBASIC Debugger

facilitates the debugging of new FlashBASIC programs and the maintenance of existing FlashBASIC programs.

The FlashBASIC debugger has the following general capabilities:

1) Step through execution of program in single or multiple steps. (see the "e" command).

2) Transfer to a specified line number. (see the "g" command)

3) "Break" execution (temporarily halt) at given line numbers or when specified conditions have been satisfied. (see the "b" command)

4) Display or change any variables, including dimensioned variables. (see the "/" command).

5) "Trace" variables. (see the "t" command)

6) Enter the system debugger. (see the "de" and "debug" commands)

7) Direct output to either terminal or printer. (see the "lp" command)

8) Display and/or pop GOSUB stack. (see the "r" command)

9) Display source code lines. (see the "l" and "c" commands)

Note: sys2 privileges are required to use the FlashBASIC debugger.

When the FlashBASIC debugger is entered, it indicates the source code line number to be executed next and prompts for commands with an asterisk (*).

Symbol Table:

When a FlashBASIC program is compiled, a symbol table is generated, unless the "s" (suppress table) option has been used with the "compile" or "basic" verb. The symbol table is stored as part of the object code and is used by the FlashBASIC debugger to reference symbolic variables.

If a program calls an external subroutine, and the FlashBASIC debugger has been entered previously, a complete symbol table is set up for the external subroutine.

Entering the debugger: There are several ways to enter the debugger. Some of them are voluntary. Among the voluntary methods are:

1) Pressing the "break" key while a FlashBASIC program is executing.

2) Running the program with a "d" option:

run bp enter.customer (d)<return>

If the program is cataloged, a "d" option may follow the program name at TCL:

enter.customer (d)<return>

3) Placing a "debug" statement within the program at the point where the debugger is needed.

4) Running the program with a "e" option, which "breaks" at any "non-fatal" warning message:

run bp enter.customer (e)<return>

If the program is cataloged, an "e" option may follow the program name at TCL:

enter.customer (e)<return>

The other ways of entering the debugger are, of course, involuntarily. This occurs if a runtime error is encountered (unless the "a" (abort) option of the "run" verb is selected), or if an "abort" statement is executed in the program.

The "*" (asterisk) character is used as the FlashBASIC Debugger prompt character. When it appears in the leftmost column of the terminal display screen, this indicates that the debugger is ready to accept any legal command. The number of "*" characters indicates the current level.

"Breakpoints" may be set in the debugger. A "breakpoint" is a conditional expression constructed with a symbol reference (one of the variables from the program), an operator (like "=" or "#"), and a specific value. For instance: item-id#"" sets up a conditional to break into the debugger when the variable "item-id" is non-null. (see the "b" command)

Break points set up for a subroutine are independent from break points set up in the main program or other subroutines; however, the execution counters "e" and "n" are global. That is, the break point counters count both main program and subroutine break points in the order they are encountered. The use of multiple symbol tables allows the programmer to set up different break points and/or variable traces for different subroutines. (see the "e" and "n" commands)

Operators are used to perform relational comparisons and are used with the "b" command for setting breakpoint conditions. The valid operators are as follows:

=	equal to.
>	greater than.
<	less than.
>=	greater than or equal to.
<=	less than or equal to.
#	not equal to.

Referencing variables : once in the debugger, the current value of any variable may be interrogated (and optionally changed) using the following forms:

*/variable

This accesses a plain variable and displays its current value.

*/dim.array.variable(subscript)

This access a single-dimensioned array variable. With the subscript designated, only that element is displayed.

*/dim.array.variable<return>

If a dimensioned array is referenced without a subsrcipt, each element is displayed. Another <break> while in this mode exits this display without stepping through all the elements.

*/dim.array.variable(n,n)

This accesses a specific location of a two-dimensional array. See the above form for array references without subscripts.

/

This displays the current assignment of every variable in the program.

In each of the above cases (except the "*" form), the operator is given the opportunity to change the requested value. If a <return> is entered at the "=" prompt, the symbol remains unchanged. Otherwise, any value entered is placed into the symbol.

Zone output specification : *[{left.limit,right.limit}] or *[

The "zone" command sets left and right output zone limits of debugger display. Both "limit" arguments must be numeric, and it is advised that the second argument be larger than the first. The "["command, followed by a <return>, removes zone limits.

#

displays the item-id of the error message which caused the most recent abort into the FlashBASIC debugger.

$

displays the current program name and current line number, and verifies the object code.

The object code is verified by performing a checksum verification. This is used to determine if the object code has been corrupted since the last time it was compiled.

The "$" and "?" commands are identical.

Example

```
:run bp conv (d<return>
*E1           <- Displayed by debugger.
*$<return>    <- This is entered.
*$ conv L 1 object verifies.
```

$!

displays the current call stack with parameter names and values being shown. If the symbol table is not present, the internal names are shown.

Syntax

{Flash.routine.name:}$!

/

reference, and optionally alter the contents of, variables within the runtime code being executed.

Syntax

{Flash.routine.name:}/variable

{Flash.routine.name:}/dim.array.variable

{Flash.routine.name:}/dim.array.variable(vector.number)

{Flash.routine.name:}/dim.array.variable(row.num,col.num)

/*

Example

```
*/variable
This accesses a plain variable and displays its current value.

*/dim.array.variable
This displays each element of a dimensioned array. If a dimensioned array is
referenced without a subscript, each element is displayed, one at a time. To
exit this display without stepping through all the locations, hit the <break>
key, or enter any non-null value. The non-null value will not replace any
location displayed. The contents of the array can NOT be altered.

*/dim.array.variable(subscript)
This accesses a single-dimensioned array variable. With the subscript
designated, only that element is displayed. This can also access a two-
dimensional array and assumes the second subscript is 1.
```

```
*/dim.array.variable(row.num,col.num)
This accesses a specific location of a two-dimensional array.

*/*
This displays the current assignment of every variable in the program.
Typically, programs have a lot of variables. This function does NOT paginate,
which results in having the variables fly by on the screen and not being very
useful, unless the reader happens to be a speed reader. The <break> key will
interrupt this "runaway" variable display and return to the debugger prompt
character.

In each of the above cases (except the "/dim.array.variable" form), the
operator is given the opportunity to change the requested value. If a <return>
is entered at the "=" prompt, the symbol remains unchanged. Otherwise, any
value entered is placed into the variable.
```

?

Displays the current program name an line number

Example

```
:run bp conv (d<return>
*E1
*?<return>
*? conv L 1 object verifies.

"conv" is the name of the program being executed. "L 1" indicates that line
one of the program is about to be executed.

If the "conv" program had been "flashed" (meaning, compiled with the "o"
option), the message produced by "?" would appear as:

*? conv L 1 FlashBASIC object verifies.
```

?!

used in the FlashBASIC debugger to display the current call stack with current parameter values being shown.

Syntax

{Flash.routine.name:}?!

b

establishes a "breakpoint" condition in the break point table, which causes the program to enter the FlashBASIC debugger when the condition defined in the breakpoint expression is met.

The valid comparative operators are >,<, and =.

A "+" (plus) character is displayed for each breakpoint successfully entered into the table until the table is full. In FlashBASIC, there is no limit to the number of table entries.

Note that the spaces between the arguments in the syntax are there for readability; they are NOT allowed when composing breakpoints.

Once a breakpoint is placed in the table, the debugger is entered each time the breakpoint is met.

Note that the syntax form, "bvariable", is only supported in D3 releases with FlashBASIC. It breaks into the debugger each time the specified variable changes.

Syntax

{Flash.routine.name:}b

{Flash.routine.name:}b variable operator variable

{Flash.routine.name:}b variable operator numeric.literal

{Flash.routine.name:}b variable operator "literal.string"

{Flash.routine.name:}b variable

{Flash.routine.name:}b $operator line.number

Example

```
run bp fred (d
*E1
*b$=5

This sets a break point on line 5.  The break point remains until execution is
complete, or it is manually removed.
*sally:b
When running FlashBASIC, this command will set a breakpoint that will cause
execution to pause every time the subroutine "sally" is entered.
```

c{!}

toggles on or off the display of the current source line when stepping through the program with <ctrl>+j. When this function is on, the line is displayed AFTER it has been executed.

Syntax

c

d

displays the contents of the "breakpoint" and "trace" table.

"breakpoints" are established using the "b" command and cause the debugger to be invoked on a specific condition.

"trace" table entries are established with the "t" command and their current values are automatically displayed each time the debugger is entered.

Syntax

{Flash.routine.name:}d

Example

```
The following table shows that there are two variables being traced and one
break point active.

*d
T1 incount
T2 client.number
T3
T4
T5
T6
B1 b$=250
B2
B3
B4
```

de

invokes the system (virtual) debugger from the FlashBASIC debugger.

Issuing a "g" command from the virtual debugger returns control to the FlashBASIC debugger.

The "de" command is identical to the "debug" command at TCL.

Example

```
*de
!g
*
```

debug

temporarily exits the FlashBASIC debugger and invokes the system (virtual) debugger.

Issuing a "g" command from the virtual debugger returns control to the FlashBASIC debugger.

The "debug" command is identical to the "de" and "debug" (TCL) command.

Example

```
<break>
*I3
***debug<return>
I bd.d.064   => 19 b$bd.com
!!!g<return>   <- Returns to FlashBASIC debugger
***g<return>   <- Resumes program execution
```

down

is used in the FlashBASIC debugger to move down the call stack.

e

indicates the number of instructions to execute prior to returning to the FlashBASIC debugger.

The "e" command, followed by a <return>, disables the previous iteration counter setting and executes "continuously".

Syntax

e{number.lines}

Example

```
*e10<return>
Sets execution counter.
*g<return>
Resumes program execution.
This invokes the debugger after executing 10 lines.

*e<return>
Sets "continuous" execution.
*g<return>
Resumes program execution.
This cancels any previous "e" commands and makes the program run without
execution interruption. The "g" command must follow after the "e" command to
resume the program.
```

edit

invokes the Update processor for editing the current program from within the FlashBASIC debugger. It automatically positions the cursor at the current execution line number.

end

terminates the run-time portion of a FlashBASIC program, exits the debugger, and returns control to TCL.

g

resumes execution of a program after it has been interrupted by the execution of a FlashBASIC debug statement or by typing the break key or esc-level key.

The optional line number is only valid for programs compiled by the native compiler (unflashed) with eol (end-of-line) characters. The "g" command will not resume execution after an error condition.

Syntax

g{line.number}

help

may be issued from the FlashBASIC debugger to produce a brief listing of the available debugger commands.

j

has the same effect as the FlashBASIC debugger "g" command except that the <ctrl>j only requires one keystroke.

If the FlashBASIC debugger "c" option is enabled, <ctrl>+j also displays the FlashBASIC instruction after it is executed.

Syntax

<ctrl>+j

k

"kills" a breakpoint previously set with the "b" command, and removes the entry from the breakpoint table.

The "k" command, followed by <return>, removes all breakpoint entries from the table.

Syntax

{Flash.routine.name:}k{breakpoint.number}

Example

```
*k1
This kills the first breakpoint in the table.

*k
This kills all breakpoints.
```

l

lists the current source line, or a specified range of lines, from the corresponding source program being executed.

"l*" displays the entire program.

"l" followed by <return> displays the current program source line.

Syntax

{Flash.routine.name:}?l{starting.line-ending.line}

{Flash.routine.name:}?l{line.number}

{Flash.routine.name:}?l{*}

{Flash.routine.name:}?l

Example

```
*l10-17<return>
This lists lines 10 through 17.

*l5<return>
This lists line 5.

*l*<return>
This lists the whole program.
```

lp

toggles the line printer bit status, either directing debugger output to the terminal screen or to the Spooler.

This is the equivalent to issuing a "printer on" statement within the program.

n

Similar to the FlashBASIC debugger "g" command except that if single-stepping is active, the <ctrl>+n command will step "over" any subroutine calls.

Syntax

<ctrl>+n

n

ignores any debugger breakpoint conditions for a specific number of occurrences.

If "number.times" is not specified, the program executes through one breakpoint, then stops.

Syntax

n{numbertimes}

Example

```
*n4
This skips four breakpoint conditions before re-entering debugger.
```

off

stops processing at all levels on the current port and returns control to the initial "logon" prompt.

p

toggles disabling output display; debugger output is still displayed.

pc

closes the current Spooler entry and releases the print job to the Spooler.

r

"pops" the top return stack address of the local subroutine from the stack, causing the next "return" statement to return to the address revealed by the "pop".

Syntax

{Flash.routine.name:} r

return

returns from the current subroutine in the FlashBASIC debugger. All code from the current debug point to the end of the routine is executed normally.

s

displays the contents of the subroutine stack.

Syntax

{Flash.routine.name:}s{!}

t

places a given variable into the trace table, causing the debugger to display the specified data element, along with the contents of the "break" and "trace" tables, on each break.

A "+" (plus) character is displayed for each variable successfully entered into the table, until the table is full. The maximum number of trace table entries is six, except in FlashBASIC, where the number is unlimited.

The "t" command, followed by a <return>, toggles the trace function on or off.

Syntax

{Flash.routine.name:} t {variable}

Example

```
*tcounter<return>
This sets a trace on the variable called "counter".
```

u

removes from the trace table a given variable, or all variables, previously specified with the "t" command.

A "-" (minus) character is displayed for each trace table entry successfully removed from the table.

The "u" command, followed by a <return>, clears all trace table entries.

Syntax

{Flash.routine.name:}u{variable}

Example

```
*uclient.number
This removes the trace on variable, "client.number" from the table.

*u
This removes all traced variables from the table.
```

up

used in the FlashBASIC debugger to move up the call stack.

v

verifies the current FlashBASIC object code by performing a checksum verification.

This determines if the object code has been corrupted since the last time it was compiled.

Example

```
run bp conv (d
*E1
*v
*$ object verifies.
```

[

sets the left and right output zone limits of debugger display.

Both "limit" arguments must be numeric, and it is advised that the second argument be larger than the first. The "[" command, followed by a <return>, removes zone limits.

Syntax

[{left.limit,right.limit]}

Example

```
*[1,30]
*[
```

****

used by the FlashBASIC debugger to display the current value of a variable or integer expression.

Syntax

{Flash.routine.name:}\variable.expression

{Flash.routine.name:}\integer.expression

Example

```
*\a(i,j)
*\a(i)+b(i)
```

]

See "[".

C Functions

Introduction

outlines how to access the D3 environment from the "C" programming language.

The functionality of the D3 environment is available from the C programming language. This is made available through a set of functions and macros defined in the include file "CPuser.h".

Library Format

Functions within the library have the following general structure:

{rc=}_CP_xxx({arg0{,arg1{,...}}});

where:

_CP_xxx is a general pattern for the name. It is always prefixed with _CP_ to avoid naming conflicts with standard libraries and is suffixed with an entirely lower-case tag like _CP_execute. Macros are provided using the same model, but they use uppercase tags like _CP_SLEN.

arg0 - argi are arguments.

rc is the result code. Almost all functions return an integer result. This result is -1 if an error occurs, and the global variable _CP_errno is set to one of the following error codes:

PE_NFILE Not enough memory to open more files

PE_ACCESS Cannot access file

PE_TRUNC String truncated due to lack of memory

PE_NOSPACE Could not allocate workspace

PE_LOCK Item is locked

PE_INVAL Invalid call

PE_NONUM Not a number

PE_BADF File not opened

PE_MISSING File not found

PE_CONV Conversion error

PE_BADATTR Bad attribute

PE_TMOUT Input time out

PE_TAPE Tape error

PE_PROC Proc read error

PE_EOF End of file/item

PE_CALLMAIN Tried to call a main as a subroutine

PE_NOTROOT Bad root variable

PE_ILL_KEY Bad key operator

PE_LONG_STR Too long a string for heading

PE_LEVEL Debugger entered

PE_DEBUG Debugger entered

PE_BAD_PARAMS Too many parameters on a call

PE_LOAD_ERR Unable to load Flash subroutine

PE_END_LIST No more items on select

PE_GETSEND Bad pib on get/send

PE_MISCERR Miscellaneous logon error

PE_INIT Virtual machine not booted or attached

PE_BADUSER Bad user ID or user password

PE_BADMD Bad MD or MD password

General Usage

To use this library, the user makes a call to _CP_logon() at the beginning of the C program to connect to the D3 virtual machine. This call goes through the normal D3 logon activities, but suppress all user prompts and messages. Note that neither the user nor the MD macro will be executed. After logging on, the user has full access to the D3 database using C calls which provide functionality previously available only through FlashBASIC. When the C program completes, the user must issue a _CP_logoff() call to terminate the D3 connection.

String Handling

The standard C string type is not sufficient for handling the dynamic nature of D3 strings, so a hybrid data type has been created called CPSTR. This data type provides allocation, deallocation, and resizing of strings at a much higher performance level than that obtained using malloc(), and free() with C strings. The internal structure of a CPSTR is reserved and SHOULD NOT BE ACCESSED DIRECTLY. Failure to abide by this rule will DEFINITELY GUARANTEE INCOMPATIBILITY WITH FUTURE RELEASES. The structure of a CPSTR is as follows:

Header Internal use only - DO NOT MODIFY

String - Character array containing the actual string

Terminator - A single character used for termination

Footer Internal use only - DO NOT MODIFY

The following macros are provided to access all desired portions of a CPSTR structure.

_CP_SLEN(s) returns the current length of a string. This length does not include any terminating character like a segment mark or a null character.

_CP_SADDR(s) returns a (char *) to the actual string data which may or may not be terminated by a 0x00. Note that this (char *) MUST NOT be passed to the free() C function or to any C function which may try to use free() on the (char *).

Allocation of CPSTR strings is accomplished via special allocation routines. CPSTR strings MUST NOT be allocated with the C function malloc(). The allocation routines provided set necessary internal flags, and they are much more efficient than malloc. Note that these routines create dynamic structures which must be deallocated with the routines listed below.

_CP_str_alloc(l) returns a CPSTR of length l. The actual string data portion of the string is undefined.

_CP_mkstrl(char *s, int l) Returns a CPSTR from a C string s, of length l. The string portion of the CPSTR will contain a copy of the C string's data.

_CP_mkstr(char *s) Returns a CPSTR from a C string s terminated by a 0x00. The string portion of the CPSTR will contain a copy of the C string's data. This function is less efficient than _CP_mkstrl as it must scan the C string to determine its length.

_CP_str_realloc(CPSTR **s, int l) Resizes the CPSTR s.

_CP_str_free(CPSTR ** s) Frees the desired string s.

Note that when the _CP_logoff call is executed, all CPSTR's remain valid, but they may not be passed to any of the _CP_ routines. All CPSTR's are properly cleaned up when the C program exits.

The user must be careful with termination. Any _CP_ routine may modify the termination character. Therefore, the user must use the _CP_TERM() macro to terminate CPSTR's with a 0x00 if it is necessary to pass the string portion of it to a standard Unix routine.

Note that it is necessary to free strings only when the user has finished with them completely. All resizing of strings is accomplished automatically by the _CP_routines. For example, if a process repeatedly reads data into a buffer via the _CP_read call, this buffer will be automatically resized to fit the new data within the _CP_read call at every read.

Environment Setting

To make it easier to handle the differences between the standard behavior of Unix programs and the standard behavior of D3 programs, two calls are provided that set the "environment" to either D3 or Unix.

_CP_pick_env() sets the application to a standard D3 environment. When using this setting, all terminal input and output must be done via D3 routines ONLY.

_CP_unix_env() sets the application to a standard "Unix" environment. With this setting, all terminal input and output must be done directly from "C". The program must not "execute" or "call" any D3 routines which may "crt" or "print" something through D3.

The original logon to D3 calls _CP_pick_env(). After that, the user may use these routines to switch environments as often as desired, but the environment functions are fairly expensive so they should only be used when switching environments is absolutely necessary. To ensure the correct functionality of the Unix shell, the system automatically calls _CP_unix_env() when the user logs off the of the D3 virtual machine.

Break Key Handling

When logged into D3, the default action for the break key is set to push a level or drop into the debugger as is standard in D3. This behavior may be changed however by using the following calls:

_CP_unix_break() causes the break key to force a logoff from D3 and terminate the C program with an exit() call. This simulates the standard behavior of the break key in a Unix application.

_CP_pick_break() causes the break key to behave in the D3 manner.

Handling General D3 Interrupts

D3 interrupts, besides breaks, such as messages, may not be masked, but they may be detected (after the occurence) by interrogating the _CP_interrupt variable. This integer location contains the same information available in BASIC from the system(37) function. However, examination of _CP_interrupt is much more efficient when programming in C.

Custom Signal Processing

The D3 libraries make extensive use of signal processing. Because of this, the practice of reprogramming signals within a C application using D3 C routines is strongly discouraged. However, if it is absolutely necessary, there are several rules which must be followed.

When calling _CP_logon, all signals are re-routed to D3 signal handlers.

When calling _CP_logoff, all signals are re-routed to Unix defaults. Note that they are not necessarily reset to the values they held before the _CP_logon call.

The signal SIGUSR2 must never be reprogrammed as it is used by D3 for logging off and sending messages.

Because interrupts may occur within critical code, the user may not make calls to _CP_ functions within signal handlers.

Example

The following program prints the name of all users in the users file.

```
#include "CPuser.h"
main()
{
  CPSTR * machine = _CP_mkstrl("pick0",5); /* default */
  CPSTR * user = _CP_mkstrl("dm",2);
  CPSTR * md = _CP_mkstrl("dm",2);
  CPSTR * filename = _CP_mkstrl("users",5);
  int     fd = -1;
  int     sl = -1; /* initialize select list */
  CPSTR * item_id = _CP_str_null;
/* Logon onto "pick0" as "dm" in the "dm" md */
  if (_CP_logon(machine, user, _CP_str_null,
      md, _CP_str_null, -1, 0) < 0)
  {
    printf("Logon error %d\n", _CP_errno);
    exit(1);
  }
  /* done with machine, user, and md */
  _CP_str_free(machine);
  _CP_str_free(user);
  _CP_str_free(md);
  /* Make the break key terminate the program */
  _CP_unix_break();
  /* Open the users file into the file descriptor fd */
  if (_CP_open(&fd, _CP_str_null, filename) < 0)
  {
    _CP_logoff();
    printf("Could not open users file \n");
    exit(2);
  }
  /* done with filename */
  _CP_str_free(filename);
  /* Do a raw select of "users" and store that select
     list in the select list descriptor sl */
  _CP_select(fd, &sl, 0);
  /* Switch to Unix-type I/O so we can use printf */
  _CP_unix_env();
  /* repeatedly read item-ids and print them as a list */
  /* Note how item_id is repeatedly passed to _CP_readnext.
     Each time, _CP_readnext will resize it if necessary,
     and write the next item-id into that CPSTR  */
  while (_CP_readnext(&item_id, &sl, 0, 0) >= 0)
  {
    _CP_TERM(item_id); /* terminate for printf */
```

```
    printf("%s\n", _CP_SADDR(item_id));
  }
  /* done with item_id - Not absolutely necessary since */
  /*   it will be cleaned up when we exit               */
  _CP_str_free(item_id);
  /* logoff and return */
  _CP_logoff();
  return 0;
}
```

C functions, integration

the steps necessary to integrate a user C program with the D3 run-time library:

- Build a branch table (which may be empty) for any functions which will be called by BASIC subroutines calling percent function calls.

- Edit and compile the user C program(s).

- Link the application with the D3 run-time library

The following describes the necessary procedures to create the "cdemo" example application shipped with the D3 package.

The steps required for creating this application are:

- Step 1: In Unix, copy "Makefile" and "cdemo.c" to the current directory.

Go to your Unix user's home directory and copy "Makefile" and "cdemo.c" from "/usr/lib/pick".

- Step 2: Enter Pick, and type "addbi" with no arguments.

The "addbi" command builds a dummy branch table for use by the BASIC-C interface. This branch table is not directly used by the C-D3 interface, but is required to properly link the D3 run-time library. This step creates two modules called "px.user.o" and "libgmu.a". These modules need only to be created once.

- Step 3: Edit the C program source code and Makefile

For the "cdemo" program, the source code is already complete. If this were a new program, the user would change the desired source file or files, and modify "Makefile" to integrate those files into the final link.

- Step 4: Link the application using "make".

In Unix, type "make cdemo". This will compile the "cdemo.c" source into an object and link this with the D3 run-time library to create a Unix application.

- Step 5: Run the application

When the "make" completes, an application called "cdemo" will exist on the current directory. This application is similar to the Unix "cat" program, except that "cdemo" outputs the raw contents of a D3 file and item. For instance, to view the "color" program in "dm,bp," type the following:

./cdemo bp color

Debugging can be done with the usual Unix debuggers like sdb or dbx.

C string

a series of characters terminated by a null (0x00) usually used in C.

Example

```
char x [] = "hi there";
```

CPSTR

defines the structure of a D3 data element manipulated from the C environment.

See the introduction to C functions for a general description of usage.

CPSTR*

points to a string structure used by routines that access D3 from the C programming language.

_CP_interrupt

is a global C integer which contains the last D3 interrupt that occured.

This value is that same as that returned by the FlashBASIC function "system(37)".

It is available when linking with the D3 libraries.

Syntax

int _CP_interrupt;

Example

```
/* The following example stops looping when some sort of interrupt occurs
(like a break or a message). */

_CP_unix_env();
while (!_CP_interrupt)    sleep(1); /* wait for something */
printf("I woke up!!!!\n");
```

_CP_str0

a static CPSTR* which points to a single character string with contains "0". It should be used for reading only.

Example

```
CPSTR * s = _CP_str0;
```

_CP_str_null

points to a null length string. It should be used for reading only, to initialize a pointer with a pointer to a null string.

Example

```
CPSTR * s = _CP_str_null;
```

_CP_alpha

equivalent to the FlashBASIC statement: result = alpha(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_alpha(int* result, CPSTR* string)

Example

```
/* is "hi" alphabetic? */

CPSTR * s = _CP_mkstr("hi");
int i;

_CP_alpha(&i,s);
_CP_unix_env();
```

```
printf("%d\n", i);
_CP_str_free(s);
```

_CP_ascii

equivalent to the FlashBASIC statement: result = ascii(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_ascii(CPSTR** result, CPSTR* string)

Example

```
CPSTR * s = _CP_mkstr("a");
_CP_unix_env();
_CP_ascii(&s,s);
printf("%c\n",*(_CP_SADDR(s)));
```

_CP_at1

equivalent to the FlashBASIC statement: result = @(x)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_at1(CPSTR** result, int x)

Example

```
/* clear the screen */
CPSTR * s = _CP_str_null;
_CP_at1(&s,-1);
_CP_print(s);
```

_CP_at2

equivalent to the FlashBASIC statement: result = @(x,y).

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_at2(CPSTR** result, int x, int y)

Example

```
/* home the cursor */
CPSTR * s = _CP_str_null;
_CP_at2(&s,1,1);
_CP_print(s);
```

_CP_break

equivalent to the FlashBASIC statement: break expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_break(int expression)

Example

```
/* turn the break off */
_CP_break(0);
```

_CP_build_msg

queues up arguments for _CP_out_msg.

This function, in conjunction with _CP_out_msg, is equivalent to the BASIC "ERROR" statement. Each parameter is first stacked with _CP_build_msg, and then output with the _CP_out_msg.

Syntax

int _CP_build_msg(CPSTR * s)

Example

```
/* Print file not found message */

CPSTR * s = _CP_mkstr("201");
CPSTR * t = _CP_mkstr("myfile");

_CP_build_msg(s);
_CP_build_msg(t);
_CP_out_msg();
```

_CP_call

is equivalent to the FlashBASIC statement: call name({string1 {, string2 {, ...}}})

The number parameter must be a pointer to an integer. On the first call to particular subroutine, this integer MUST contain -1. Upon successful return, this location will contain an index number which should be passed down to all succeeding _CP_call's to the same subroutine. This index allows the system to jump directly into the subroutine code without having to look up the name in the master dictionary.

The name parameter should be a CPSTR* pointing to the subroutine name.

The subroutine to be called MUST be previously compiled with FlashBASIC and be cataloged in the current master dictionary.

The expression must be the number of parameters to pass.

Each parameter passed to FlashBASIC must be a CPSTR**. Upon return from the subroutine, FlashBASIC will reconvert its variables back to strings, if necessary, and stuff the results back into the passed CPSTR**'s.

This function returns -1 if an error occurs. The error code is contained in _CP_errno. Specific error codes include:

PE_BAD_PARAMS indicates that too many parameters were passed as expression. The user may only pass _CP_MAX_PARAM parameters.

PE_LOAD_ERR indicates that the system could not load the subroutine. The subroutine must be cataloged, and must be compiled with the current version of FlashBASIC.

Syntax

int _CP_call(int* number, CPSTR* name, int expression, { CPSTR** string1 {, CPSTR** string2 {, ... }}})

Example

```
/* call a user-written routine */

CPSTR * s = _CP_mkstr("hi");
int i = -1;

r = _CP_call(&i,s,0);
if (r < 0)
```

```
{
_CP_logoff();
exit(-1);
}
/* Now that the subroutine is loaded, it can be called */
/* very efficiently                                    */

for (j=1; j < 100; j++)   _CP_call(&i,s,0);
```

_CP_casing

equivalent to the FlashBASIC statement: casing expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_casing(int expression)

Example

```
CPSTR * s = _CP_mkstr("a");
CPSTR * t = _CP_mkstr("A");
int i;

_CP_unix_env();
/* default is casing off - print 0 */
_CP_compare(&i,s,t);
printf("%d\n",i);
/* turn casing on - print 1 */
_CP_casing(1);
_CP_compare(&i,s,t);
printf("%d\n",i);
```

_CP_cat

is equivalent to the FlashBASIC statement: result = string1:string2

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_cat(CPSTR** result, CPSTR* string1, CPSTR* string2)

Example

```
CPSTR * s = _CP_mkstr("a");
CPSTR * t = _CP_mkstr("A");

_CP_unix_env();
_CP_cat(&s,s,t);
_CP_TERM(s);
printf("%s\n",_CP_SADDR(s));
```

_CP_clearfile

equivalent to the FlashBASIC statement: clearfile

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_clearfile(int expression)

Example

```
CPSTR * s = _CP_mkstr("myfile");
int f;
```

```
_CP_open(&f,_CP_str_null,s);
_CP_clearfile(f);
```

_CP_clearselect

equivalent to the FlashBASIC statement: clearselect

The expression should be an integer file descriptor returned by the _CP_select call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

This function should always be used when only part of a select list is referenced.

Syntax

int _CP_clearselect(int expression)

_CP_close

equivalent to the FlashBASIC statement: close

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_close(int expression)

Example

```
CPSTR * s = _CP_mkstr("myfile");
int f;

_CP_open(&f,_CP_str_null,s);
_CP_clearfile(f);
_CP_close(f);
```

_CP_col1

equivalent to the FlashBASIC statement: result = col1.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_col1(int* result)

Example

```
/* The following example prints "ab 2 5" */

CPSTR * s = _CP_mkstr("zdabdz");
CPSTR * t = _CP_mkstr("d");
int i,j;

_CP_field(&s,s,t,2);
_CP_col1(&i);
_CP_col1(&j);
_CP_TERM(s);
_CP_unix_env();
printf("%s %d %d\n",_CP_SADDR(s),i,j);
```

_CP_col2

equivalent to the FlashBASIC statement: "result = col2".

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_col2(int* result)

Example

```
The following example prints "ab 2 5"

CPSTR * s = _CP_mkstr("zdabdz");
CPSTR * t = _CP_mkstr("d");
int i,j;

_CP_field(&s,s,t,2);
_CP_col2(&i);
_CP_col2(&j);
_CP_TERM(s);
_CP_unix_env();
printf("%s %d %d\n",_CP_SADDR(s),i,j);
```

_CP_compare

equivalent to the FlashBASIC statement: result = (string1 = string2)

This function has the following return codes:

-1 string1 < string2

0 string1 = string2

1 string1 > string2

Syntax

int _CP_compare(int* result, CPSTR* string1, CPSTR* string2)

Example

```
CPSTR * s = _CP_mkstr("a");
CPSTR * t = _CP_mkstr("A");
int i;

_CP_unix_env();
/* default is casing off - print 0 */
_CP_compare(&i,s,t);
printf("%d\n",i);
/* turn casing on - print 1 */
_CP_casing(1);
_CP_compare(&i,s,t);
printf("%d\n",i);
```

_CP_convert

equivalent to the FlashBASIC statement: convert string1 to string2 in result

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_convert(CPSTR** result, CPSTR* string1, CPSTR* string2)

Example

```
/* The following example prints "bbc". */

CPSTR * s = _CP_mkstr("a");
CPSTR * t = _CP_mkstr("b");
CPSTR * u = _CP_mkstr("abc");

_CP_unix_env();
_CP_convert(&u,s,t);
_CP_TERM(u);
printf("%s\n",_CP_SADDR(u));
```

_CP_count

is equivalent to the FlashBASIC statement: result = count(string1, string2)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_count(int* result, CPSTR* string1, CPSTR* string2)

Example

```
/* The following example prints "2". */

CPSTR * s = _CP_mkstr("bbc");
CPSTR * t = _CP_mkstr("b");
int i;

_CP_count(&i,s,t);
_CP_unix_env();
printf("%d\n", i);
```

_CP_crt

equivalent to the FlashBASIC statement: crt

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_crt(CPSTR* string)

Example

```
CPSTR * t = _CP_mkstr("b");

_CP_crt(t);
```

_CP_crt_n

equivalent to FlashBASIC statement: crt

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_crt_n(CPSTR* string)

Example

```
CPSTR * t = _CP_mkstr("b");

_CP_crt_n(t);
```

_CP_data

Equivalent to the FlashBASIC statement: data string

data string

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_data(CPSTR* string)

Example

```
/* The following example copies the "xx" item in the md to item "yy".
*/

CPSTR * s = _CP_mkstr("copy md xx");
CPSTR * t = _CP_mkstr("yy");
CPSTR * c = _CP_str_null;
CPSTR * r = _CP_str_null;

_CP_data(t);
_CP_execute(_CP_EXECUTE,s,&c,&r);
```

_CP_date

is equivalent to the FlashBASIC statement: result = date()

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_date(int* result)

Example

```
/* The following example prints the numeric date */
int i;
_CP_date(&i);
_CP_unix_env();
printf("%d\n",i);
```

_CP_dcount

equivalent to the FlashBASIC statement: result = dcount(string1, string2)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_dcount(int* result, CPSTR* string1, CPSTR* string2)

Example

```
/* The following example prints "3". */

CPSTR * s = _CP_mkstr("bbc");
CPSTR * t = _CP_mkstr("b");
int i;

_CP_dcount(&i,s,t);
_CP_unix_env();
printf("%d\n", i);
```

_CP_debug

equivalent to the FlashBASIC statement: debug

The C function differs slightly however. The expression passed to _CP_debug will display as a pseudo line number within the FlashBASIC debugger. Variables within the C program are not known to the Flash debugger. For access to these, the user should use a system debugger such as "dbx" or "sdb". The _CP_debug statement is useful however, as it may be used to set breakpoints in Flash subroutines that will be called later.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_debug(int expression)

Example

```
/* The following example enters the FlashBASIC debugger and then calls routine
"mysub". */

CPSTR * s = _CP_mkstr("mysub");
int i = -1;

_CP_debug(1);
_CP_call(&i,s,0);

When run, the user may enter break points for the not-yet-executed subroutine
"mysub". For example, to stop when "mysub" reaches line number 3, the user
should type "mysub:b$=3" followed by a "g" at the debugger prompts. The Flash
debugger will then break within "mysub" at line number three.
```

_CP_delete

equivalent to the FlashBASIC statement: result = delete(string, ac.expression, vc.expression, sc.expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_delete(CPSTR** result, CPSTR* string, int ac.expression, int vc.expression, int sc.expression)

Example

```
/* The following example prints "a". */

CPSTR * s = _CP_mkstr("a\376b");

_CP_delete(&s,s,2,0,0);
_CP_print(s);
```

_CP_delete_item

equivalent to the FlashBASIC statement: delete expression,string

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_delete_item(int expression, CPSTR* string)

Example

```
/* The following example deletes item "1" from "myfile". */

CPSTR * s = _CP_mkstr("myfile");
CPSTR * t = _CP_mkstr("1");
int f;

_CP_open(&f,_CP_str_null,s);
_CP_delete_item(f,t);
```

_CP_dtx

equivalent to the FlashBASIC statement: result = dtx(expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_dtx(CPSTR** result, double expression)

Example

```
/* The following example prints "F". */

CPSTR * s = _CP_str_null;

_CP_dtx(&s,15.0);
_CP_print(s);
```

_CP_ebcdic

equivalent to the FlashBASIC statement: result = ebcdic(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_ebcdic(CPSTR** result, CPSTR* string)

Example

```
/* The following example prints "a". */

CPSTR * s = _CP_mkstr("/");
_CP_unix_env();
_CP_ascii(&s,s);
printf("%c\n",*(_CP_SADDR(s)));
```

_CP_echo

equivalent to the FlashBASIC statement: echo expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_echo(int expression)

Example

```
/* The following example gets input without echoing it. */

CPSTR * s = _CP_str_null;

_CP_echo(0);
_CP_input(_CP_INPUT,&s,0,0);
```

_CP_execute

equivalent to the FlashBASIC statement: execute command capturing string1 returning string2

The proper variation of execute is indicated by one of the following codes which should be passed as the type parameter:

_CP_EXECUTE execute command

_CP_EXECUTE_C execute command capturing string1

_CP_EXECUTE_R execute command returning string2

_CP_EXECUTE_CR execute command capturing string1 returning string2

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_execute(int type, CPSTR* command, CPSTR** string1, CPSTR** string2)

Example

```
/* The following example returns information about the current user.
*/

CPSTR * s = _CP_mkstr("who");
CPSTR * c = _CP_str_null;
CPSTR * r = _CP_str_null;

_CP_execute(_CP_EXECUTE_CR, s, &c, &r);
```

_CP_extract

equivalent to the FlashBASIC statement: result = extract(string, ac.expression, vc.expression, sc.expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_extract(CPSTR** result, CPSTR* string, int ac.expression, int vc.expression, int sc.expression)

Example

```
/* The following example prints "a". */

CPSTR * s = _CP_mkstr("a\376b");

_CP_extract(&s,s,1,0,0);
_CP_print(s);
```

_CP_field

is equivalent to the FlashBASIC statement: result = field(string1, string2, expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_field(CPSTR** result, CPSTR* string1, CPSTR* string2, int expression)

Example

```
/* The following example prints "ab 2 5" */

CPSTR * s = _CP_mkstr("zdabdz");
CPSTR * t = _CP_mkstr("d");
int i,j;

_CP_field(&s,s,t,2);
_CP_col1(&i);
_CP_col2(&j);
_CP_TERM(s);
_CP_unix_env();
printf("%s %d %d\n",_CP_SADDR(s),i,j);
```

_CP_field_store

equivalent to the FlashBASIC statement: result[string2, expression1, expression2] = string1

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_field_store(CPSTR** result, CPSTR* string1, CPSTR* string2, int expression1, int expression2)

Example

```
The following example prints "zdhidz"

CPSTR * s = _CP_mkstr("zdabdz");
CPSTR * t = _CP_mkstr("d");
CPSTR * u = _CP_mkstr("hi");

_CP_field_store(&s,u,t,2,1);
_CP_print(s);
```

_CP_filelock

equivalent to the FlashBASIC statement: filelock expression

This function returns -1 if an error occurs or if the file is already locked. The error code is contained in _CP_errno.

Syntax

int _CP_filelock(int expression)

_CP_fileunlock

equivalent to the FlashBASIC statement: fileunlock expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_fileunlock(int expression)

_CP_fold

equivalent to the FlashBASIC statement: result = fold(string1, string2, string3)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_fold(CPSTR** result, CPSTR* string1, CPSTR* string2, CPSTR* string3)

Example

```
The following example prints "A|long|strin|g"

CPSTR * s = _CP_mkstr("A long string");
CPSTR * t = _CP_mkstr("5");
CPSTR * u = _CP_mkstr("|");

_CP_fold(&s,s,t,u);
_CP_print(s);
```

_CP_footing

equivalent to the FlashBASIC statement: footing string

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_footing(CPSTR* string)

Example

```
/* The following example sets the footing to "footing". */

CPSTR * s = _CP_mkstr("footing");

_CP_footing(s);
```

_CP_get

equivalent to the FlashBASIC statement: get result, expression1 setting expression2 from expression3 until string1 returning string2, waiting expression4

The type parameter is bit mask that may be used to set various options:

_CP_GET_X the "getx" form of get.

_CP_GET_LEN a desired length is present.

_CP_GET_SET a setting parameter is present.

_CP_GET_UNTIL a until parameter is present.

_CP_GET_RTN a returning parameter is present.

_CP_GET_WAIT a waiting parameter is present.

_CP_GET_THEN_ELSE then/else condition.

_CP_GET_ELSE else only condition.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_get(int type, CPSTR** result, int expression1, int expression2, int expression3, CPSTR* string1, CPSTR** string2, int expression4)

Example

```
/* The following example gets input from port 1. */

CPSTR * s = _CP_mkstr("dev-att 1");
CPSTR * c = _CP_str_null;
CPSTR * r = _CP_str_null;

_CP_execute(_CP_EXECUTE,s,&c,&r);
_CP_get(_CP_GET_X+_CP_GET_LEN, &r, 1, 0, 1, c, &c, 50);
```

_CP_heading

equivalent to the FlashBASIC statement: heading string

This function returns -1 if an error occurs. The error code is contained in _CP_errno. If the heading is too long, _CP_errno will contain PE_LONG_STR.

Syntax

int _CP_heading(CPSTR* string)

Example

```
/* The following example sets the heading to "heading". */

CPSTR * s = _CP_mkstr("heading");

_CP_heading(s);
```

_CP_iconv

equivalent to the FlashBASIC statement: result = iconv(string1, string2)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_iconv(CPSTR** result, CPSTR* string1, CPSTR* string2)

Example

```
/* The following example prints "a". */

CPSTR * s = _CP_mkstr("a2");
CPSTR * t = _CP_mkstr("mca");

_CP_iconv(&s,s,t);
_CP_print(s);
```

_CP_in

Equivalent to the FlashBASIC statement: in result for expression

If no "for" clause is desired, the user should pass -1 in expression.

This function returns -1 if an error occurs. The error code is contained in _CP_errno. If the timeout occurs, _CP_errno contains PE_TMOUT.

Syntax

int _CP_in(int* result, int expression)

Example

```
/* The following example gets 1 character of input. */

int i;

_CP_in(&i,-1);
_CP_unix_env();
printf("%d\n", i);
```

_CP_index

equivalent to the FlashBASIC statement: result = index(string1, string2, expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_index(int* result, CPSTR* string1, CPSTR* string2, int expression)

Example

```
/* The following example prints "2". */

CPSTR * s = _CP_mkstr("bbc");
CPSTR * t = _CP_mkstr("b");
int i;

_CP_index(&i,s,t,2);
_CP_unix_env();
printf("%d\n", i);
```

_CP_input

equivalent to the FlashBASIC statements: input

type FlashBASIC statement

_CP_INPUT input result

_CP_INPUT_N input result:

_CP_INPUT_L input result, expression1

_CP_INPUT_LN input result, expression1:

_CP_INPUT_L_ input result, expression1_

_CP_INPUT_LN_ input result, expression1:_

_CP_INPUT_LF input result, expression1 for expression2

_CP_INPUT_LNF input result, expression1: for expression2

_CP_INPUT_L_F input result, expression1_ for expression2

_CP_INPUT_LN_F input result, expression1:_ for expression2

This function returns -1 if an error occurs. The error code is contained in _CP_errno. If the timeout occurs, _CP_errno contains PE_TMOUT.

Syntax

int _CP_input(int type, CPSTR** result, int expression1, int expression2)

Example

```
/* The following example gets 1 line of input. */

CPSTR * s = _CP_str_null;

_CP_input(_CP_INPUT, &s, 0, 0);
```

_CP_insert

equivalent to the FlashBASIC statement: result = insert(string1, ac.expression, vc.expression, sc.expression, string2)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_insert(CPSTR** result, CPSTR* string1, int ac.expression, int vc.expression, int sc.expression, CPSTR* string2)

Example

```
/* The following example prints "a^b^c". */

CPSTR * s = _CP_mkstr("a\376b");
CPSTR * t = _CP_mkstr("c");

_CP_insert(&s,s,3,0,0,t);
_CP_print(s);
```

_CP_is_

converts an integer into a CPSTR.

Syntax

CPSTR * _CP_is_(int number)

Example

```
CPSTR * s = _CP_is_(1.123);

_CP_print(s);

 This example displays "1".
```

_CP_key

equivalent to the FlashBASIC statement: key(string1, expression, string2, string3)

An additional parameter, value, is available from C which returns the current value of the key if it is multi-valued.

This function returns -1 if an error occurs. The error code is contained in _CP_errno. If the key operator is invalid, _CP_errno will contain PE_ILL_KEY. If the root pointer is invalid, _CP_errno will contain PE_NOTROOT.

Syntax

int _CP_key(CPSTR* string1, int expression, CPSTR** string2, CPSTR** string3, int* value)

Example

```
The following gets the first item-id which contains "a" as attribute
1.

CPSTR * n = _CP_mkstr("myfile");
CPSTR * a = _CP_mkstr("a1");
CPSTR * op = _CP_mkstr("n");
CPSTR * k = _CP_mkstr("a");
CPSTR * i = _CP_str_null;
int r,dummy;

_CP_root(n,a,&r);
_CP_key(op,r,&k,&i,&dummy);
```

_CP_load

is an optional function for pre-loading FlashBASIC subroutines, before they are needed.

The number parameter must be a pointer to an integer. This integer MUST contain -1 when loading a new subroutine. Upon successful load, this location will contain an index number which should be passed down to succeeding _CP_call's to the same subroutine. This index allows the system to jump directly into the subroutine code without having to look up the name in the master dictionary.

The name parameter should be a CPSTR* pointing to the subroutine name.

The subroutine to be called MUST be previously compiled with FlashBASIC and be cataloged in the current master dictionary.

This function returns -1 if an error occurs. The error code is contained in _CP_errno. Specific error codes include:

PE_LOAD_ERR indicates that the system could not load the subroutine. The subroutine must be cataloged, and must be compiled with the current version of FlashBASIC.

Syntax

int _CP_load(int* number, CPSTR* name)

Example

```
/* call a user-written routine */

CPSTR * s = _CP_mkstr("hi");
int i = -1;

r = _CP_load(&i,s);
if (r < 0)
{
_CP_logoff();
exit(-1);
}
/* Now that the subroutine is loaded, it can be called */
/* very efficiently                                    */

for (j=1; j < 100; j++)   _CP_call(&i,s,0);
```

_CP_locate

equivalent to the FlashBASIC statement: locate(string1, string2, ac.expression, vc.expression, start.expression, position.variable, string3) then result = 1 else result = 0

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_locate(int* result, CPSTR* string1, CPSTR* string2, int ac.expression, int vc.expression, int start.expression, int * position.variable, CPSTR* string3)

Example

```
/* The following example prints "1 2". */

CPSTR * s = _CP_mkstr("a\376b");
CPSTR * t = _CP_mkstr("b");
int r,l;

_CP_locate(&r,t,s,0,0,0,&l,_CP_str_null);
_CP_unix_env();
printf("%d %d\n", r, l);
```

_CP_lock

equivalent to the FlashBASIC statement: lock

The variant of lock used depends upon the type parameter:

Type FlashBASIC statement

_CP_LOCK lock expression

_CP_LOCK_W lock expression else * return -1

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_lock(int type, int expression)

Example

```
The following example sets basic lock 1.

_CP_lock(_CP_LOCK,1);
```

_CP_logon

logs into D3 from a C main program.

This function must be called prior to any other function. The parameters used are the following:

machine

D3 virtual machine

user

D3 user

u_passwd

D3 user password

md

D3 master dictionary

md_passwd

D3 master dictionary password

pib

Requested pib or -1 for first available

flags

Reserved field - always pass a 0

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_logon(CPSTR* machine, CPSTR* user, CPSTR* u_passwd, CPSTR* md, CPSTR* md_passwd, int pib, int flags);

Example

```
The following example logs into D3 as the "dm" user in the "dm" account. It
assumes that no passwords are present.

CPSTR * machine = _CP_mkstr("pick0");
CPSTR * user = _CP_mkstr("dm");
CPSTR * md = _CP_mkstr("dm");
```

```
int r;

r=_CP_logon(machine,user,_CP_str_null,md,_CP_str_null,-1,0);
```

_CP_match

equivalent to the FlashBASIC statement: result = string1 match string2

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_match(int* result, CPSTR* string1, CPSTR* string2)

Example

```
/* The following example prints "1". */

CPSTR * s = _CP_mkstr("a1c");
CPSTR * t = _CP_mkstr("1a1n1a");
int i;

_CP_match(&i,s,t);
_CP_unix_env();
printf("%d\n", i);
```

_CP_mkstr

dynamically allocates a CPSTR* and fills it with a copy of cstring.

Syntax

CPSTR* _CP_mkstr(char* cstring)

Example

```
CPSTR * s = _CP_mkstr("This is a string");
```

_CP_mkstrl

dynamically allocates a CPSTR* of length expression and fills it with a copy of cstring.

Syntax

CPSTR* _CP_mkstrl(char* cstring, int expression)

Example

```
CPSTR * s = _CP_mkstr("Hi",2);
```

_CP_num

equivalent to the FlashBASIC function: result = num(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_num(int* result, CPSTR* string)

Example

```
/* is "1" alphabetic? */

CPSTR * s = _CP_mkstr("1");
int i;

_CP_num(&i,s);
_CP_unix_env();
printf("%d\n", i);
```

_CP_occurs

equivalent to the FlashBASIC function: occurs

result = occurs(string, expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_occurs(CPSTR** result, CPSTR* string, int expression)

Example

```
/* The following example prints "a". */

CPSTR * s = _CP_mkstr("a\376a");

_CP_occurs(&s,s,2);
_CP_print(s);
```

_CP_oconv

equivalent to the following FlashBASIC statement: result = oconv(string1, string2)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_oconv(CPSTR** result, CPSTR* string1, CPSTR* string2)

Example

```
CPSTR * s = _CP_mkstr("a2");
CPSTR * t = _CP_mkstr("mca");

_CP_oconv(&s,s,t);
_CP_print(s);
Prints "a".
```

_CP_open

equivalent to the FlashBASIC statement: open

open string1,string2 to fd

Upon successful return, the first parameter is a pointer to an integer file descriptor which may be passed to other file system calls.

string1 should be "dict" or "data", or may be passed as _CP_str_null.

string2 should be the file name.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_open(int* fd, CPSTR* string1, CPSTR* string2)

Example

```
CPSTR * s = _CP_mkstr("myfile");
int f;

_CP_open(&f,_CP_str_null,s);
_CP_clearfile(f);
_CP_close(f);
```

_CP_out

equivalent to the FlashBASIC statement: "out"

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_out(int expression)

Example

```
/* The following example prints a space. */

_CP_out(0x20);
```

_CP_out_msg

outputs an error message.

This function outputs an error message which has been previously queued by _CP_build_message. See _CP_build_message for a coding example.

Syntax

int _CP_out_msg()

_CP_ovfly_subs

equivalent to the FlashBASIC statement: result[expression1, expression2] = string1

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_ovly_subs(CPSTR** result, CPSTR* string1, int expression1, int expression2)

Example

```
/* The following example prints "hicdef" */

CPSTR * s = _CP_mkstr("abcdef");
CPSTR * u = _CP_mkstr("hi");

_CP_ovly_subs(&s,u,2,1);
_CP_print(s);
```

_CP_page

equivalent to the FlashBASIC statement: page

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_page()

Example

```
_CP_page();
```

_CP_page_n

equivalent to the FlashBASIC statement: page expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_page_n(int expression)

Example

```
_CP_page_n();
```

_CP_pick_break

causes all subsequent breaks to push a level or drop into the debugger as is standard in D3.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_pick_break()

Example

```
_CP_pick_break();
```

_CP_pick_env

sets the application to a standard D3 environment.

When using this setting, all terminal input and output must be done via D3 routines only (execute, print, crt).

The default environment after logging onto D3 is _CP_pick_env.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_pick_env()

Example

```
_CP_pick_env();
```

_CP_precision

equivalent to the FlashBASIC statement: precision number.

This function affects the precision of the _CP_rs_ and the _CP_sr_ functions.

Syntax

void _CP_precision(int number)

Example

```
CPSTR * s = _CP_mkstr("1.12345");
double r;

_CP_precision(2);
r = _CP_sr_(s);
_CP_unix_env();
printf("%g\n", r);
_CP_str_free(s);

 This example displays "1.12" since the conversion is truncated to the
precision of 2.
```

_CP_print

equivalent to the FlashBASIC statement: print string

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_print(CPSTR* string)

Example

```
CPSTR * t = _CP_mkstr("b");
_CP_print(t);
```

_CP_printer

equivalent to the FlashBASIC statement: printer expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_printer(int expression)

Example

```
/*The following example prints "hi" to the printer.*/

CPSTR * s = _CP_mkstr("hi");

_CP_printer(1);
_CP_print(s);
_CP_printer(0);
_CP_printer(-1);
```

_CP_print_n

equivalent to the FlashBASIC statement: print string:

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_print_n(CPSTR* string)

Example

```
CPSTR * t = _CP_mkstr("b");
_CP_print_n(t);
```

_CP_print_on

equivalent to the following FlashBASIC statement: print on expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_print_on(int expression)

Example

```
The following example prints "hi" to print job 1.

CPSTR * s = _CP_mkstr("hi");

_CP_print_on(1);
_CP_print(s);
```

_CP_prompt

equivalent to the FlashBASIC statement: prompt

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_prompt(CPSTR* string)

Example

```
CPSTR * s = _CP_mkstr(">");

_CP_prompt(s);
Sets the prompt to ">".
```

_CP_read

equivalent to the FlashBASIC statements depending upon type: read, readu, readu locked

type FlashBASIC statement

_CP_READ read result from expression,string

_CP_READU readu result from expression,string

_CP_READUL readu result from expression,string locked value = 1 else value = 0

If value is not needed, the user may pass (int*) 0.

The _CP_read call uses the same optimized read routine as FlashBASIC which is about 2-5 times faster than that which is used by AQL and standard Pick/BASIC.

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_read(int type, CPSTR** result, int expression, CPSTR* string, int* value)

Example

```
The following example reads the item "myid" from "myfile".

CPSTR * s = _CP_mkstr("myfile");
CPSTR * id = _CP_mkstr("myid");
CPSTR * xx = _CP_str_null;
int f;

_CP_open(&f,_CP_str_null,s);
_CP_read(_CP_READ, &xx, f, id, 0);
```

_CP_readnext

equivalent to the FlashBASIC statement: readnext result,value from list

If expression is non-zero, then the "secondary" list is assumed.

The _CP_readnext call must be proceeded by a _CP_select call or a _CP_execute call which produces an external select list. In the case of an external list generated by a _CP_execute, the user should pass a pointer to an int containing -1 as the list parameter when readnext is first called.

This function returns -1 if an error occurs. The error code is contained in _CP_errno. This situation occurs when there are no more items in the list. In this case, _CP_errno will contain PE_END_LIST.

Syntax

int _CP_readnext(CPSTR** result, int* list, int* value, int expression)

Example

```
The following example prints the first file name in the current account.

CPSTR * s = _CP_mkstr("select md with a1 \"d]\" sampling 1");
CPSTR * id = _CP_str_null;
int sl = -1;

_CP_execute(_CP_EXECUTE, s, (CPSTR**) 0, (CPSTR**) 0);
_CP_readnext(&id, &sl, (int*) 0, 0);
_CP_print(id);

The following example prints the item names in "myfile".

CPSTR * n = _CP_mkstr("myfile");
CPSTR * id = _CP_str_null;
int sl = -1;
int f = -1;

_CP_open(&f, _CP_str_null, n);
_CP_select(f, &sl, 0);
while (_CP_readnext(&id, &sl, (int*) 0, 0) >= 0)
  _CP_print(id);
```

_CP_readt

equivalent to the FlashBASIC statements: readt

Which flavor of "readt" is dependent upon the command "type".

type FlashBASIC statement

_CP_READT_ELSE readt result else * return -1

_CP_READT_L_ELSE readtl result else * return -1

_CP_READTX_ELSE readtx result else * return -1

_CP_READT_ONERR readt result onerr * return -1

_CP_READT_L_ONERR readtl result onerr * return -1

_CP_READTX_ONERR readtx result onerr * return -1

This function returns -1 if an error occurs. The error code is contained in _CP_errno. PE_TAPE indicates a tape error.

Syntax

int _CP_readt(int type, CPSTR** result)

Example

```
/*The following example prints the next tape block.*/

CPSTR * xx = _CP_str_null;

_CP_readt(_CP_READT_ELSE, xx);
_CP_print(xx);
```

_CP_readv

equivalent to the FlashBASIC statement: readv

Depending upon type, _CP_read acts as the following FlashBASIC commands:

type FlashBASIC statement

_CP_READV readv result from expression1, string, expression2

_CP_READVU readvu result from expression1, string, expression2

_CP_READVUL readvu result from expression, string, expression2

The value parameter returns a 1 if the item is locked.

If value is not needed, the user may pass (int*) 0.

The _CP_readv call uses the same optimized read routine as FlashBASIC which is about 2-5 times faster than that which is used by AQL and standard Pick/BASIC.

The expression1 should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno. If the attribute number is bad, then _CP_errno will contain PE_BADATTR.

Syntax

int _CP_readv(int type, CPSTR** result, int expression1, CPSTR* string, int expression2, int* value)

Example

```
/* The following example reads the first attribute of the item "myid"
from "myfile". */

CPSTR * s = _CP_mkstr("myfile");
CPSTR * id = _CP_mkstr("myid");
CPSTR * xx = _CP_str_null;
int f;

_CP_open(&f,_CP_str_null,s);
_CP_readv(_CP_READV, xx, f, id, 1, 0);
```

_CP_release

is equivalent to the FlashBASIC statement: release

release expression,string

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_release(int expression, CPSTR* string)

Example

```
/* The following example releases a previously set item lock on item "1" in
"myfile". */

CPSTR * s = _CP_mkstr("myfile");
CPSTR * t = _CP_mkstr("1");
int f;

_CP_open(&f,_CP_str_null,s);
_CP_release(f,t);
```

_CP_release_all

equivalent to the FlashBASIC statement: release

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_release_all()

Example

```
/*The following example releases all previously set item locks.*/

_CP_release_all();
```

_CP_replace

equivalent to the FlashBASIC statement: result = replace(string1, ac.expression, vc.expression, sc.expression, string2)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_replace(CPSTR** result, CPSTR* string1, int ac.expression, int vc.expression, int sc.expression, CPSTR* string2)

Example

```
/* The following example prints "c^b". */

CPSTR * s = _CP_mkstr("a\376b");
CPSTR * t = _CP_mkstr("c");

_CP_replace(&s,s,1,0,0,t);
_CP_print(s);
```

_CP_replace_bridge

equivalent to the following FlashBASIC statement: replace expression,string1 with string2

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_replace_bridge(int expression, CPSTR* string1, CPSTR* string2)

_CP_rewind

equivalent to the FlashBASIC statement: rewind

type FlashBASIC statement

_CP_REWIND_ELSE rewind else * return -1

_CP_REWIND_ONERR rewind else * return -1

This function returns -1 if an error occurs. The error code is contained in _CP_errno. PE_TAPE indicates a tape error.

Syntax

int _CP_rewind(int type)

Example

```
/* The following example rewinds the tape. */

_CP_rewind(_CP_REWIND_ELSE);
```

_CP_root

equivalent to the FlashBASIC statement: root string1, string2 to result.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_root(CPSTR* string1, CPSTR* string2, int* result)

Example

```
/* The following gets the first item-id which contains "a" as
attribute 1. */

CPSTR * n = _CP_mkstr("myfile");
CPSTR * a = _CP_mkstr("a1");
CPSTR * op = _CP_mkstr("n");
CPSTR * k = _CP_mkstr("a");
CPSTR * i = _CP_str_null;
int r,dummy;

_CP_root(n,a,&r);
_CP_key(op,r,&k,&i,&dummy);
```

_CP_rs_

converts a double floating point number into a CPSTR.

The precision of this function can be changed using the _CP_precision function.

Syntax

CPSTR * _CP_rs_(double number)

Example

```
CPSTR * s = _CP_rs_(1.123);

_CP_print(s);

 This example displays "1.123".
```

_CP_SADDR

returns a standard char* pointing to the first character of the string buffer within a CPSTR structure.

Because _CP_SADDR is a macro, it may be used either on the left-hand or the right-hand side of an assignment operator.

Syntax

char* _CP_SADDR(CPSTR* string)

Example

```
CPSTR * s = _CP_mkstr("hi");
_CP_SADDR(s)[1] = 'o';          /* replace 2nd character  */
_CP_unix_env();                 /* Unix print environment */
_CP_TERM(s);                    /* null terminate */
printf("%s\n",_CP_SADDR(s));
```

_CP_select

equivalent to the following FlashBASIC statement: select expression1 to list

If expression2 is non-zero, then the "secondary" list is assumed.

The default behavior of _CP_select is to create an internal select list which contains the item-id's of the file pointed to by expression1. Expression1 must have been previously created by a _CP_open call.

Note however, that, as in FlashBASIC, if an external select list is present (i.e. one created by a previous call to _CP_execute), that the value of expression1 will be ignored. In this case, the user may pass -1 for expression1 if desired.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

The user C program should always process all of the items in the list or call _CP_clearselect to dispose of the list prematurely. Failure to do this may cause unreasonable memory or overflow usage.

Syntax

int _CP_select(int expression1, int* list, int expression2)

Example

```
The following example prints the item names in "myfile".

CPSTR * n = _CP_mkstr("myfile");
CPSTR * id = _CP_str_null;
int sl = -1;
int f = -1;

_CP_open(&f, _CP_str_null, n);
_CP_select(f, &sl, 0);
while (_CP_readnext(&id,&sl,(int*)0,0)>=0)
  _CP_print(id);

The following example demonstrates how an external select list will override
the file variable passed as expression1.  Here, "myfile" is passed to the
select, but the readnext will return items from the "md".

CPSTR * n = _CP_mkstr("myfile");
CPSTR * s = _CP_mkstr("select md sampling 3");
CPSTR * id = _CP_str_null;
```

```
int sl = -1;
int f = -1;

_CP_execute(_CP_EXECUTE, s, (CPSTR**) 0, (CPSTR**) 0);
_CP_open(&f, _CP_str_null, n);
_CP_select(f, &sl, 0);
while (_CP_readnext(&id, &sl, (int*) 0, 0) >= 0)
  _CP_print(id);
```

_CP_send

equivalent to the FlashBASIC statement: send{x} string{:} to port.number

The type parameter is a bit mask that may be used to set various options that must or'ed together:

_CP_SEND_X requests the "sendx" form of send.

_CP_SEND_N suppresses the new-line

_CP_SEND_ELSE indicates an else condition

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_send(int type, CPSTR* string, int port.number)

Example

```
/* The following example sends "hi" to port 1. */

CPSTR * s = _CP_mkstr("dev-att 1");
CPSTR * t = _CP_mkstr("hi");
CPSTR * c = _CP_str_null;
CPSTR * r = _CP_str_null;

_CP_execute(_CP_EXECUTE_C,s,&c,&r);
_CP_send(0, t, 1);
```

_CP_si_

converts a CPSTR into an integer.

Syntax

int _CP_si_(CPSTR * string)

Example

```
CPSTR * s = _CP_mkstr("3.12345");
int i;

i = _CP_si_(s);
_CP_unix_env();
printf("%d\n", i);
_CP_str_free(s);

 This example displays "3".
```

_CP_sleep

equivalent to the FlashBASIC statement: sleep(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_sleep(CPSTR* string)

Example

```
/* sleep 5 seconds */

CPSTR * s = _CP_mkstr("5");
_CP_sleep(s);
_CP_str_free(s);
```

_CP_SLEN

returns the length of the CPSTR*, string.

Syntax

int _CP_SLEN(CPSTR* string)

Example

```
CPSTR * s = _CP_mkstr("hi");
int i = _CP_SLEN(s);
```

_CP_sort

equivalent to the FlashBASIC statement: "result = sort(string)"

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_sort(CPSTR** result, CPSTR* string)

Example

```
The following example prints "a^b".

CPSTR * s = _CP_mkstr("b\376a");

_CP_sort(&s,s);
_CP_print(s);
```

_CP_soundex

equivalent to the FlashBASIC statement: result = soundex(string, type).

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_soundex(CPSTR** result, CPSTR* string, int type)

Example

```
The following example prints "H000".

CPSTR * s = _CP_mkstr("hi");

_CP_soundex(&s,s,0);
_CP_print(s);
```

_CP_space

equivalent to the FlashBASIC statement: result = space(expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_space(CPSTR** result, int expression)

Example

```
The following example prints "   ".

CPSTR * s = _CP_str_null;
```

```
_CP_space(&s,3);
_CP_print(s);
```

_CP_sr_

converts a CPSTR into a double float number.

The precision of the conversion can be changed with the _CP_precision() function.

Syntax

double _CP_sr_(CPSTR * string)

Example

```
CPSTR * s = _CP_mkstr("1.12345");
double r;

_CP_precision(2);
r = _CP_sr_(s);
_CP_unix_env();
printf("%g\n", r);
_CP_str_free(s);

 This example displays "1.12" since the conversion is truncated to the
precision of 2.
```

_CP_str

equivalent to the FlashBASIC statement: result = str(string, expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_str(CPSTR** result, CPSTR* string, int expression)

Example

```
The following example prints "aaa".

CPSTR * s = _CP_mkstr("a");

_CP_str(&s,s,3);
_CP_print(s);
```

_CP_str_alloc

allocates space for a CPSTR.

Equivalent of "malloc()", but returns a CPSTR*. "expression" should contain the length of the string buffer to allocate. A 0 will be returned if failure occurs.

Syntax

CPSTR* _CP_str_alloc(int expression)

_CP_str_copy

makes a copy of a CPSTR. The actual contents of the string are copied rather than just the pointer.

Syntax

CPSTR * _CP_str_copy(CPSTR * string)

Example

```
CPSTR * s = _CP_mkstr("Null value");
CPSTR * a[20];
```

```
int     i;

for (i = 0; i < 20; i++)
    a[i] = _CP_str_copy(s);
```

This example assigns the string "Null value" to all the elements of the array "a". Note that copies are necessary so that each element can be modified without affecting the others.

_CP_str_free

releases the CPSTR* string back to the string pool to be allocated at a later time. All CPSTR*'s must eventually be released with _CP_str_free.

Syntax

_CP_str_free(CPSTR* string)

_CP_str_realloc

reallocates a CPSTR* to a new length.

"old" points to the string to reallocate. "expression" is the desired new length. If the request is for a smaller length than the current one, or if the request is for a larger length, but the system senses that the internal buffer size of the old CPSTR* is sufficient to handle the extra length, then the routine simply changes the sizea and returns a pointer equal to "old". If the requested size is too big for the internal buffer, then a new pointer is created, the old data copied to the new one, and the old string released. Any new buffer space not filled in by the old copy is undefined.

Syntax

CPSTR* _CP_str_realloc(CPSTR* old, int expression)

Example

```
CPSTR * s = _CP_mkstr("hi");
s = _CP_str_realloc(s, 3);
_CP_SADDR(s)[2] = '!';
```

_CP_substr

equivalent to the FlashBASIC statement: result = string[beg.pos.expression, len.expression]

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_substr(CPSTR** result, CPSTR* string, int beg.pos.expression, int len.expression)

Example

```
The following example prints "h".

CPSTR * s = _CP_mkstr("hi");

_CP_substr(&s,s,1,1);
_CP_print(s);
```

_CP_sum

equivalent to the FlashBASIC statement: result = sum(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_sum(CPSTR** result, CPSTR* string)

Example

```
The following example prints "3".

CPSTR * s = _CP_mkstr("1\3762");

_CP_sum(&s,s);
_CP_print(s);
```

_CP_system

Equivalent to the FlashBASIC STATEMENT: result = system(expression)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_system(CPSTR** result, int expression)

Example

```
/* Get a unique item-id in s */

CPSTR * s = _CP_str_null;

_CP_system(&s, 19);
```

_CP_TERM

terminates a CPSTR* string with a 0x00.

Syntax

_CP_TERM(CPSTR* string)

Example

```
CPSTR *s= _CP_mkstr( "Enter name " );
_CP_TERM( s );
printf( _CP_ADDR( s ) );
Creates a string and terminates it with a 0, so that 'printf' can be used.
```

_CP_time

equivalent to the FlashBASIC statement: result = time()

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_time(int* result)

Example

```
The following example prints the numeric time.

int i;

_CP_time(&i);
_CP_unix_env();
printf("%d\n",i);
```

_CP_timedate

equivalent to the FlashBASIC statement: result = timedate()

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_timedate(CPSTR** result)

Example

```
The following example prints the date and time.

CPSTR * s = _CP_str_null;

_CP_timedate(&s);
_CP_print(s);
```

_CP_trans

equivalent to on e of several FlashBASIC transaction processing statements depending on the "op" parameter. Valid values of "op" are:

_CP_TRANS_ONOFF Equivalent to the BASIC "TRANSACTION ON/OFF" statement. Pass in the on/off parameter in "exp", and _CP_str_null in "str".

_CP_TRANS_START Equivalent to the BASIC "BEGIN WORK {str}" statement. If no transaction name is desired, then pass 0 in "exp" and _CP_str_null in "str". If a transaction name is desired, then pass 1 in "exp" and the desired name in "str".

_CP_TRANS_COMMIT Equivalent to the BASIC "COMMIT WORK" statement. Pass 0 in "exp" and _CP_str_null in "str".

_CP_TRANS_ROLLBACK Equivalent to the BASIC "ROLLBACK WORK" statement. A 0 MUST be passed in "exp" and _CP_str_null in "str".

_CP_CACHE Equivalent to the BASIC "TRANSACTION CACHE ON/OFF" statement. Pass in the on/off parameter in "exp", and _CP_str_null in "str".

_CP_FLUSH Equivalent to the BASIC "TRANSACTION FLUSH ON/OFF" statement. Pass in the on/off parameter in "exp", and _CP_str_null in "str".

See the BASIC "Commit Work" documentation for more information on transaction processing.

This function returns -1 if an error occurs. The error code is contained in _CP_errno and is always PE_ACCESS.

Syntax

int _CP_trans(int op, int exp, CPSTR * str)

_CP_trim

equivalent to the FlashBASIC statement: result = trim(string)

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_trim(CPSTR** result, CPSTR* string)

Example

```
The following example prints "3".

CPSTR * s = _CP_mkstr("  3   ");

_CP_trim(&s,s);
_CP_print(s);
```

_CP_unix_break

causes all subsequent breaks to terminate the process as is standard in Unix applications.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax
int _CP_unix_break()

Example
```
_CP_unix_break();
```

_CP_unix_env

sets the application to a standard Unix environment. When using this setting, all terminal input and output must be done via standard Unix routines only (system, printf, scanf).

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax
int _CP_unix_env()

_CP_unlock

Equivalent to the FlashBASIC statement: unlock expression

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax
int _CP_unlock(int expression)

Example
```
_CP_unlock(1);
```

_CP_unlock_all

equivalent to the FlashBASIC statement: unlock

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax
int _CP_unlock_all()

Example
```
The following example releases all basic locks owned by the current process.

_CP_unlock_all();
```

_CP_weof

equivalent to the FlashBASIC statement: weof

type FlashBASIC statement

_CP_WEOF_ELSE weof else return -1

_CP_WEOF_ONERR weof onerr return -1

This function returns -1 if an error occurs. The error code usually returned in system(0) is contained in _CP_errno. PE_TAPE indicates a tape error.

Syntax
int _CP_weof(int type)

Example
```
The following example writes an eof on the tape.

_CP_weof(_CP_WEOF_ELSE);
```

_CP_write

equivalent to the FlashBASIC statements: write or writeu

The equivalency is dependent upon the "type" parameter.

type FlashBASIC statement

_CP_WRITE write string1 on expression,string2

_CP_WRITEU writeu string1 on expression,string2

The expression should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_write(int type, CPSTR* string1, int expression, CPSTR* string2)

Example

```
The following example writes "hi" in item "myid" in "myfile".

CPSTR * s = _CP_mkstr("myfile");
CPSTR * t = _CP_mkstr("hi");
CPSTR * id = _CP_mkstr("myid");
int f;

_CP_open(&f,_CP_str_null,s);
_CP_write(_CP_WRITE, t, f, id);
```

_CP_writet

equivalent to the FlashBASIC statements: writet

type FlashBASIC statement

_CP_WRITET_ELSE writet string else return -1

_CP_WRITET_ONERR writet string onerr return -1

_CP_WRITET_L_ELSE writetl string onerr return -1

_CP_WRITET_L_ONERR writetl string onerr return -1

This function returns -1 if an error occurs. The error code is contained in _CP_errno. PE_TAPE indicates a tape error.

Syntax

int _CP_writet(int type, CPSTR* string)

Example

```
The following example writes the next tape block.

CPSTR * xx = _CP_mkstrl("data",4);

_CP_writet(_CP_WRITET_ELSE, xx);
```

_CP_writev

equivalent to the FlashBASIC statements: writev, writevu

type FlashBASIC statement

_CP_WRITEV writev string1 on expression1,string2,expression2

_CP_WRITEUV writevu string1 on expression1,string2,expression2

The expression1 should be an integer file descriptor returned by the _CP_open call.

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_writev(int type, CPSTR* string1, int expression1, CPSTR* string2, expression2)

Example

The following example writes "hi" into attribute 1 of item "myid" in "myfile".

```
CPSTR * s = _CP_mkstr("myfile");
CPSTR * t = _CP_mkstr("hi");
CPSTR * id = _CP_mkstr("myid");
int f;

_CP_open(&f,_CP_str_null,s);
_CP_writev(_CP_WRITE, t, f, id, 1);
```

_CP_xtd

equivalent to the FlashBASIC statement: result = xtd(string).

This function returns -1 if an error occurs. The error code is contained in _CP_errno.

Syntax

int _CP_xtd(double* result, CPSTR* string)

Example

The following example prints 15.

```
CPSTR * s = _CP_mkstr("F");
double f;

_CP_xtd(&f,s);
_CP_unix_env();
printf("%f\n",f);
```

Editor

one of the original tools of the D3 System for adding, changing and deleting items in files.

It is a "line-oriented" editor, and provides an assortment of commands that move the "line pointer" to specific attributes (lines) in the item being edited.

Since everything in D3 is an item in a file, the editor can be used on any item in any file, unless otherwise prevented by retrieval or update lock codes.

Each attribute in an item being edited is treated as a separate line. The numbers to the left of the attribute indicate the "line" or "attribute" number.

error messages

a brief discussion of the various error messages associated with the line editor.

col#?	Appears when a non-numeric character appears in a syntax position that would ordinarily require a number.
Cmnd?	Short for "command?". Indicates that an invalid Editor command was issued.
Eoi n	End of item occurs at line "n".
L n	Indicates either the current line number (with the "?" command), or the last line affected by an "x" (oops) command.
Not on file	Can't find the requested item with the "me" (merge) command.
Seqn?	Sequence? Changes must be made from the top down. An "f" (flip) command clears this up. Note that when this appears, the requested change does NOT take place.
String?	Usually means that the second part of the "replace" string has been omitted or that the syntax for the merge command is incomplete. Also appears when a non-numeric character appears in a syntax position that would ordinarily require a number.
Top	Indicates that line pointer is positioned at attribute 0, or the "top" of the item. (An "i" ("insert") command issued here will insert before attribute one).

Wrapup error messages:

'id' deleted.	(error message item-id = 222)
'id' exited.	(error message item-id = 220)
'id' filed.	(error message item-id = 221)

Example

```
014 print customer.id
.i print
.12
012 print current.balance
.i
seqn?

The most common error message is the "seqn?" message. It indicates that a
change is being attempted to a line "above" a line in which a change has been
made. To correct this, use the "f" ("flip") command, then try the same
operation that previously failed.
```

wildcards

see editor command r

<return>

advance the line pointer to the next line and displays the line if in command mode. If in insert mode and at a blank line, it returns to command mode.

Syntax

<return>

<linefeed>

<ctrl>+m

<enter>

Example

```
013 crt " enter response " :
.<return>
014 input response

015+print "hello"
016+<return>
. .
```

?

displays the current file name and item-id of the item currently being edited and displays the line number where the line pointer is positioned.

Example

```
.?
bp enter.invoice L 014
.

This indicates the current item is from the filename "bp", the item-id is
"enter.invoice" and the line pointer is positioned on line 14.
```

a

repeats the last "l" (locate) command issued in the line editor.

Example

```
.l/print
014 print customer.id
.a
020 print

The line pointer in this example is left at line 20.
```

as

toggles the assembly format on or off. It is used exclusively on D^3 assembler source listings to format the instructions and operands into a more readable format.

The 4 fields are: the label, the op-code/operand, the operators, and the optional comment.

b

positions the line pointer to the "end" ("bottom", or last line) of the current item.

Example

```
067 * Process next value
.b
eoi 167

The "end" of this example occurs at line 167. If a new line is entered, it
becomes 168.
```

c

displays a column position guide to indicate the actual positions of characters on the screen for use with the "l" (locate) and "r" (replace) commands.

Displays terminal columnar positions.

Example

```
.c
         1         2         3
12345678901234567890123456789012345678901...

The column numbers extend to the 75th column position.
```

de

deletes one or more lines from the current item, starting at the current line number position.

If a string is specified, only the lines containing the string are deleted.

The "start.col" and "end.col" parameters respectively indicate the beginning and ending column ranges in which string comparisons are made. See the "c" command for determining column positions.

A "de" command ending in a delimiter displays the lines as they are deleted. Note that the line pointer is not incremented.

Lines are not actually deleted until the item is "flipped" with an "f" command. See the "x" ("oops") command to cancel the "de".

Syntax

de

de#lines{/}

de#lines/string

de#lines/string/start.col-end.col

Example

```
.de<return>

This deletes the current line.

.de10<return>

This deletes the current line and the next nine lines.

.de10/<return>

This deletes and displays the current and next nine lines.

.de10/abc<return>

This deletes lines containing the string, "abc", in the current and next nine
lines. Any lines not containing this string will not be deleted.
```

```
.de10/abc/5-9<return>

This deletes and displays the lines containing the string, "abc", in column
ranges 5 through 9, in the current and next nine lines. Any lines not
containing this string, starting at column position 5, will not be deleted.
```

ex

exits the current item without saving changes; control returns to TCL, or to the next item in the list, if more than one item was requested. See also the "exk" command.

If the item has changed since the editor was invoked and the "ex" command is issued, the message "item changed, exit it (y/n=cr) ?" appears. A response of "y" exits the item without saving changes. To save the item, see the "fi" or "fs" commands.

Example

```
ed bp *
first.program
top
.ex
'first.program' exited.
second.program
top
.ex
'second.program' exited.
```

exk

exits the current item without saving changes and returns control directly to TCL, abandoning any active list, if present.

Example

```
ed bp *
first.program
top
.exk
'first.program' exited.
```

f

toggles the dual editor buffers so that additional changes may be made to an item, or for reviewing previous changes.

After the "flip", the line pointer is positioned to the beginning (top) of the item.

Changes to items must be made from the "top down". This means that changes may not be made to any line above a line in which a change has already been made without previously using the "f" command.

As a general rule, use the "f" command whenever changes are required above the last line in which a change was made, or upon receiving a "seqn?" error message.

Example

```
014 print customer.id
.i print
.12
012 print current.balance
.i
seqn?

The most common error message is the "seqn?" message. It indicates that a
change is being attempted to a line "above" a line in which a change has been
```

```
made. To correct this, use the "f" ("flip") command, then try the same
operation that previously failed.
```

fd

deletes the current item; control returns to TCL, or to the next item in the list, if more than one item was requested.

To recover the item, use the TCL "recover-fd" command. This command must immediately follow the "fd" command.

Example

```
ed bp print.invoices.bak
top
.fd
'print.invoices.bak' deleted.
```

fdk

deletes the current item and abandons an active list, if present. Control returns directly to TCL.

Example

```
ed bp *
first.program
top
.fdk
'first.program' deleted.
```

fi

files the current item, saving all changes made.

Control returns to TCL, or to the next item in the active list.

A different file.reference and/or "item-id" may be specified prior to filing the item.

Syntax

fi

fi item-id

fi(file.reference

fi(file.reference item-id

Example

```
ed bp program1<return>
top
.fi<return>
'program1' filed.

This files the item, with any changes made, over the orginal item.

ed bp program1<return>
top
.fi new.program1<return>
'new.program1' filed.

This files the item, with any changes made, under a different item-id. The
original item remains unchanged.
```

fio

files the current item and overwrites, when a duplicate item-id already exists on file.

This form is used when specifying an item-id other than the one originally requested.

Syntax

fio item-id

fio(file.reference

fio(file.reference item-id

Example

```
ed bp program1<return>
top
.fi program2<return>
'program2' filed.

Any changes made to "program1" in this example are saved in
"program2" only. No changes are saved to "program1".

ed bp program1<return>
top
.fi new.program1<return>
cmnd?

This does not actually mean that the command was invalid. Rather, it means
that the item, "new.program1", was already on file. It can be overwritten as
follows:

.fio new.program1<return>
'new.program1' filed.
```

fs

"saves" a copy of the current item and remains in the editor within the current item.

It is essentially the same as the "fi" command, but it doesn't exit the item.

When editing large items, the "fs" command should be issued periodically to ensure an update to the file. A different file.reference and/or item-id may be specified prior to actually saving the item.

See the "fso" command for writing over pre-existing items.

Syntax

fs

fs item-id

fs(file.reference

fs(file.reference item-id

Example

```
ed bp main.program<return>
top
.fs<return>

This writes a copy of the item back to the file from which it was originally
retrieved.

ed bp main.program<return>
top
.fs main.program.original<return>

This effectively makes a backup copy of "main.program", calling the new copy
"main.program.original".  This is useful in making historical copies or saving
```

```
originals under different names prior to making changes without leaving the
Editor.
```

fso

saves the current item and overwrites the item when a duplicate item-id already exists on file. This form is used when specifying an item-id other than the one originally requested.

Syntax

fso item-id

fso(file.reference

fso(file.reference item-id

Example

```
ed bp main.program<return>
top
.fs main.program.original<return>
cmnd?

This does not actually mean that the command was invalid. Rather, it means
that the item, "main.program.original", was already on file. It can be
overwritten as follows:

.fso main.program.original<return>
```

g

positions the line pointer to a specific line.

The attribute number (line number) may be entered withour the "g" command.

Syntax

{g}line.number

Example

```
g25
```

i

places the editor into "insert mode" for entering one or more lines.

Issuing a <return> at the first position of a line while in insert mode returns control to command mode.

Syntax

i string

i<return>

Example

```
Inserting a single line:

 .i string<return>
-or-
 .i<return>
nnn+string<return>
nnn+<return>

Note the space between the "i" command and the text being inserted.
Without it, this will not work.

Inserting a "null" line:
```

```
 .i <return>

Note the space following the "i" command. Without it, this will not work.

Multiple lines may be inserted, if each text "string" (new line) is delimited
with an attribute mark (<ctrl>+{shift}^). (The shift is only required when the
up-arrow keyboard character is designated as a "shifted" character):

 .i newline1^newline2^newline3<return>
```

l

lists a specific number of lines and increments the line pointer to the last line listed.

The "p" (or "p0") command defaults to "l22" ("list 22 lines") each time the editor is invoked.

The line pointer is incremented to the last line listed.

Syntax

l{ }#lines

Example

```
014 print customer.id
.l3
015 print
016 print customer.name
017 print customer.address
```

l

searches for one or more occurrences of a text string within a specified range of line (attribute) numbers.

If no #lines is entered, the next occurrence is located and displayed.

The ending delimiter is required when the string contains trailing blank characters.

If the "#lines" parameter is specified, the requested number of lines are searched, and those that contain a match are displayed.

The line pointer is incremented by the specified number and may therefore not be positioned at the last line displayed.

The "column", "start.col" and "end.col" parameters respectively specify the beginning and ending column ranges within which strings are eligible for matching. (see the "c" command).

The up-arrow (^) character is used as a "wildcard" search character within the string parameter for locating or replacing variable strings within strings. It matches any character.

The colon (:) is a special delimiter that indicates a column-dependent correspondence.

Syntax

l/string

l#lines/string

l#lines/string/column

l#lines/string/start.col-end.col

Example

```
.l/abc<return>

Locates and displays the next occurrence of the string, "abc".
```

```
.l99/abc<return>

Locates and displays the string, "abc", in the next 99 lines.

.l99/abc/1-5<return>

Locates and displays, in the next 99 lines, the occurrence of the string,
"abc", in column ranges 1 through 5.

.l99/a^c<return>

Locates and displays, in the next 99 lines, any line containing a string
beginning with the letter "a", followed by any character, and ending with the
letter "c".

.l:abc :<return>

Locates the next line containing the string, "abc", followed by one blank
character in the first four positions of the line.

.l999/^<return>

Locates any attributes in the next 999 lines which contain any data.
```

m

used exclusively on D3 assembler source listings to expand included macros into a more readable format.

me

merges one or more lines from the current item, another item in the current file, or another item in a different file, beginning with line number "start.attr" in item-id.

Any delimiter, except a left parenthesis, specifies that the following parameter is an item-id.

A left parenthesis specifies a different file.reference.

Default specifications:

No "#lines" merges 1 line.

No item-id uses the current item-id.

No item-id following file.reference means current the item-id

No starting attribute (line) number ("start.attr") merges from the first line.

Syntax

me{#lines}/{item-id}{/{start.attr}}

me{#lines}(file.reference {item-id{) start.attr}}

Example

```
.me10//<return>

Merges 10 lines from the current item, beginning with line number 1.

.me10/abc/3<return>

Merges 10 lines from the item called "abc", beginning with line number 3.

.me22(appdoc prog.doc)10<return>
```

```
Merges 22 lines from the item called, "prog.doc", which resides in the file,
"appdoc", beginning with line number 10.
```

n

positions the line pointer down a specified number of lines from the current position.

An "n" command, followed by a <return>, redisplays the current line.

"n1" ("n-one") is the same as the Editor "<return>" command.

Syntax

n{number.lines}

Example

```
014 print customer.id
.n4
018 print
```

o

sets object assembly mode in the editor.

Syntax

o

p

either defines or executes a "prestored" command; "p" by itself assumes "p0" (zero).

Defining prestored commands:

An editor command sequence may be assigned to a specified number (in the range 0-9). Each prestored command may contain up to 100 bytes. When no number is specified, zero (0) is assumed. The <escape> key, is used as a command delimiter to separate multiple editor commands.

Prestored editor command sequences must be reloaded each time the editor is initiated.

The default prestore, P0, is "l22", or list 22 lines.

Executing prestored commands:

To execute a prestored command, the "p" command is issued along with the desired number of the command to execute: p{number}

The "p" command, followed by a <return>, executes the command sequence stored in "p0" (zero).

The "pd" command displays the current prestored commands. Its a good idea to review prestored commands after defining them and before executing them.

Syntax

p{number} editor.command{[editor.command}

Example

```
1)  .p1 b[u22[l22<return>

This defines prestored command "p1" with the following command sequence:

1)  go to "bottom" of item
2)  move the line pointer "up" 22 lines
3)  list 22 lines.
```

```
Each time the "p1" command is issued, the "last" 22 lines of the item are
displayed.

Note that the "[" character displayed in BOTH examples is the character echoed
to the screen when the <escape> key is
pressed. This further assumes that the <escape> key is NOT
currently programmed to "push" a level.

2)  .p2 ru9999.abc.xyz[fi[p2<return>

This defines prestored command "p2" as:

1)  replace, in the next 9999 lines, all occurrences of "abc" with "xyz".
2)  file the item
3)  repeat the process on the next item (by calling the "p2" command), or, if
all items have been processed, the process falls out to TCL.
```

pd

displays any prestored command.

Prestore commands are lost each time the editor returns to TCL.

Example

```
ed bp print.invoices
top
.pd
P0 L22

See the "p" command for defining prestored commands.
```

r

replaces the current line, or replaces a string of characters within the current line, or replaces a string of characters within a specific range of lines.

Only the first occurrence of the string referenced in "old.string" is replaced with the string referenced in "new.string". The "ru" form replaces all occurrences.

If the "#lines" parameter is specified, the line pointer is incremented accordingly.

An "r" command, followed by a <return>, allows the replacement of an entire line. Entering a <return> at the first position of the line returns control to the editor command prompt and the line remains unchanged.

The "start.col" and "end.col" parameters respectively indicate the beginning and ending column ranges in which strings are eligible for replacement. See the "c" command for determining column ranges.

If the "old.string" parameter is null, the "new.string" parameter is inserted at the beginning of the line. If the "new.string" parameter is null and the "old.string" parameter is found, the "old.string" parameter is deleted from the line.

Multiple replacements may be performed on a line without "flipping" the buffers with the "f" command.

Wild cards:

The "up-arrow" (^) character is used as a "wildcard" search character within the string parameter, for replacing known and sometimes unknown strings within strings. It matches any character. Note that the up-arrow character is a normal character, not a control character. See the example: .r/^^^//<return>. See also the "^" command.

Using attribute marks in a replace:

The <ctrl>+{shift}^ represents an attribute mark in the editor. Used with the replace command, it has the effect of terminating the line. See the example: .r/abc/^<return> below.

Syntax

r

r{#lines}/old.string/new.string{/}{start.col{-end.col}}

ru{#lines}/old.string/new.string{/}{start.col{-end.col}}

Example

```
.r<return>
nnn _

Positions cursor at the beginning of line "nnn" for replacement of the entire
line. A subsequent <return> leaves the line intact.

.r10<return>

Positions cursor at the beginning of each of the next ten lines for entire
line replacement.

.r/^^^//<return>

Replaces the first 3 characters (wildcards, or "regular" up-arrows)
of the current line with null. (i.e. Deletes first three characters).


.r/abc/^<return>

Replaces the first occurrence of the string, "abc", with an attribute mark
(<ctrl> or <ctrl>+<shift>^), terminating the
line prior to the string, abc.

.r10//abc<return>

Places the string, "abc" at the beginning of the next ten lines.

.r10/abc/xyz<return>

Replaces the first occurrence of the string, "abc", with the string "xyz" in
the current and the next nine lines.

.ru10/abc/xyz<return>

Replaces all occurrences of the string, "abc", with the string, "xyz", in the
current and next nine lines.

.r10/abc/xyz/5-9<return>

Replaces the first occurrence of the string, "abc", with the string "xyz", in
the current and next nine lines, when the string is found
in column positions 5 through 9.

.ru/^//<return>

Replaces all characters on the current attribute with "null", leaving
the current attribute intact, but empty.

.ru999/  / /<return>
```

```
Replaces every occurence of two spaces with one space, from the current line
through the end of the item.
```

ru

Universal replace.

s

toggles the display of editor line numbers, in normal editor mode, and suppresses the display of object code when the assembly formatter is on.

Example

```
014 print customer.id
.s
suppress on
.l3
print customer.name
print customer.address
print customer.city
.s
suppress off
.l2
018 print customer.state
019 print customer.zip
```

s?

displays the size of the current item in bytes.

Example

```
.s?
items size is 3187 bytes
```

sl

sets the number of lines, rather than attributes, to display by the default "p" ("prestored") command, "p0" ("zero").

Syntax

sl {number.lines}

Example

```
.sl 22

This sets the display to 22 lines.
```

t

positions line pointer to the beginning (top) of the item, without flipping buffers.

The "f" ("flip") command also positions to the top.

Any lines inserted at the "top" of the program are placed before line 1 of the item.

Example

```
014 print customer.id
.t
top
.  .
```

tb

defines tabstops for use by subsequent uses of the <tab> key.

Each <tab> embeds a specific number of spaces in the current item.

Each argument is separated by a space. The maximum is 16 tabstops. When the <tab> key is depressed (or <ctrl>+i) while in "insert" mode, the cursor is positioned to the next defined tabstop. The <tab> key has no effect when positioned beyond the last tabstop. See also the "tabs" (TCL) command.

Syntax

tb tabstop{ tabstop...}

Example

```
.tb 3 6 9 12 15 18 21<return>
```

u

moves the line pointer "up" a specific number of lines from the current position.

The "u" command, followed by a <return>, redisplays the current line.

Syntax

unumber.lines

Example

```
014 print customer.id
 .u3
011 * begin main logic
```

x

cancels the effect of the last insert ("i"), delete ("de"), merge ("me"), or replace ("r") command.

The "x" command will not work after multiple replacements within the same line.

The "xf" command cancels the effect of all commands executed since the last "f" ("flip") command was issued.

xf

Undoes commands back to last (f)lip.

z

defines zone display limits. The display is limited to only the data between the specified beginning ("start.col") and ending ("end.col") column ranges.

If both parameters are omitted, the zone function is reset to display the entire line.

Syntax

z

zstart.col-end.col

Example

```
014 print customer.id
.z 3-8              <- this establishes "zone" limits.
.14                 <- go to line 14 and display.
014 int cu
.z                  <- this cancels "zone" limits.
.14                 <- go to line 14 and display.
014 print customer.id
```

^

toggles the function of the "wildcard" search character used in (l)ocate and (r)eplace commands. Initially, this function is "on". When toggled "off", the "^" is treated as any other character. This allows the "r" command to look for a literal "^", rather than treating "^" as a wildcard.

System Files

abs file

contains information about the binary executable object module which is the D3 System.

The dictionary of this file contains a binary item "%abs%" which contains the actual code. On the account 'dm', the "%abs%" item is an empty regular item, to indicate that the code associated to that file is in fact the boot abs. The boot abs is not actually part of the file system.

acc

Synonym for "accounts" file.

accounts

contains a history of computer usage by user.

To capture accounting data for a specific user add the character "a" to the "options" attribute (9) in the "users" file.

An item exists in the "accounts" file for each user. The item-id from the "users" file is the item-id in the "accounts" file.

Accounting may be suppressed system-wide by removing the "accounts" file.

Accounting information is gathered from the "pibs" file when the user logs off.

"acc" is a synonym of "accounts".

The attributes in the "accounts" file are:

0 user Item-id consists of "users" id.

1 md Name of md (item-id in the "mds" file) for the accounting period.

2 l-date Log date of the accounting data.

3 l-time Log time of the accounting data.

4 acct-time Accounting period to the second.

5 cpu Central processing unit (cpu) execution time (in tenths of seconds) used by this process for the accounting period.

6 pages Number of printer pages output for accounting period.

block-convert

contains the instructions used by the TCL verb "block-print" to display enlarged characters.

Each master dictionary has a "q-pointer" to "block-convert" that is created when the account is created.

"bc" is a synonym of "block-convert"

bp file

contains FlashBASIC source programs for system utilities.

Verbs in the "newac" file that are cataloged FlashBASIC programs point to this file.

bc

"q-pointer" to the "block-convert" file.

bulletin

contains company messages to display at logon time.

ac.. name description.....

1 from Contains the employee who created this bulletin.

2 title Contains the title for this bulletin.

3 message Contains free-form message.

cap-file

captures the screen output from the "converse" command.

The item-id used consists of "user-id*port.number".

clashes

holds "invalid" items removed from the "md" by the program "update-md". Each account updated gets its own data section in the "clashes" file. The command, "lf dm,clashes," displays these data sections.

country

defines the postal codes of countries, as part of the zip code processing code. See "zc".

devices

contains an item for every terminal or printer device recognized by the D3 System. The actual escape sequences produced by "@" functions in FlashBASIC and PROC are defined within these device drivers.

The item-ids are device codes (e.g., WY-100 for the WYSE WY-100 video display terminal). The "define-terminal" command is used to define additional terminals and printers.

The attributes in the devices file are:

ac.. attribute-name.. description.....

0 1 Full name and model number information for device.

1 pitch Character pitch. This is the number of characters per horizontal inch of printout. Valid responses are 10, 12 and 16.

2 points Character height, in points. The smaller the number, the smaller the character. Valid responses are 7, 8, 8.5, 10 and 12.

3 lpi Number of lines per vertical inch of paper. Valid responses are 6 or 8.

4 lpp Number of lines per page.

5 orient Printer orientation. Valid responses are: "P" Portrait mode; printer prints across the width of the page, short ends at top and bottom. "L" Landscape mode; printer prints across the length of the paper, short ends on sides.

6 laser.ok Contains "Y" if parameters are correct.

devs

defines the various devices in the system.

This file, located on the 'dm' account, has two data levels:

init This data level contains static information, which can be changed by the System Administrator, to set up the various entities in the system. For example, the initial baud rate of the serial ports, whether 'tandem' is allowed on the D3 ports or not, can be specified in this data level. It contains an item for all possible elements in the system, even if they are not physically present.

devs This data level contains dynamic information. The TCL command ':reset-async', normally run at boot time in the 'system-coldstart' macro, clears the file 'devs,devs', and copies the information from 'devs,init' into 'devs,devs'. Only the physically present system elements are represented in this file. The fact or writing an item into this data level actually programs the device. Items in this file may be added or removed by the "dev-make" or "dev-remov" TCL commands (D3 Unix only).
The item-id's in this file are the entity ids, which are composed of a one letter code ('p' for D3 ports, 's' for serial device, 'e' for Ethernet, etc...), followed by the entity number in decimal (the D3 port number, the serial device number, the TCP/IP connection id, etc...). See the section "entity" in this document for more information.

Both data levels can be examined using AQL or the Update processor.

The structure of an item in these files is described in the include 'dm,bp,includes qcb.inc'.

Example

```
u devs,init s12
 Examine the characteristics of serial device 12. This brings up the Update
processor, and allows changing the serial device initial setting. Filing this
item does not actually change the setting. It will be changed the next time
the command ':reset-async' is run (normally at boot time) or by running the
command ':reset-async s12'

u devs p0
 Examine the characteristics of D3 port 0. The only element than can be
changed is the 'tandem' flag, which controls whether tandem is allow or not on
port 0, and whether the target process should be notified about the tandem
on/off. See "tandem" in this document.

sort devs = "s]"
 Show the current status of all serial ports.
```

dm-bp

"q-pointer" for the "bp" file on the "dm" account.

dv

"q-pointer" for the "devices" file.

er

a "q-pointer" for the "errors" file.

errmsg

a "q-pointer" to the "messages" file.

errors

contain system error messages used to determine how the system is functioning.

The errors file contains error and "log-msg" messages. See "log-msg".

The item-ids are nine-digit date-time stamps (four-digit date plus five-digit time) plus an alpha character for multiple creations within a second.

The specific types of errors which generate messages are:

1) System errors: illegal opcode, crossing frame limit, forward link zero, backward link zero, priv opcode, illegal frame reference, disk error, stack format error, overflow release error, gfe, index B-tree error.

2) "logoff" commands.

3) Errors from the TPS (test procedure suite) on the QA account.

The attributes for the "errors" file are:

ac attribute-name description.....

0 date Date error was logged.

0 timedate Item-id in errors file consists of date and time error was logged.

1 error Error code. Possible errors are:

01 phantom read

02 rtn full

03 zero fid

04 frm lmt

05 fwd link

06 bck link

07 priv op

08 bad fid

09 disk err

0a break

0b stk fmt

0c lvl push

0e end off

0f trace

11 gfe count

12 gfe link

13 gfe body

14 gfe hash

15 b-tree index

16 ovf mul rel

17 end

18 unused

19 logoff

1a overflow runaway

1b ovf abort
20 ovf signature error
21 ovf link error
22 fcb corruption
23 (unknown)
24 (unknown)
25 CLEAR-LOCKS performed
p tps program error
s shutdown
2 user The user of the error logging process.
3 r Register triggering the error.
4 mode-addr Mode address when the error occurred.
5 abs-fid Base fid of the abs area executed by the process.
6 pgm-ctr Program counter when the error occurred.
7 abs-date Creation date of the abs in which the error occurred
8 abs-imp Implementation type of the abs in which the error occurred.
11 pcb-fid Process control block (pcb) frame id.
12 userpib The user and pib of the error logging process
20 registers Process register image.
21 run-stack Process execution stack.
22 last-tcl-entry The last three tcl stack entries.
23 gfe-fid Contains four values:
1) frame id of the group with gfe,
2) address (in storage register format) of the item in which gfe was detected,
3) forward link of the frame in which gfe occured,
4) backward link of the same frame
24 gfe-file The path name of the file in which gfe was found.
Disk Error
On D3 Unix, the error log for a disk error has the following format:
ee dd 0000 00 00 ffffffff xx
ee Unix error code. (value of errno)
dd Logical disk number, as defined in configuration file
ffffffff Frame number, in hex
xx Command: 00=read, 01=write
The other fields are unused. Depending on the implementation, some other information can be obtained from the Unix error log.

ff

"q-pointer" for the "file-of-files" file.

file-of-files

contains statistical information for each file on the system. Whenever a file is created, restored or deleted, this file is updated. "File-of-files" items are created when a file is restored or created.

This file is also used to locate files on the disk during restores from incremental and transaction-log tapes.

The item-ids are sequential file numbers assigned when the file is created. File number one (1) is always the "file-of-files" file and file number two (2) is always the "mds" file. The numbers of the other files are determined by the order in which they are restored or created.

On a file or account restore, the files are recreated in the order in which they were saved on the file save media.

The item "restored" in the dict of the "file-of-files" contains the time in attribute 1, and the date in attribute 2 of the last full restore. This item should not be deleted or modified.

"fof" and "stat-file" are synonyms (q-pointer) of "file-of-files".

The attributes in the "file-of-files" file are:

```
ac attribute-name description
0   f-fms       Total number of frames in the primary file space as of last
                file save.
0   f-size      Total number of bytes in the primary file space as of last file
                save.
0   file#       Item-id assigned in order in which file was created.
0   t-frames    Total number of frames in the file space as of last file-save.
0   stat.acc    Sum of all reads and writes
0   stat.ovf    Sum of all overflow group accesses
0   sug.mod     Suggested modulo for this file, based on file-of-files
                information.
1   md          Name of the md owning the file.
2   file-name   File name stored in the master dictionary (md) of the account.
3   data-name   One dictionary may have many subsidiary data files each with
                its own unique name. The default data section name is identical
                to the file name as stored in the md.
4   base        Base of file.
4   mod         Modulo of file. The number of contiguous frames comprising the
                primary file space.
4   modulo      Modulo of file. The number of contiguous frames comprising the
                primary file space.
5   items       Total number (including pointer items) of items in file from
                last file save.
6   ptr-items   Number of pointer items saved in last file save.
7   ovf-itms    Number of item which were partially or wholly stored in
                secondary file space during the last file save.
8   bytes       Total number of bytes in file as of last file save.
9   ptr-bytes   Total number of bytes in all pointer items as of last file
                save.
10  frames      Total number of frames in file as of last file save. (Does not
```

		include index frames).
11	ptr-fms	Total number of frames of pointer items as of last file save.
12	svdate	Date when file was last saved.
13	reel#	Tape, diskette, etc. number in a multi-'reel' file save where this file begins.
14	seq#	Decimal sequence number indicating the order in which the file was saved on the file save media.
15	opendate	Date when file was last opened (update at save-time).
16	stat#	Number of last file save on which this file was saved.
17	mask	Masks desired operations from being logged in attributes 18-20. Currently supported masks are: "c" clear file; and "d" delete-file.
18	file-code	Valid file statuses are: "c" clear file; "d" delete file; "n" new file; "r" rename file; "t" restored from tape. This attribute controls attributes 19 (timedate) and 20 (user).
19	timedate	Time-date when file activity occurred. This attribute is dependent on attribute 18 (file-code).
20	user	User id concatenated with account id causing the associated file action. This attribute is dependent on attribute 18 (file-code).
21	dx/dy-date	Date when dx/dy file was skipped on save.
25	save-list	Contains list of specific items to be saved for this file. Used to selectively save items in a file. Each item name is a value. To save all items in a file, use an asterisk (*).
29	stat.date	Contains the date when the file access statistics were last cleared.
30	stat.rdu	Number of READU operations on the file.
31	stat.rdub	Number of blocked READU operations on the file.
32	stat.rdul	Number of READU LOCKED operations on the file
33	stat.rdulb	Number of blocked READU LOCKED operations on the file.
34	stat.rdptr	Number of pointer items read.
35	stat.rd	Total number reads on the file.
36	stat.wtu	Number of WRITEU operations to the file
37	stat.wtb	Number of blocked writes to the file
38	stat.wtptr	Number of pointer items written.
39	stat.wt	Total number of writes to the file.
40	stat.sel	Total number of selects on the file.
41	stat.dels	Total number of deletes to the file.
42	stat.clr	Total number of clear-file's done on the file.
43	stat.open	Total number of open's on the file.
44	stat.ovf	Read overflow group accesses.
45	stat.wtovf	Write overflow group accesses.
46	read-date	Last date the file was read.
47	write-date	Last date the file was written.
60	Seg.Bas	Segment bases for resized files.
61	Seg.Mod	Segment modulos for resized files.

files

"q-pointer" for the "file-of-files" file.

See "file-of-files".

fk

"q-pointer" for the "funckeys" file.

fof

Synonym for "file-of-files" file.

fonts

see "download".

funckeys

contains preset function key definition items.

This file is used with the TCL verb "set-func".

Function keys can also be part of a keyboard definition item in the "keyboards" file, selected by using the "set-kbrd" verb.

gsym

global symbol file in the data-manager (dm) account. Global symbols are used to reference the system-wide set of global variables.

Attribute 0: Global symbol name is the item-id

Attribute 1: item type code

c character

h half-tally (1 byte)

t tally (2 bytes)

d double-tally (4 bytes)

n constant

s 6-byte storage register

e 8-byte quad tally

b bit the displacement for the bit definition is the actual number of bits. Divide by 8 to find the byte the bit is stored in.

Attribute 2: variable displacement in hexadecimal.

Attribute 3: variable register number in hexadecimal.

hosts

file which defines file system connections to "remote" environments.

The hosts file is used when accessing super-Q-pointers to locate the appropriate OSFI driver and any options which should be passed to that driver.

Attribute	Purpose
0	Host identifier to be used in attribute 3 of super-Q-pointers
1	Generic File System Interface driver number
2	Driver-specific configuration information

Driver-specific information can be found in the see-also entries.

hotkey.all

specifies the name of a FlashBASIC subroutine to execute according to the rules described in the "hotkeys" Pick Systems Reference Manual token.

The name of the subroutine to call is placed in the "hotkey.all" attribute (attribute 20) in a file-defining item or attribute-defining item.

iomap-file

defines the one-to-one keyboard input and output translation used by the "set-iomap" command. The data section of the file contains items that define the character translation. The dictionary level contains binary items which are compiled by the "set-iomap" program when used with the "c" option. The items in the dictionary have the same item-id's as the corresponding items in the data section, except that the (dictionary) items are preceded and followed by a "%".

Each item in the data section contains up to 33 attributes. Attribute 1 is used for comments. Attributes 2 through 33 contain data for the actual character translation. Each attribute consists of 8 characters (32 x 8 = 256, the number of characters in the ASCII set). ASCII characters 0 through 7 (in hex) are stored in attribute 2. Characters 8 through 16 are stored in attribute 3, and so on. The "set-iomap" program reads a specified item from the "iomap-file" and converts the ASCII characters into binary equivalents, then writes the item to the dictionary level of the "iomap-file".

Example

```
set-iomap 1234 (c
Assuming that there was an item in the "iomap-file" that contained the
following:

item-id:   1234

Attr#      Contents

1        this is just a comment
2        0001020304050607
3        08090a0b0c0d0e0f
.
.
.
9        6061626364656667
10       68696a6b6c6d6e6f
11       7071727374757677
12       78797a7b7c7d7e7f
.
.
.
33       f8f9fafbfcfdfeff

This translates the characters "A" through "Z" into "a" through "z". All
other characters remain the same. The resulting item, "%1234%" would then be
written to the dictionary of the "iomap-file".
```

jb

a "q-pointer" for the "jobs" file.

jobs

contains data for each phantom job submitted for execution.

The item-ids are nine-digit date-time stamps (four digit date plus five digit time). If more than one item is filed at the same time, an alpha character beginning with the letter "a" is appended to the item-id.

An item also exists in the "jobs" file for each port currently being used by a phantom job.

Job status' include:
c Done

e Error

l Logoff

q Queue

r Running

s Abort

p Sched id

The attributes in the "jobs" file are:

ac.. attribute-name description.....

0 id Item-id of the jobs file.

1 status Status of phantom job.

2 user User who submitted the job.

3 md Execution md of the job.

4 command Tcl command for job execution.

5 out Output status of the job:

n Spooler hold file number

m Message back enabled

s All output suppressed

6 who Port.number which initiated the job

7 start Date the job was actually started.

8 startt Time the job was actually started.

9 tcl-command Tcl command for job execution.

11 pcb-fid Process control block (pcb) frame id of the job.

12 pib Pib or port.number where job is running.

15 stopd Date the job terminated.

16 stop Time the job finished.

The "phantom scheduler" keeps two queues in the dictionary of the "jobs" file. They are:

%Q Contains all jobs waiting to run.

%R Contains all finished jobs not yet released back to overflow.

The dictionary of the "jobs" file also contains an item called "%P" which consists of a list of phantom pibs' status bytes.

kb

a "q-pointer" for the "keyboards" file.

kb.fk (D3/SCO)

used for mapping the keyboard on the console of D3/SCO PC-class implementations only.

kb.pc

maps the keyboard.

Syntax

see "set-func" and "set-kbrd".

keyboards

contains two types of items:

1) Alternative keyboard definitions, including British, French, German, Italian, Spanish and USA for PC type systems (D3/SCO), for the TCL verb "set-kbrd".

2) Keyboard input translation items for the TCL verb "set-imap".

The format of the keyboard input translation for "set-imap" is as follows:

- Attribute one must contain the keyword 'IMAP', optionally followed by one or more modifiers, separated by spaces. Valid modifiers are:

'esc-data'

Sets esc-data. This option should be used if any input sequence contains an escape.

'esc-level'

Sets esc-level. This option could be used if no input sequence contains an escape.

'xcs-on'

Enables the extended character set (8 bit characters).

'xcs-off'

Disables the extended character set.

'timeout' value

Sets the default timeout to 'value', expressed in milliseconds. This value can be overridden on a per-port basis by "set-imap". See "set-imap" for the meaning of the timeout and precaution about this feature.

- Each line defines a translation where the input sequence and the converted sequence are separated by a colon (':'). If the converted string is null, then the input sequence corresponds to a key which will be ignored.

"input sequence" : "converted sequence"

- Any text after a '*' is ignored, except when enclosed in quotes. Spaces are ignored, except when enclosed in quotes. Empty lines are ignored.

- Each element in the input sequence or the converted string are separated by commas.

- Characters and strings are represented by c'abc..' or c"abc...". Example:

c']'

c']A'

- Hexadecimal values and strings are represented by x'NN' or x'NNNNNN..', where each pair of hexadecimal digits are assembled into one byte. Example:

x'ff'

x'27ff'

- Decimal values are represented normally.

- Control characters are specified by a caret ('^') followed by the corresponding letter, in upper or lower case. Example:

^a

- The following keywords can be used anywhere to represent the usual ASCII codes:

BS : x'08' LF : x'0A' CR : x'0D'

ESC : x'1F' DEL: x'7F'

Example

```
The following example illustrates the IBM 3151 keyboard input definition to
use special keys in the Update processor.

IMAP timeout 50 esc-data
* IBM 3151

* Cursor movement
ESC,c'D'           : ^J          * <-
ESC,c'C'           : ^K          * ->
ESC,c'B'           : ^N          * down arrow
ESC,c'A'           : ^B          * up arrow

* Editing keys
ESC,c'Q'           : ^l          * delete
ESC,c'P ',BS       : ^w          * insert

* Function keys F1 - F12
ESC,c'a',13        : ^x,c'1'     * F1  : Hot key 1
ESC,c'b',x'0d'     : ^x,c'2'     * F2  : Hot key 2

* Function keys F13 - F24
ESC,c'!a',CR       : c'off',CR   * F13 : OFF
ESC,c'!b',CR       : c'end',CR   * F14 : END

* One key functions
DEL                : ^l         * Del

* Ignored keys
ESC,c'"A'          :            * Num Lock
```

md

abbreviation for master dictionary.

mds

is the highest level file in the D^3 file structure and resides on the dm account.

The "mds" file contains pointers to all master dictionaries (accounts) on the system. The "real" account pointers are those that are defined by "d-pointers". See "create-account".

There are also "synonym" definition items in the system file which essentially define alternate names for existing accounts.

A few "system-level" files may also be found in the "mds" file. These are files only, meaning that they have no verbs in them. An example of this is "restore-errors".

messages

contains messages used by system processes to convey information to the user.

The item-ids are the message numbers referenced by system processors.

The "bnf" item is used by the BASIC processor.

The "logon" item is used by the Logon processor.

The "legend" item is used by the Output processor.

The "newmd" item is used to create new accounts.

The "seq" item is used by the "ms" Correlative processor.

Error Message Operands:

The items in the messages file use the following codes to format messages in the output buffer before displaying them:

! Any line of an error message that begins with an "!" is ignored by the message handler. This feature allows storing comments within the message.

a{(n)} Inserts into the output buffer the next parameter from the list of parameters passed with the message. The parameter is left justified in a field of "n" blanks.

b Rings the terminal "bell".

c Clears the terminal screen.

d Places the current date in the output buffer.

e Inserts the message number (item-id in the messages file) into the output buffer.

hstring Places a literal string in the output buffer, with no carriage return or line feed.

l{(n)} Prints the output buffer, followed by "n-1" blank lines.

r{(n)} Inserts the next parameter right-justified in a field of "n" blanks.

s(n) Inserts "n" spaces in the output buffer.

t Places the current time in the output buffer.

x Skips a parameter in the parameter list.

Example

```
1202
001 HNeeds to start printers.

This outputs: "Needs to start printers".

781
001 H'
002 A
003 H' added
```

```
This outputs: "'thing' added", where 'thing' is passed into the error message
handler by the process that triggers this message.

B156
E Line
A
H illegal to ENTER a subroutine.

This outputs: "[B156] Line (number) illegal to ENTER a subroutine.", where
(number) is the actual program line number where the error occurred.
```

msgs

a "q-pointer" for the "messages" file. See "messages".

net-nodes

pointer to the "hosts" file. Note that the actual format of the "hosts" file may differ somewhat than the "net-nodes" file found on some other licensee platforms.

newac

contains all the master dictionary items which are copied to each new D3 account.

See "create-account" for creating new accounts.

pb

is a "q-pointer" for the "pibs" file.

peqs

see peqs in Spooler

pf

synonym for "pointer-file" file.

pibs

contains device specifications for each port.

A "pibs" item is required for each process executing on the system.

The items in the "pibs" file define the memory resident hooks of the process work spaces.

The "pibs" file also defines the physical devices connected to the system's ports, like terminals, printers, phantoms or communication ports. The "port.numbers" are the item-ids.

Device control information is derived from the "pibs" file and the "devices" file. The "pibs" item specifies the device type and the "devices" item defines device characteristics and drive functions.

ac.. attribute-name and description

0	pib#	The item-id consists of the pib number.
1	location	This is the physical location of the device connected to the pib.
2	device	This is the item-id (in the "devices" file) identifying the specific device on the port (e.g. wy-50 terminal). When the port becomes active, it is automatically setup to the device type.
3	status	This is the current status of the pib. It is a display-only attribute called through a program at list-time.

4	abs-fid	This is the frame number (fid) in decimal of the first frame in the contiguously linked machine code item (abs area) being executed. It is a display-only attribute called through a program at list-time.
5	pcb-fid	This is the frame number (fid) in decimal of the process control block (pcb) linked to this pib. It is a display-only attribute called through a program at list-time.
6	pgm-ctr	This is the program location and execution stack within the machine code item (abs area) where the process is currently executing. It is a display-only attribute called through a program at list-time.
7	user	This is the item-id (in users file) of user logged on to this pib. It is updated at user logon and deleted at logoff. It is used in the "accounts" file if accounting is enabled.
8	u-date	This is the date the user associated with this pib logged on to this user-id. It is updated at the same time as the user and is deleted at logoff.
9	u-time	This is the time of day the user logged on. It is updated at the same time as the user and is deleted at logoff.
10	md	This is the name of the md (item-id in "mds" file) this pib is logged on to. It is updated at md logon, and deleted at logoff. This is used in the "accounts" file if accounting is enabled.
11	md-date	This is the md logon date. It is updated at the same time as the md and is deleted at logoff. It is used in the "accounts" file if accounting is enabled.
12	md-time	This is the md logon time. It is updated at the same time as the md and is deleted at logoff. It is used in the "accounts" file if accounting is enabled.
13	cpu	This is the Central Processing Unit (cpu) execution time (in tenths of seconds) used by this process since logging on to the current account. Note that this field is valid only as of the last "charges" or "logto" done on the pib on question.
14	disk	This is the number of disk reads since logging on to the account. It is called through a program at list-time. Note that this field is valid only as of the last "charges" or "logto" done on the pib on question.
15	pages	This is the number of printer pages output since logging on to the account. It is a display-only attribute called through a program at list-time. It is used in the "accounts" file if accounting is enabled. Note that this field is valid only as of the last "charges" or "logto" done on the pib on question.
16	attdev	This is the list of attached devices.
18	linkdev	This is the linked device for the pib.

Example

```
:who
49 ge ba

:list pibs 49
Page   1     pibs                            11:17:54  17 Jan 1997

pibs link-dev att-devs location.. device.... user...... udate utime
```

```
account...
  49                    George      79,,0,1,1, ge          01/17  7:11
ba
                        Elvin       8,80,59,ib
                                    m3151

[405] 1 items listed out of 1 items
```

pointer-file

the default file containing items from "save-list" commands.

The item-id defaults to "%user-id" (percent sign concatenated with the user-id) when not specified with the list process verbs such as "save-list" or "edit-list".

The "pointer-file" resides on the "dm" or "sysprog" account. Separate "pointer-files" can be created on any "account". When using a "pointer-file" on any account other than "dm", the verbs associated with the file need to be modified to point to the correct account. "pf" is a q-pointer to "pointer-file".

"save lists" can be saved in ANY file, and "pointer-file" is a "d" type file.

psym

synonym to the gsym file.

resizing

contains information about files currently being resized.

restore-errors

contains errors which occurred during a restore. If the "restore-errors" file does not exist, it is created when the first error occurs.

The "d-pointer" to this file is placed in the "mds" file, rather than in the "dm" account md, since a full restore may be in progress, and the "dm" account may not be available.

One of the following errors may be generated:
err "d" type segment level (1=account, 2=dict, 3=data)

This indicates that either an incorrect "d-pointer" was found on the tape or an item with no preceding "d-pointer" was found on the tape. This type of message usually occurs during unusual restore sessions, such as forcing tape to get past errored blocks, starting on wrong reel number's, overriding or continuing after bad labels, etc.

If an item is found with no preceding "d-pointer", the restore must be told where it should be placed. For example: If the restore was restoring the data level (level = 3) when a block was dropped or bad and at the beginning of the next block an item was found that didn't belong to this file, the restore will not know where it goes. If 2 is chosen (dict) in the above example, the restore stops restoring items to the data level and begins restoring items to the dict level of this file.

If a "d-pointer" is read off the tape and the level number is not in the range 0 to 3, where zero is the "mds" file, the restore processor must be told at which level it is to be placed. If a "d-pointer" update is read off an incremental or txlog tape and the file number exists in "file-of-files" but it's the wrong file control block, then this message will be displayed.
tape format err (segment skipped)

If an item is read from the tape and its associated file "d-pointer" has not previously been read, it will been skipped. Or, if the item read does not conform to D3 tape format it will be skipped, especially if binary object code is not in D3 format. If an index b-tree is not in the correct format on the tape this message may also appear.
not found in the "file-of-files"

This is more of a warning, the file number was not found in the "file-of-files". It creates one.
obj data err

If the object code is not in the proper format, then blocks of it will be skipped. If there is no header, which details how many blocks are to follow, than all blocks will be skipped.
'xxx' is not a file name

Could not find the file in the "file-of-files"

'xxx' error in rename, segment skipped

The restore was probably from incremental or txlog tape. File not found in "file-of-files" and could not be renamed because restore does not know the path name of the old file. This is one reason why incremental saves can't be restored from TCL. (Too much reliance on file number rather than path name) The "rename-file" or "move-file" was skipped.

stat-file

see file-of-files

state

lists the states, counties, provinces of various countries, as part of the zip code processing code. See "zc".

sysprog-bp

synonym ("q-pointer") to the "dm,bp," file.

sysprog-pl

synonym ("q-pointer") to the "dm,bp," file.

system

is a "q-pointer" for the "mds" file.

system-errors

"q-pointer" for the "errors" file. See "errors".

tcl-stack

keeps a record of all commands issued from TCL.

Normally, the item-id of the "tcl-stack" file is the "users" id. If a T option is included in the options attribute of the users item, the item-id of the tcl-stack is the line number of the process logged on.

A "users item" must be created in the "users" file prior to any user logging on to the system. However, "tcl-stack" items are created automatically when the user executes the first command from TCL.

Each and every unique command typed in at the TCL prompt character (:) is saved as an attribute in the "tcl-stack" file. Having unlimited item size means that there is no limit to size of the items in the tcl-stack. Therefore, the file size should be monitored by the System Administrator.

Because the "tcl-stack" file is on the "dm" account, it should be saved ("t-dumped") prior to an operating system upgrade.

ts

"q-pointer" for the "tcl-stack" file. See "tcl-stack".

tv.log

records t-verify errors when the option is chosen to verify the tape during a "file-save".

This file is created, if necessary, by the "file-save" process. Each "t-verify" error detected is logged into "tv.log".

ulk

references attribute 5 of a file-defining item describing the file's update-lock codes.

Example

```
list md with ulk = "MIS"

Lists all items in the current account which have  "MIS" as a value in
attribute 5.
```

users

contains specific data relevant to each authorized user on the system.

Each user should have a unique logon item-id in the "users" file. The tcl-stack item for each user has the same item-id as the logon item-id in the "users" file. The simplest means of updating user items is to use the Update processor from the dm account, using the command u users item-id, where the item-id is a new or existing item-id in the "users" file. The attributes in the "user's" file are:

0 (item-id) Usually the users initials.

1 Name

2 Address

3 Zip

4 Phone

5 Unused

6 User's retrieval and update keys. See "retrieval locks".

7 Optional logon password(s). See "password".

8 Privilege level. See "system privileges".

9 User session options:

a	Updates accounting history file at logoff.
c	If this option is set, then any entry into the virtual debugger will comatize with a message indicating that the user should contact the system administrator. The administrator can then log the process off (with "logoff") or let it continue (with the "toggle.coma" command).

i Indicates that the item-id of the tcl-stack accessed will be the same as the pib number.

m Invokes GFE handler on any GFE encountered. See "gfe handler".

p Causes phantom processes initiated by this user-id to terminate and wrap up, rather than simply going to sleep when a GFE is encountered.

r Restarts account logon procedure or macro on any <break>.

t{"command"} Restarts account logon procedure on any attempt to go to TCL. An optional TCL command may be specified in quotes immediately after the "t" option to force execution of an alternate command in this situation. This TCL command may contain spaces and/or options, but cannot contain other double-quotes.

10 unused.

11 unused

12 User logon macro. This is the same as a TCL macro without the macro type code, (in attribute 1) which is assumed to be 'n' (for "non-stop"). Any number of TCL commands may be placed on attributes 12 and beyond (multi-valued). Each command will be executed before the user is logged on. Typical logon commands might include "logto master.dictionary", "sp-assign form.queue", "brk-level", "bulletin.board" and so on.

Example

```
:up md,users,  ge

users  'ge' size=318

name George Elvin
address Shipping & Receiving
zip
phonex.449
keys
password
privilege  sys2
options n
macrologto dm
10 set-sym gsym
11 brk-level
12 bulletin.board new
13 tcl-hdr-off
```

var

pseudo-file which maps to the user's TCL shell variables.

See the "var" OSFI driver definition for more information.

vi.dump

is created and used by the "verify-index" process as a temporary work file. It is created as a "dx" pointer, so it is not saved during backup.

words

contains all the valid words used by the UP (Update processor) spelling checker.

This file resides on the "dm" account. An index must be created for attribute 0 of the words file in order to activate the spelling checker.

wsym

used by the "x" option of the "OP" command, this file contains the results of counting each word in a document, along with the page number and line number of each word.

zcf

(zip code file) is used as part of the zip code processing code.

Output Processor

formats and prints text from commands embedded in the textual data.

Utilizing OP capabilities, data can be extracted directly from files within the data base, formatted, and incorporated in the text of a document without special manipulation.

Other capabilities supported by OP include:

Proportional spacing (both horizontal and vertical)

Text formatting

Font control

Automatic Index and Table of Contents generation

Boldfacing

Underlining

Headings and footings

Automatic numbering of chapters, sections, figures, and tables

OP treats each attribute in an item as a paragraph. Commands may be embedded anywhere in the text or data.

An OP command is preceded by a space and a period (the period is part of the command) followed by the characters that make up the command. OP commands end with a space, another OP command, or an attribute mark. OP commands are not shown in the formatted and printed text.

Execution of OP commands can be suppressed by utilizing the override character. The default override character is the underscore (__). When the underscore is placed before the period in the dot command, the command is not executed, but output like normal texual data. To change this character, use the .oc command. If the underscore is to be used in the text, then the override character should be changed.

Several OP commands are used to select the characteristics of the printer. To use these commands effectively, you must be familiar with the features and limitations of the printer. Before using the D^3 release of OP, the printer must be described as an item in the devices file. The information in the item includes the sequences for underlining, boldfacing, tabbing, font selection, and other printer parameters. In addition, if the printer has multiple font capabilities, up to fifteen generic fonts can be described. The D^3 release of OP makes it easy to use the printer's font capabilities. Fonts may be changed anywhere in a line and as many times in a document as needed. To set up a form queue with fonts for a printer device, use the assignfq verb.

Throughout the OP chapter, the values for all commands that measure horizontal space are given in tenths of an inch; therefore, when a command is described as inserting n spaces, it is actually inserting n /10 inches. For example, if the command .lm 10 is entered to set the margin 10 spaces,10" reality the margin is being set to 10/10 of an inch or 1 inch.

Syntax

op file.reference {item.list} { (options) }

Options

n{-m} to print a single page, enter the page number. To print a range of consecutive pages, enter the beginning page number, a dash or a dot, and then the ending page number.

c Suppress chain and read commands.

d Double space document.

f Suppress footings.

h Suppress headings.

j Suppress highlights.

k Suppress legends.

n Do not pause when output to terminal.

p Send document to printer.

s Suppress boldface, underline and fonts.

t Triple space document.

x Count word usage into WSYM file, format word:page number:line number.

Example

```
op doc test.item (pfh
```

margins, Output Processor

defines the boundaries in which OP places text.

OP commands

defines the general format of OP commands.

<space><period><command><space or next OP command>

preceded by a space and a period (the period is part of the command) followed by the characters that make up the command and ends with another space, another OP command, or an attribute mark.

Most OP commands also have a shortcut syntax. The shortcut is referenced with each command.

*

treats the entire line following the command as a remark.

Any text following the command is ignored by OP.

Syntax

.* {text}

Example

```
.* This is comment text.
```

appendix

alphabetically assigns letters to appendix titles and creates entries in the Table of Contents.

Syntax

.appendix title

.apx title

Example

```
.appendix ASCII codes
.apx Device Types
.ptoc
Prints two appendices and a Table of Contents.
```

apx

see "appendix".

bc

See "block center".

begin page

forces a page break.

A page break terminates the current page, prints the optional footing, ejects a page, increments the page counter, and prints the optional heading.

The text immediately following the ".begin page" command is printed on the next page.

Syntax

.begin page

.bp

Example

```
This is page one.
.begin page
This is page two.
.bp
This is page three.

This outputs "This is page two." on the second page and "This is page three."
on the third page.
```

bf

see "boldface".

bk

see "break".

block center

centers all of the attributes following the command until an ".xblock center" command is issued.

Syntax

.block center

.bc

Example

```
.block center
Heading One
Heading Two
.xblock center
Summary of points.

This causes "Heading One" and "Heading Two" to be centered. "Summary of
points" is not centered.
```

boldface

prints text in boldface until an ".xboldface" command is encountered.

If the printer does not have a boldface font, the text is double struck. On terminals boldfaced text is often displayed as reverse-video.

Syntax

.boldface

.bf

Example

```
To emphasize the word "primary", use  .boldface primary .xboldface
or  .bf primary .xbf .

This example causes the words "primary" to be output in boldface mode.
```

box

draws a box at the column positions specified in the command and overwrites anything which occupies those positions.

The starting column position is calculated using the current left margin as the starting point. "pos1" is the number of characters to offset from the left margin.

If "pos3" and "pos4" are specified, a second box is drawn from column "pos3" to column "pos4" ("pos3" must be greater than "pos1", and "pos4" must be greater than "pos2:).

If no parameters are specified, the width of the box is determined by the left and right margins. The box is drawn on the margin and the text margins are reduced to fit inside the box.

If "pos1" is entered, but "pos2" is not, the right side of the box defaults to the right margin. If "pos3" is entered, but "pos4" is not, the right side of the box defaults to the right margin.

All text is "boxed" until an ".xbox" command is encountered.

If two ".box" commands are specified without an ".xbox" command in between, the second ".box" command terminates the first box and begins a new box.

Syntax

.box {(pos1,pos2{(,pos3,pos4)})}

Example

```
.nf
.box 6,16
Note: Example A is shown on page n1.
.xbox

The words "Example A" are enclosed in a box.

.box 6,16,43,53
Note: Example A is shown on page n2 and Example B is shown on page n3.
.xbox

The words "Example A" and "Example B" are enclosed in boxes.
```

bp

See "begin page".

break

creates a new paragraph.

The ".break" command, surrounded by at least one space, can be placed anywhere in the text.

Syntax

.break

.bk

Example

```
This is text .break Example

Produces the following output:

This is text
Example
```

c

see "center".

cap sentences

capitalizes the first letter of each sentence or attribute.

The beginning of a sentence is considered to be the first letter of each attribute. The first letter after (". "), ("? "), or ("! ") is also considered the beginning of a sentence, and is therefore capitalized.

If the first word of a sentence is already capitalized, it remains capitalized.

Sentences are capitalized until an ".xcap sentences" command is encountered.

Syntax

.cap sentences

.cs

Example

```
.cap sentences
this is the first sentence.  this is the second. And, this is the third.

The output from this document is:

 "This is the first sentence.  This is the second. And, this is the third."
```

center

centers the single line of text that follows the command.

This command causes a ".break" to occur.

This command can be anywhere in the text. It does not need to be turned off.

Syntax

.center

.c

Example

```
.center Heading A

This produces the following:
Heading A
```

```
Phone number : .center 714 261-7425

This produces the following:

Phone number : 714 261-7425
```

ch

see "chapter".

chain

passes control to a specified item-id in the same (current) file or in a different (file.reference) file.

If "file.reference" is not specified, the current "file.reference" is used.

The {dict} and {file.reference} are both optional.

If "dict" is not specified, the "data" section of the file is assummed. If no "file.reference" is given, the item is read from the same file as the item being processed.

Control does not return to the source item after executing the 'chained' item.

Syntax

.chain {dict} {file.reference} item-id

Example

```
:u document letter
Dear  .readnext

You may have won one million ($1,000,000) dollars.

Sincerely,
John Doe
.chain letter

This command proves particularly useful for generating  form-letters. It is
possible to insert the name and address of each recipient of the letter from a
separate file.

A "sselect" statement is used to extract the relevant data from the file and
save it in a list.

A series of ."readnext" statements inserts the data into the text of the
letter.

At the end of the letter, a ".chain" command is used to restart the next
letter. When the list is exhausted, OP will stop.
```

chapter

starts a new chapter

Chapters start on new pages and the chapter counter is incremented. The literal, "Chapter" followed by the optional text is printed centered between the current margins.

The chapter, page number, and text are accumulated for the table of contents.

Syntax

.chapter {text}

.ch {text}

Example

```
.chapter
Update processor
.chapter
Output processor
.chapter

The output of this document is :

Chapter 1
Update processor

(page eject)
Chapter 2
Output processor

(page eject)
Chapter 3
```

char

sends a raw string of characters to the printer or terminal without translation.

The default delimiter is the space character.

A forced delimiter occurs at the end of the attribute.

Since neither a carriage return nor a line feed is printed, this command may be included in a sentence as a separate attribute without breaking the paragraph.

To specify unprintable characters, enter a value mark (<ctrl>+v) followed by the decimal representation.

For example, to specify the control sequence "<escape>p", where the decimal representation of <escape> is 27, the sequence is "<ctrl>+v 27<escape>p".

Note: Value marks are not displayed by the Update processor. They can be seen by using the "list-item" (TCL) command, where they are displayed as a right bracket ("]").

Syntax

.char {delimiter} character{s} {delimiter}

Example

```
.char /]27(s1010p0s3T/
```

col

uses the column headings and spacing specified with the last ".columns set" or ".variable columns" command to determine the position of the specified column number.

If the column number is less than the previous column number, the columns are printed and a new paragraph begins.

A maximum of seven columns are available.

Syntax

.col number text {.col number+1 text} {...}

Example

```
01 .columns set Name,10,5;Phone,10,1
02 .col1 Fred .col2 555-1212
03 .col1 Barney .col2 555-2121
```

```
This produces the output:

name            Phone
Fred            555-1212
Barney          555-2121
```

columns

see "columns set".

columns set

specifies the column heading, column width, and column spacing.

The columns are turned off by using the Output processor ".xcolumns" command.

Headings can contain Output processor commands and are output at the start of the columns and at the top of each subsequent page until columns are turned off.

A maximum of 7 columns can be specified.

The ".coln" command uses the column headings and spacing specified with the last ".columns set" or ".variable columns" command to determine the position of the specified column number. If the column number is less than the previous column number, the columns are printed and a new paragraph begins. A maximum of seven columns are available.

Syntax

.columns {set} heading1,width1,space1

;heading2,width2,space2 {;...}

;heading7,width7,space7

Example

```
.columns set heading1,10,5;heading2,10,1
.col1  one  .col2 two
```

crt

directs the output from OP to the terminal, overriding the "p" option specified at TCL.

Syntax

.crt

Example

```
.crt This message prints on the terminal.

Even if the text was output using the "p" option, the text following the
".crt" command is output on the screen.
```

cs

see "cap sentences".

cursor

outputs the text on the column and row specified.

If "column" or "row" is not specified with subsequent ".cursor" commands, the last specified value is used.

OP does not take into account any line or column position changes made by the ".cursor" command.

This may affect line wrapping and end of page processing.

Syntax
.cursor {column}{,row}

Example
```
.cursor 30,15

This line prints at column 30 and row 15.
```

date

outputs the current system date in the form "month day, year".

Example
```
.date

Outputs the current date, such as "June 8, 1992".
```

default

sends the device a control code to select the default font.

Most LASER printers have a "reset" or "default" code. This code is supplied to the Output processor by the CCB (Cursor Control Block) using the "assignfq" command.

If the printer does not support a default font, this command has no effect.

Syntax
.default

.df

df

see "default".

em

inserts spaces in output text and is dependent on the printer type.

"number.spaces" is a spatial insertion parameter and has set possible values (0,1,2...). This is used for exact spatial alignment on LASER printers.

".em number.spaces" inserts a "number.spaces-to-the-em" space. One "em" equals the width of the space character, i.e., "1-to-the-em" is one space character, "2-to-the-em" is one-half of a space character, and "0-to-the-em" is one-tenth of a space character.

The effect this command has on the output text is dependent upon the type of printer being used.

Syntax
.em number.spaces

f

see "font".

fig

see "figure".

figure

numbers the figures sequentially using the current chapter number followed by a dash and the sequence number within the chapter.

The descriptive text and the sequential number are recorded in the Table of Contents under the heading Figures.

If chapter numbers are suppressed, figures are numbered sequentially with a single number, i.e., Figure 1, Figure 2, etc.

Syntax

.figure text

.fig text

font

selects the font defined in the device item or specifies fonts that are not predefined in the printer item. The "number" selects the corresponding font number defined in the device item. (See the "devices" file).

"orientation" may be either "portrait" or "landscape".

"symbolset" may be either "ASCII", "legal", "roman8" or "linedraw".

"spacing" may be either "proportional" or "fixed".

"pointsize" specifies the point size.

"pitch" specifies the pitch.

"style" may be either "upright", "italic" or "reverse".

"strokeweight" may be "light", "lightmedium", "mediumlight", "medium", "mediumbold", "boldmedium" or "bold".

"typeface" may be any of the valid types defined in the corresponding item in the "devices" file for the printer being used. (For an example, see "hp-lzrii" in the "devices" file).

If the device does not support fonts, or the device item has not been configured, this command has no effect.

Syntax

.f{ont} number

.f{ont} orientation, symbolset, spacing, pointsize, pitch, style, strokeweight, typeface

Example

```
.font portrait,legal,fixed,10,12,upright,medium,courier
```

footing

designates a text string composed of literals and special options to output at the bottom of each page. If the line following the ".footing" command is null, the footing is suppressed.

Syntax

.footing {{text} {'options'}...}

.ft {{text} {'options'}...}

Options

* see "options: footing".

Example

```
.footing "'lc'page 'pn' printed at 'tlc'"
```

ft

see footing

ght

see gohanging tab

gohanging tab

indents the text to the column set by the previous ".ht" command.

Syntax

.ght

.gohanging tab

Example

```
Alpha  .ht  Designation for the first site to receive a new release of the
software.
Beta .ght Designation for the multiple test sites to receive a test copy of a
new release

Output from this example:

Alpha Designation for the first site to receive a new release of the software.
Beta Designation for the multiple test sites to receive a test copy of a new
release
```

hanging tab

sets a tab stop at the column in the text where it is entered.

If there are "number" spaces prior to the ".hanging tab", text is printed "number"+1 spaces from the left margin.

Any text that wraps is also printed "number"+1 spaces from the left margin.

To reset the margin, place a ".hanging tab" at the appropriate column position.

A number parameter may be entered with the ".hanging tab" command to set the hanging tab. When a number is entered, the hanging tab is set at that column number. The number may contain up to two decimal places.

The ".ht" command is equivalent to the ".hanging tab" command.

Syntax

.hanging tab {number}

.ht {number}

Example

```
.hanging tab 10
Now .ght is the time for all good men to come to the aid of their country.
So, come all 'ye faithful.

Output from this example:

Now       is the time for all good men to come to the aid of their country.
So, come all 'ye faithful.

.hanging tab 10
.ght Now is the time for all good men to come to the aid of their country.
So, come all 'ye faithful.
```

```
Output from this example:

          Now is the time for all good men to come to the aid of their
country. So, come all 'ye faithful.
```

heading

designates a text string composed of literals and special options to output at the top of each page.

If the line following the ".heading" command is null, the heading is suppressed.

Syntax
.heading {{text} {'options'}...}

hi

see hilite

hilite

prints a single character on the right margin and is in effect until an ".xhilite" command is issued.

Syntax
.hi{lite} character

Example

```
.hilite *

An asterisk appears on the right margin.
```

ht

see "hanging tab".

i

see "indent".

im

see "indent margin".

in

see "input".

indent

indents text "number.spaces" spaces from the left margin.

If "number.spaces" is negative, the line begins to the left of the previously set left margin.

If "number.spaces" is not specified, indent defaults to 1.

If the command is in the middle of an attribute, the preceding text is output and the next word begins a new paragraph that is indented "number.spaces" spaces.

All lines following the first line are output at the left margin. Only the first line is affected by the indent command.

Syntax
.i{ndent} {-}number.spaces

Example

```
 .indent 10
This line beings 10 characters from the left margin.
This line will print on the left margin.

Produces the following:

          This line begins 10 characters from the left margin.
This line will print on the left margin.
```

indent margin

adjusts the left margin a specified number of spaces.

If "number.spaces" is negative, the line begins to the left of the previously set left margin. If "number.spaces" is not specified, the indent defaults to 1 (one).

If the command is in the middle of an attribute, the preceding text is output and the next word begins a new paragraph that is indented "number.spaces" spaces.

The command stays in effect until the margin is changed.

Syntax

.indent margin {-}number.spaces

.im {-}number.spaces

Example

```
 .nf
 .indent 10
test
 .indent -5
line five
```

indent rmargin

adjusts the right margin a specified number of spaces.

If "number.spaces" is a positive number, the right margin is increased by "number.spaces" spaces. If "number.spaces" is a negative number, the right margin is decreased by "number.spaces" spaces.

If the command is in the middle of an attribute, the preceding text is output and the next word begins a new paragraph with a right margin of "number.spaces" spaces.

This command has no affect on the left margin.

See the ".right margin" command to set or reset the right margin.

Synonym for ".irm".

Syntax

.indent rmargin {-}number.spaces

.irm {-}number.spaces

index

inserts a "phrase" and the current page number into the index.

"Subheadings" are specified by appending a ":" (colon) to the ".index" command. The phrase following the colon is the 'main heading' and all subsequent phrases are subheadings.

If the main heading or phrase consists of multiple words, the words must be enclosed in quotation marks.

The ".index" command may be embedded in text, in which case, the end of the index text must be indicated by the ".xindex" command.

The index can be saved in an item with the ".save index" command. The index is printed with the ".print index" command.

Syntax

.index 'phrase'

.ix 'phrase'

.index: 'main heading' 'phrase' {'phrase'...}

.ix: 'main heading' 'phrase' {'phrase'...}

Example

```
.index 'Schedule'
.xindex

Produces an index entry such as:

Schedule   . . . 13

.index: 'Schedule' 'production'
.xindex

Produces an index entry such as:

Schedule
production    . . . 13
```

index heading

substitutes the heading text as the header for index pages. The heading text is output on the top of each index page.

Syntax

.index heading text

.ixh text

Example

```
.index heading Topics covered
.index 'Topic 1' 'Topic 2'
.index 'Topic 3'
.index 'Topic 4'
.xindex
.print index

Produces the following:

Topics covered
Topic 1,1
Topic 2,1
Topic 3,1
Topic 4,1
```

input

accepts and processes text input from the terminal.

Syntax
.in{put}

Example
```
Dear .input,
Please find attached ...
OP pauses, prompts the user, and resumes processing inserting the data in lieu
of the .input statement.
```

irm

see "indent rmargin".

it

See "italics".

italics

causes the text following the command to be output in italics.

If the printer does not support italics, this command has no effect.

Italics mode is terminated with a ".x" or a ".xitalics" command.

Syntax
.it{alics}

Example
```
In this example, the word .it example .x is italicized.

"example" prints in italics.
```

ix

see "index".

ixh

see "index heading".

j

see "justify".

justify

right-justifies each line of text with the right margin by adding spaces in each line to pad to the end of the line.

This command stays in effect until an ".xjustify" command is executed.

Syntax
.j{ustify}

left margin

sets the left margin

The left margin is set to the specified number of spaces from the left edge of the paper. The left margin remains in effect until an ".im", ".left margin" or ".standard" command is issued.

If no left margin is specified, it will default to 5.

Syntax
.left margin margin.setting

.lm margin.setting

Example
```
.left margin 10
```

line length

sets the line length a specified number of characters from the left margin.

This command remains in effect until changed by another ".line length" command.

Syntax
.line length number.characters

.ll number.characters

line printer

sends all subsequent output to the currently assigned printer.

Syntax
.line printer

.lp

ll

see "line length".

lm

see "left margin".

lp

see "line printer".

lpi

sets the number of lines-per-inch for output.

The default is six lines-per-inch.

If the printer does not support "number" lines per inch, this command has no effect. (It may be necessary to change the term setting to get the required number of lines to print on a page.)

Syntax
.lpi number

macro file

defines the file name that contains OP macro items.

Macros may contain multiple OP commands and text.

A macro is an item in the given file.reference with an item-id of the format ".macroid". The item-id of the macro must begin with a period (".").

The "macro file" name cannot consist of any periods (".").

Items from the macro file are included in OP documents by entering the item-id of the macro in the document.

Any string preceded by a period (".") that is not a standard OP command is treated as a macro.

A macro can reside in the file being edited, or in another file.

If a macro file name has not been declared, OP looks for the macro in the current file.

After the macro has been processed, control returns to the original document item.

Syntax

.macro file file.reference

.mf file.reference

Example

```
The item to process resides in the "templates" file, with an item-id of
".fax".

u document letter

01 This letter is to inform you of the following information.
02 .macro file templates
03 .fax
04
05 Sincerely yours,
06
07 Your Name Here

The item to process resides in the "dm" account in the "templates" file, with
an item-id of ".fax".

u document letter

01 This letter is to inform you of the following information.
02 .macro file dm,templates,
03 .fax
04
05 Sincerely yours,
06
07 Your Name Here

The item to process resides on the current account and same file ("document")
with an item-id of ".fax".

u document letter

01 This letter is to inform you of the following information.
02
03 .fax
04
05 Sincerely yours,
06
07 Your Name Here
```

mf

see "macro file".

nf

see "nofill".

nofill

resets both the ."justify" and ".fill" modes.

Removal of extra spaces does not take place, and end-of-lines are not stripped from the input.

Output lines are not "filled" nor are right margins justified.

This command causes a ".break" (new paragraph).

Syntax

.nofill

.nf

oc

see "over char".

over char

any character that follows the ".over char" (or ".oc") command and is treated as text.

An "override character" is used to allow OP commands to be printed as text. The override character can also be used with blanks or hyphens to prevent word-wrap.

The override character is not displayed.

The default override character is an underscore ("_").

Syntax

.over char character

.oc character

p

see "paragraph".

page length

sets the page length to the number of lines specified.

The physical page length, in terms of inches, can be calculated by the ".vmi" height of each of the printed lines multiplied by 1/48's of an inch. The length (in inches) of each page can vary as the vertical motion index is changed.

Syntax

.page length number.lines

.pl number.lines

page number

sets the default page number to a specified integer number.

Syntax

.page number page.number

.pn page.number

Example

```
.ft "Page 'p5'"
{text ...
.page number 10
{more text ...
After printing 'text' on page 5, the page number changes to 10 then prints
"more text".
```

paging

pauses at the end of each page when the output is sent to the terminal.

At the end of a page of terminal output, any key may be pressed to advance to the next page.

The ".xpaging" command turns off the paging option, causing the output to scroll.

Syntax

.paging

.pg

paragraph

specifies the number of spaces to indent the first line of a paragraph.

To utilize this command, begin each paragraph with exactly one space. This affects an output with an indent of "number.spaces", and a blank line between each paragraph.

Only the first line of the paragraph is indented. Subsequent lines are output at the left margin. When "number.spaces" is negative, the first line of the paragraph is output that many spaces relative to the left of the left margin.

The ".xparagraph" command terminates paragraph mode.

Syntax

.p{aragraph} {-}number.spaces

pf

see "preface".

pg

see "paging".

pi

see "print index".

pl

see "page length".

pn

see "page number".

pp

see "prefix page".

pptoc

see "print ptoc".

preface

sets automatic preface formatting.

Heading text is centered at the top of the next page and begins a separate page numbering scheme.

If a ".preface" command has already been issued, the page numbers resume where the last ".preface" command ended.

Preface sections appear in the Table of Contents with Roman numeral page numbers. To set the page numbers in the Preface header/footer to Roman numerals, specify "r" in the ".footing" command.

Preface mode continues until an ".xpreface" command is encountered.

The ".pf" command is equivalent to the ".preface" command.

Syntax

.preface text

.pf text

prefix page

turns on the prefix page numbering mode, which starts numbering each chapter at page 1. The current chapter number is used as a prefix for the page number.

For example, the fourth page of the third chapter would print as "3-4". The command ".xprefix page" turns off ".prefix page".

Syntax

.prefix page

.pp

Example

```
.pp
.chapter 1
.footing "page 'p'"
This is page one of chapter 1.
.bp
2nd page of chapter 1.
.chapter 2
1st page of chapter 2.
.bp
2nd page chapter 2.
```

print index

prints the Index.

This command changes the tab settings and begins a new page.

The index is sorted in alphabetical order and printed in two columns per page.

After the index is printed, it is cleared.

To save the index in an item for editing, use the ".save index" command.

Syntax

.print index

.pi

print ptoc

prints a partial Table of Contents.

This includes entries made by the following commands encountered since the last ".print ptoc" command:

".chapter", ".section", ".figure", ".table", and "appendix".

To print the complete Table of Contents, including preface entries, use the ".ptoc" command.

Syntax

.print ptoc

.pptoc

print toc

prints the complete Table of Contents.

This includes entries made with the ".prefix", ".chapter", ".section", ".figure", ".table", and ".appendix" commands.

Syntax

.print toc

.ptoc

prm

see "prompt".

prompt

outputs the text on the following line to the terminal.

This command is often used with the ".input" command to prompt for user input.

Syntax

.prompt

.prm

Example

```
.prompt
Enter the name:
Dear  .input,

Enclosed is the additional product information that you requested.

Sincerely,

Brian
```

ptoc

Print table of contents, see "print toc".

r

see "read".

read

reads an (OP) item and outputs the item as part of the current text.

At the end of the item, OP continues with the next line following the ".read" command.

Syntax

.read file.reference item-id

.r file.reference item-id

Example

```
.read chapter1
.read chapter2
```

readnext

reads data, one value at a time, from an "active list".

It has an effect only if, prior to entering OP, a "select", "sselect", "qselect" or "get-list" statement has been executed.

Each ".readnext" command in OP extracts the next value from the active list and places it in the text stream.

".readnext" does not cause a "break".

If there is no pre-selected list, or when the list is exhausted, the ".readnext" command terminates OP, and returns to the calling process.

Syntax

.readnext

.rn

Example

```
Dear .readnext

  You are the only one for me.

Love,

Pat
.chain love.letter
```

readnext null

reads data from an active list.

"readnext" does not terminate if a list value does not exist.

reset

resets the printer to its initial state.

If the printer does not support reset, this command has no effect.

Syntax

.reset

.rst

right margin

defines the right margin as a specified number of spaces from the edge of the printable page.

The right margin remains in effect until an ".indent rmargin", ".right margin" or ".standard" command is issued.

If no right margin is specified, it will default to 0.

Syntax

.right margin number.spaces

.rm number.spaces

rm

OP "right margin"

rn

see "readnext".

rst

see reset

save contents

saves the Table of Contents to a defined item-id.

The item is saved in the same file as the current document.

After saving, the item can be formatted and edited to create a customized Table of Contents.

Syntax

.save contents item-id

.sc item-id

Example

```
.chapter
.save contents
```

save index

saves the index to a defined item-id in the same file where the OP document resides.

The ".save index" command is placed in the text item itself and must precede the ".index" commands.

Only the indexed data which has been preceded by the ".save index" command is saved in the specified file.

The saved index is formatted on one column. The item can then be formatted and edited to create a custom index format.

Syntax

.save index item-id

.si item-id

Example

```
.si indexbody
.index: 'topic one' 'subtopic one' 'subtopic two' .xix
.bp
.index 'topic two' .xix
.pi

Results in index body:

topic one
subtopic one, 1
subtopic two, 1
topic two, 1
```

sc

OP "save contents".

sch

see "set chapter".

sec

Define a section

section

produces automatic section numbering and formatting.

This command begins the section at level "number". If "number" is not specified, it defaults to "1".

Section numbers are sequentially incremented within levels. When a section number is encountered, the section number and title text are output.

The section number and title text are recorded in the Table of Contents. Section numbering may be suppressed using the ".set section" command.

Syntax

.sec{tion} {number} title text

set chapter

sets the options for chapter headings.

This command must precede any ".chapter commands".

Syntax

.set chapter {options}

.sch {options}

Options

b Boldface.

i Italics.

u Underline.

s Suppresses section numbers. The "s" option also suppresses the word "Chapter", followed by the chapter number, from being output above the chapter heading. The chapters are still numbered internally, so the ".prefix page" command can still be used.

r Roman numeral chapter numbers.

set section

defines the options for the section headings.

This command must precede any ".section" commands.

The options are entered one after the other, and are not separated by commas or spaces. Any combination of options may be used. However, some printers do not accept a combination of boldface, italics and underline.

Syntax

.set section {options}

.ss {options}

Options

n Integer number of lines to skip.

b Boldface.

i Italics.

u Underline.

s Suppresses section numbers.

si

See "save index".

sk

See "skip".

skip

outputs a specific number of blank lines using the current line spacing.

The line spacing can be set either by the ".spacing" command or as part of the statement that invokes OP.

If the current line spacing is 2, a ".skip" command skips 4 lines.

If "number.lines" is not specified, "number.lines" defaults to one.

To skip lines based on a single-line spacing, instead of the current line spacing, use the ".space" command.

Syntax

.sk{ip} {number.lines}

Example

```
.spacing 4
{text
.skip 2
{more text
```

sp

see "space".

space

or ".sp", command outputs a specific number of blank lines using single-line spacing.

This command is independent of the line spacing.

To skip lines based on line spacing, use the ".skip" command.

Syntax

.sp{ace} number.lines

Example

```
Sentence one on line one.
.space 3
Sentence two on line 5.
```

spacing

sets the line spacing to a specific number of lines.

This command causes blank lines to be printed in between each line of text.

Syntax

.spacing number.lines

.spg number.lines

Example

```
.spacing 3
```

spg

see "spacing".

ss

OP "set section".

standard

defines the default parameter and mode settings for OP text.

Resets OP to the default parameters and modes. The default settings are:

left margin 5

right margin 5

line length term width minus left & right margins

paragraph mode off

text single spaced

override character underline(_)

hilite character asterisk (*)

hyphenation on

crt output pause at the end of page

block center off

Syntax

.standard

.std

std

see "standard".

sub

see "subscript".

subscript

moves text down one-half line.

Text remains in this position until an ".xsubscript" command is issued.

If the printer cannot move in half lines, this command has no effect.

Syntax

.subscript

.sub

sup

see "superscript".

superscript

moves text up one-half line.

If the printer cannot move in half lines, this command has no effect.

To move the text back down one-half line, use the ".xsuperscript" command.

".sup" and ".xsup" are synonyms for ".superscript" and ".xsuperscript".

Syntax

.superscript

.sup

.xsuperscript

.xsup

tab fill

defines a pattern which is repeated to fill the line from the current position to the next tab stop.

Tab stops must be set (see ".tab set") prior to using this command.

Syntax

.tab fill {"}pattern{"}

.tf {"}pattern{"}

Example

```
.ts 50
.tf "*"
```

tab left

moves the word or specified phrase following the command to the next tab position and left-justifies the last character of the word or phrase at the tab position.

The ".tl" command is equivalent to the ".tab left" command.

Syntax

.tab left {"}phrase{"}

.tl {"}phrase{"}

Example

```
.ts 10,20
.tl text1 .tl text2
```

tab right

moves the word or delimited phrase following the command to the next tab position and right-aligns the last character of the word or phrase at the tab position.

To right tab a phrase, delimit the phrase with quotes (").

To use single quotes in the phrase, double them up. (") outputs a quote in a right-tabbed phrase.

Syntax

.tab right {"}phrase{"}

.tr {"}phrase{"}

tab rightm

right-aligns the optional text with the right margin.

Syntax

.tab rightm {"phrase"}

.trm {"phrase"}

tab set

defines the location of tab stops.

Tab stops must be greater than zero and in increasing order.

The position, "tabstop", is relative to the left margin, which includes any indent margin (".im") settings.

A tab must be less than the current line length.

There may be a maximum of 20 tab stops per command.

A tab position may contain up to two decimal places. This is useful when using proportional pitch fonts and printers with horizontal movement capabilities.

To move the text to these tabs, use the ".tl" and the ".tr" commands. (Do not use the <tab> keys.)

Syntax

.tab set tabstop,tabstop,tabstop,...

.ts tabstop,tabstop,tabstop,...

Example

```
.tab set 5,10,15
```

table

provides automatic table numbering with sequential numbers in the form of a chapter number followed by a dash and the sequence number within the chapter, i.e., Table 1-4, Table 3-2.

The descriptive text and the table number are recorded in the Table of Contents under the heading Tables.

If chapters are not used, tables are numbered sequentially, i.e., Table 1, Table 2, etc.

Syntax

.table text

.tbl text

tbl

see "table".

tc heading

substitutes the heading text as the header for the Table of Contents pages.

Syntax

.tc heading "{{text} {'options'}...}"

.tch "{{text} {'options'}...}"

Options

* see "options: heading".

Example

```
.tc heading "'lc'The Pick Systems Reference'l'"
```

tc

see "tc heading".

tcl

executes the specified "TCL.command" and insert the output in the OP document.

When the "TCL.command" completes, control returns to the document and continues processing on the following line.

Syntax

.tcl TCL.command

Example

```
The "listu" command is used to show the users currently logged on to
the system.
Example:
.tcl listu
```

tcl box

executes the specified TCL.command as part of the output and draws a box around it.

This command is similar to the ".box" command, except box parameters may not be specified.

The width of the box is determined by the left and right margins.

Syntax

.tcl box TCL.command

.tclbox TCL.command

Example

```
The "who" command is used to show the current user-id and account name.

.tcl box who
```

tclbox

see "tcl box".

test page

tests the number of lines left on the current page and determines whether text is output on the current or next page.

If the "number.lines" parameter is less than the count returned, the following text is output on the current page. Otherwise, a page is ejected and the text is output on the new page.

This prevents blocks of text from being split across a page boundary.

Syntax

.test page {number.lines}

.tp {number.lines}

Example

```
.test page 10

This tests to determine if there are at least 10 print lines
remaining on the current output page. If there are, the following text is
```

```
printed on the same page, otherwise, a form-feed is issued and the text prints
on the next page.
```

tf

OP "tab fill".

tl

see "tab left".

tp

see "test page".

tr

see "tab right".

trm

see "tab rightm".

ts

See "tab set".

ul

see "underline".

underline

underlines the following text until an ".xul" or ".x" command is encountered.

This command underlines spaces as well a words.

This command works on printers and terminals that support the underline function.

Syntax

.underline

.ul

Example

```
.ul D3 .xul
D3
```

underline words

underlines all words until an ".xuw" command is issued.

Spaces between words are not underlined.

Syntax

.underline words

.uw

uw

see "underline words".

variable columns

specifies parameters for formatting in variable-width multiple columns.

These parameters are used by subsequent ".columns set" commands.

The ".variable columns" command is designed to be used when the actual line length is unknown. For example, if the document is to be printed with a variety of different printers or fonts, each with a different number of characters that are printed on a line, ".variable columns" guarantees that columnar material fills in the entire page width.

The space parameter specifies the number of spaces between columns.

The divisions parameter specifies the number of sections into which the current line length is to be divided. This parameter should be equal to or greater than the sum of the width parameters or the command is ignored.

The heading parameters specify the headings for each column. The headings are output at the current location and at the top of each page.

The width parameters specify the number of divisions to be included in the width of each column.

Syntax

.variable columns space,divisions, heading1, width1 {;heading2,width2 {;...;heading7,width7}}

.vcol space,divisions, heading1, width1 {;heading2,width2 {;...;heading7,width7}}

vcol

see "variable columns".

vmi

specifies the amount of vertical space each line uses.

"vmi" spacing is allocated in 1/48th-inch increments, and is essentially the "flip side" of the "lpi" command. For example, if the "lpi" is set to 6, the equivalent "vmi" setting would be "8" (48/6).

Syntax

.vmi number.lines

Example

```
.vmi 12

This example sets "vmi" to approximately 4 lines per inch. This gives an
actual line spacing of 1/4 inch per line.
```

x

turns off boldface, underline word, underline and italic modes.

Syntax

.x

xbc

see "xblock center".

xbf

see "xboldface".

xblock center

turns block center mode off.

Syntax
.xblock center

.xbc

xboldface

The ".xboldface" (or ".xbf") command turns boldface mode off.

Syntax
.xboldface

.xbf

xbox

terminates a box command and draws the bottom line of the box.

Syntax
.xbox

xcap sentences

terminates the ".cap sentences" mode.

Syntax
.xcap sentences

.xcs

xcol

See "xcolumns"

xcolumns

turns off the column heading, width and spacing set by the ".columns" or ".columns set" commands.

Syntax
.xcolumns

.xcol

xcs

see "xcap sentences".

xhi

see "xhilite".

xhilite

terminates the ".hilite" or ".hi" commands.

Syntax
.xhilite

.xhi

xindex

designates the end of an index entry for an entry embedded in text.

Syntax
.xindex
.xix

xit

see "xitalics".

xitalics

turns off the ".italics" mode.

Syntax
.xitalics
.xit

xix

see "xindex".

xj

see "xjustify".

xjustify

turns off "justify" mode. The ".xj" command is equivalent to the ".xjuxtify" command.

Syntax
.xjustify
.xj

xp

See "xparagraph".

xpaging

turns off the paging option and causes the output to scroll down the terminal.

Syntax
.xpaging
.xpg

xparagraph

turns off paragraph mode.

Syntax
.xparagraph
.xp

xpf

see "xpreface".

xpg

see "xpaging".

xpp

see "xprefix page".

xpreface

turns off preface mode.

Syntax

.xpreface

.xpf

xprefix page

turns off the prefix page mode.

Syntax

.xprefix page

.xpp

xsub

see "xsubscript".

xsubscript

turns off subscript mode.

Syntax

.xsub

.xsubscript

xsup

see "xsuperscript".

xsuperscript

turns superscript mode off.

Syntax

.xsuperscript

.xsup

xul

turns off underline mode.

Syntax

.xul

xuw

terminates the underline words command.

Syntax

.xuw

Proc

linkages

the means by which PROCs either pass control one-way to another PROC, or call another PROC as an external subroutine.

The type of "linkage" is determined by the character used in the syntax.

"( )" (parentheses) Transfers control to the PROC in the designated filename as a "one-way" transfer (like the "chain" statement in FlashBASIC, meaning that it does not return control upon completion). If no item-id is specified, the default item-id is the value in the current position of the input buffer. Control is not returned to the calling PROC. The optional "label" parameter indicates the statement label number where execution should begin.

"[ ]" (brackets) Transfers control to the PROC in the designated filename as a "two-way" transfer (like the "call" statement in FlashBASIC, meaning that it will eventually return). Control returns to the next executable statement in the calling PROC upon completion. If no item-id is specified, the item-id defaults to the value in the current position of the input buffer.

The "[ ]" form may be used as an "internal" subroutine call, when both the filename and item-id are omitted, and a statement label is included. The "internal" subroutine "returns" to the next line after the "call" when it encounters an "x" command.

Syntax

({dict} filename {item-id})

({dict} filename {item-id}) {label}

[{dict} {filename} {item-id}]

[{dict} filename {item-id}] {label}

Example

```
[procs example1]
Transfers control to 'example1' in the 'procs' file. Control returns to the
next line in the current PROC upon completion.

[procs example2] 25
Transfers control to 'example2' in the 'procs' file, and begins execution at
statement label 25. Control returns to the next line in the current PROC upon
completion.

[] 150
Local subroutine call. Transfers control to statement label 150 in the current
PROC and begins execution.

(procs)
Transfers control to the 'procs' file, using the item-id indicated in the
input buffer.
```

logon PROCs and Macros

a PROC or macro which automatically executes at log on.

After the user has provided a valid account name and optionally the password, the md of the designated account is checked for a PROC or macro with the same name as the account. If it exists, it starts up automatically. If no such PROC or macro exists, control is transferred to TCL.

In D3, the "user-macro" may also be used for personalized logon sequences.

pq

dictionary code in a master dictionary item defining itself as a PROC. It is required in the first line of every PROC.

Example

```
001 pq
002 c This PROC runs the month-end processing
```

primary input buffer

the point through which all keyboard input passes between the operator and the PROC being executed. The PROC "sp" and "s" commands activate the primary input buffer.

primary output buffer

place where commands are constructed which will subsequently be passed to TCL for processing. A number of commands are used on the primary output buffer and are linked to this item.

PROC

a processing language which drives commands through TCL.

secondary input buffer

used exclusively for holding the item-id's of error messages returned by processes executed by PROC. The PROC "ss" command activates the secondary input buffer.

secondary output buffer

used to store strings which will subsequently be passed to TCL for processing, after the contents of the primary output buffer have been processed. A number of commands are used on the secondary output buffer and are linked to this item. The "ston" (or "st on") command invokes the secondary output buffer.

See the (PROC) "ston" command for an example.

+

adds the specified integer number to the current parameter of the presently active input buffer.

Syntax

+number

-

subtracts the specified integer number from the current parameter of the presently active input buffer.

Syntax

-number

a

moves data from the specified parameter number in the currently active input buffer to the currently active output buffer.

Optionally the "a" command surrounds the data by a specific character, such as a quotation mark.

The "surround.char" parameter specifies the character to place before and after the string, after it is moved.

The "number.characters" parameter specifies the number of characters to move.

Syntax

a{surround.character}{{(}param.number}{, number.characters{)}}

Example

```
a"1

This moves the contents of input buffer one to the currently active output
buffer, surrounded by double quotes.

a(2,999)

This moves the second through 999th (or whenever it runs out of characters) to
the currently active output buffer.

s2
a

This moves the contents of parameter 2 to the currently active output buffer.
```

b

decrements the currently active input buffer pointer by "1"

bo

decrements the currently active output buffer pointer by "1" (one).

c

indicates a "remark" statement.

All text which follows on the same line is ignored by the PROC interpreter.

Syntax

c {text}

Example

```
pq
c  This PROC runs the month-end processing
```

d

displays either a specific location from the currently active input buffer, or all locations.

When used in the form: "d0", all locations in the input buffer are displayed.

If the optional parameter.number is specified, only the contents of the specified input buffer are displayed.

The "d0" command displays the entire current input buffer.

The "+" character suppresses the automatic linefeed at the end of the display.

Syntax

d{{(}parameter.number}{,number.characters{)}}{+}

d0

f

increments the currently active input buffer pointer by "1"

g

Goto statement label.

go

transfers program control to another PROC statement.

The first form transfers control to the specified statement label. The second form is an indirect transfer, according to the value of the label, as extracted from a parameter in the input buffer

Syntax

g{o} statement.label

g{o} aparameter.number

Example

```
go 99
Transfer control to PROC statement with label 99.

go a4
Transfer control to the PROC label that matches the fourth parameter in the
active PROC input buffer.
```

h

moves a literal string of characters to the currently active output buffer.

The "<" character is the equivalent of a <return>, and must follow each discrete parameter when moving multiple parameters to the secondary output buffer (following the "ston" command).

Syntax

hstring{<}

Example

```
hlist entity
h name address phone
p

The "h" commands move the strings following them to the end of the currently
active output buffer.

hget-list invoices
ston
hlist-label invoices cust.name cust.addr custcsz (cip<
h1,3,1,0,30,0,c<
p

The "<" characters in the example are required to separate, or "stack", the
strings to execute after the list is retrieved.
```

if

performs a conditional expression.

Fist form: The "e" operand tests for a specified message number from the "dm,messages," file. If true, the specified PROC command is performed.

Second form: The "s" operand tests for an active list. May be used after a "select", "sselect", "qselect" or "get-list" command.

Third form: This tests the contents of the buffer location specified in the "a{param.number}" expression with any logical or pattern matching operator, and performs the specified PROC command if successful. The optional ",n" argument allows only the specified number of characters to be compared.

The logical operators allowed in constructing PROC conditionals are: "=" equal to; "#" not equal to; "<" less than; ">" greater than; "[" less than or equal to; "]" greater than or equal to. The "#" character tests for the absence of the expression, when used on the left of the a-command.

Pattern matching operators for composing "match.strings":

(na) accepts "n" alphabetic characters only.

(nn) accepts "n" numeric characters only.

(nx) accepts "n" of any characters.

('string') accepts literal string.

The "n" parameter in the pattern match string specifies the length of the match operator field. A length specification of zero (0) allows variable length input. The parentheses are required around the match.strings.

Syntax

if {#} e {operator message.number} command

if {#} s command

if {#} a{param.number}{,n} {operator {match.string}} command

Example

```
if e = 401 xno items were selected...
Checks for error message 401 and stops/returns if present.

if a # (3n'-'2n'-'4n) oinvalid ss#.
Checks for 3 numbers, a dash, 2 numbers, a dash, and 4 more numbers.

if a = x xdone.
Checks the currently active input buffer location for the presence of the
letter "x".

if a = (0n) go 10
Checks the currently active input buffer location for the presence of zero or
more numbers.

if a1,1 = q x proc terminated voluntarily
Checks the first character of the currently active input buffer location for
the presence of the letter "q".

if # a1 go 10
Checks for the absence of a value in the first location of the currently
active input buffer.
```

ih

removes the current parameter from the currently active input buffer and replaces it with a null value.

If the pointer is in the middle of a parameter, the parameter is truncated, starting at the current position.

"ih \" inserts a new null parameter in front of the current parameter in the currently active input buffer. If the pointer is in the middle of a parameter, the parameter is truncated starting at the current position and a new null parameter is added after the truncation.

"ih string" inputs a data string into the active input buffer.

Syntax

ih\

ih \

ih string

Example

```
s1
o print (y/n=<return>) +
ihn
ip:

This stores "n" into the input buffer location, giving the operator two ways
to just say "no".
```

ip

stops execution of the PROC and awaits a response from the keyboard.

The input data is placed into the currently active location of the active input buffer, which is determined by previous "s", "b" or "f" commands. The default prompt character is a colon (:).

Syntax

ip{prompt.character}

Example

```
o enter response +
s1
ip:

The input received from the keyboard is placed into the first location of the
active input buffer.
```

is

temporarily stops execution of the PROC until a response is provided from the keyboard. Input is placed into the secondary input buffer.

The default prompt character is a colon (:).

Syntax

is{prompt.character}

it

inputs data from the tape label of the currently attached magnetic media directly into the primary input buffer.

o

outputs literal text to the terminal at the next available line.

The "+" character suppresses the automatic linefeed/carriage return after output.

Syntax

o {text}{+}

Example

```
o  This reports takes a while to run...
o  Please sit by...
o
```

p

processes the command(s) in the currently active output buffer(s). The primary output buffer is processed first, followed by the secondary output buffer.

Both the primary and secondary output buffers are automatically reset after the "p" command, and the active buffer pointer is reset to point to the primary output buffer.

Example

```
hlist entity
h name address
p
o report completed at +
htime
p

The first "p" in this example processes the Access sentence, "list entity name
address". The second "p" processes the TCL "time" command.

hselect entity by name
ston
hsave-list entity.by.name
p

The "p" command in this example processes the commands in both output buffers.
```

ph

processes the command(s) in the currently active output buffer(s) and "hushes" terminal output. Output which would normally go to the screen is turned off.

Example

```
hsselect entity by name
ston
hsave-list entity.list
ph
```

pp

displays and processes the command(s) in the currently active output buffer(s).

Example

```
hsselect entity by name
ston
hsave-list entity.list
pp
```

pw

displays the command(s) in the current output buffer(s) and "waits" for a response prior to processing.

Valid responses:

g (or <return>) "go". Proceeds with process.

s "skip". Proceeds with next command in PROC.

x (or n) "cancel". Returns control to TCL.

Example

```
hsselect entity by name
ston
hsave-list entity.list
pw
```

px

processes the command(s) in the currently active output buffer(s) and unconditionally stops the PROC upon completion of the command(s) returning control directly to TCL.

Example

```
hsselect entity by name
ston
hsave-list entity.list
px
oEND OF PROC
```

ri

"resets" either the entire PROC input buffer, or all of the locations following a specified location.

This command resets both input buffers to null. The "start.parameter" indicates the "starting" position to reset. All parameters following the specified parameter and the entire secondary input buffer are cleared.

Syntax

ri{start.parameter}

Example

```
ri
Clears both input buffers.

ss
ri
sp
Clears only the secondary input buffer.
```

ro

resets both of the output buffers.

There is hardly a case where this is ever required.

s

"sets" the input pointer to a specified parameter number, activating the specified parameter number as the currently active input buffer location.

Syntax

s{parameter.number}

Example

```
s1
ip:
```

sp

selects the primary input buffer, and positions the pointer to the beginning of the input buffer.

ss

selects the secondary input buffer and positions the pointer to the beginning of the secondary input buffer.

stoff

selects and directs output to the primary output buffer.

Syntax

stoff

st off

ston

selects, and directs output to, the secondary output buffer.

Syntax

ston

st on

Example

```
hsselect entity by name
ston
hsave-list entity.list
p
```

t

directs output to the terminal screen and controls special terminal display functions.

Functions:

+ Suppresses carriage return after output.

b Rings terminal "bell".

c Clears terminal screen.

(column,row)

Outputs the value at the column and row coordinates.

in Outputs ASCII character with an integer value of "n".

xn Outputs ASCII character with integer hexadecimal value of "n".

"text" Outputs literal text. Double quotes are required around literal text.

(@function)

The entire complement of "@" functions are available. See "@ function (FlashBASIC)" for a complete listing.

Syntax

t {function,function,...}{+}

Example

```
t (-1),(20,0),"Main Menu"

This command clears the screen, (-1), positions the cursor at column 20, row
0, and displays the string "Main Menu".
```

```
t (10,10),(-13),"Chosen",(-14),(-4)

This command positions the cursor to column 10, row 10, displays the string
"Chosen" in reverse video, (-13) and (-14), and clears to the end of the line,
(-4).

s1
ihy
t (0,5),"Is this OK ?"
d1+
t i8,i8
ip?

This set of instructions sets the input buffer pointer to the first position
(s1), places a "y" in the first input buffer position (ihy), outputs the
string "Is this OK ?" at column 0, row 5, displays the contents of the first
buffer position and holds the cursor (d1+), outputs two backspaces (t i8,i8)
and prompts for input (ip?). This displays the default value "y", backs up the
cursor two positions so that it is one character in front of the "y" and then
issues a prompt character, "?", which places the cursor directly over the "y".

t c,i7,(10,10),b

This statements clears the screen (c), rings the bell by issuing a character 7
(i7), positions the cursor at column 10, row 10, and rings the bell again (b).

t x1b,x8,"exec dir",x0

This statement is typically used to control an intelligent DOS terminal (such
as ViaDuct). It sends a "preamble", consisting of an escape, backspace (x'1b'
and x'8', specified in hex), then it sends the command string, "exec dir",
followed by the "end" command character (x'0').  This statement can be written
using integer references:

t i27,i8,"exec dir",i0
```

u0032

logs off the current process.

u0070

performs a correlated match on a multi-valued set.

"d1" is the value to match, and "d2" is the data to display.

Syntax

u0070,d1,d2

u0190

performs arithmetic on a reverse-Polish notation string which has been built in the current output buffer.

"string.delimiter" is a single-character delimiter which may be any character of the user's choosing, as long as it does not appear in the calculation string.

"calculation.string" is the arithmetic function to perform, in reverse-Polish notation. It is built in the primary input buffer prior to the user exit call. The final argument in the calculation string is always a "?" to indicate that the calculation is to be performed.

The line following the user exit contains the closing string.delimiter.

"output.target" specifies where the output is to be placed. "Output.targets" are : "S" for current output buffer, "P" for primary input buffer, "A" for secondary output buffer. Any other entry is taken to mean output to terminal, however "T" is usually used.

The calculation string is evaluated, the input buffer is cleared, and the result is output to the specified "output.target".

The following arithmetic operators are available for use within the calculation string:

```
+    Addition.
-    Subtraction.
*    Multiplication.
/    Division.
&    Logical AND.
!    Logical OR.
\    Logical NOT.
<    Less than.
<=   Less than or equal to.
=    Equal to.
\=   Not equal.
>=   Greater than or equal to.
>    Greater than.
\<   Not less than.
\>   Not greater than.
```

Syntax

string.delimiter

calculation.string.data

?

u0190

string.delimiter output.target

Example

```
001 pq
002 OEnter A and B
003 ip
004 st on
005 h:
006 a1
007 h
008 a2
009 h +
010 a1
011 h
012 a2
013 h - * ?
014 u0190
015 : t
016 x

Builds the calculation string ":<a><b>+<a><b>- * ?:", which evaluates to
"(<a>+<b>) * (<a>-<b>)"

002 s2
003 h: 0 ?
004 u0190
005 : P

Places a zero in the second position of the primary input buffer.
```

u0191

sets system execution lock number 9. If the lock takes place succesfully, the next physical line of the PROC is skipped. When it does NOT lock succesfully, the line immediately following the user exit is performed.

Example

```
PQ
U0191
XUnable to lock at present..
OHere we go...
```

u0192

used for general output formatting.

Output values can be literals, attribute values, or converted attribute values.

"Format.spec" may be one or more of the following values:

h{nn} string

Places "string" in the output buffer. If "nn" is specified, the value is output starting at output column "nn". If "nn" is omitted, the value is output at the current column position in the output buffer.

h{nn} %mm

Places the value located at the "mmth" position of the primary input buffer into the current output buffer. If "nn" is specified, the value is output starting at column "nn". If "nn" is omitted, then the value is output at the current column position in the output buffer.

h{nn} #mm

Places the value located at the "mmth" position of the primary output buffer into the current output buffer. If "nn" is specified, the value is output starting at output column "nn". If "nn" is omitted, the value is output at the current column position in the output buffer.

l

Re-routes output to the printer instead of the terminal. The terminal is re-selected for output at the end of the user exit.

p{nn}

Page-ejects. The value, "nn", places "nn" formfeeds in the current output buffer. If "nn" is omitted, it is assumed to be one.

s{nn}

Vertical spacing. The value, "nn", places "nn" carriage return/line feeds in the current output buffer. If "nn" is omitted, it is assumed to be one.

v{nn} file.reference item.reference attribute.mark.count

Retrieves a specific value from an attribute in an item. If "nn" is specified, the value is output starting at output column "nn". If "nn" is omitted, the value is output at the current column position in the output buffer.

x{nn}

Horizontal spacing. The value, "nn", places "nn" spaces in the current output line. If "nn" is omitted, it is assumed to be zero.

*{nn} file.reference item.reference attribute.reference

Retrieves the specified value of an attribute in an item. If "nn" is specified, the value is output starting at output column "nn". If "nn" is omitted, the value is output at the current column position in the output buffer. This is similar to the "v" format.spec, with the following important exceptions:

a) the value is referenced by attribute name instead of the attribute number, and

b) conversion is performed as specified by the dictionary v/conv entry, before output.

-> t{+}

Format specification terminator. A space is required between the "->" and the "t". The "t" may optionally be followed by a "+" to suppress the final carriage return/line feed.

Syntax

u0192

format.spec

:

-> t

u0193

used to move parameter from "source" buffer to "destination" buffer. The "source" and "destination" parameters can be %n or #n where % indicates the primary input buffer and # indicates the active output buffer. "n" indicates the parameter number.

Syntax

u0193

source destination

u0195

saves an active list. The address of the currently active lists, the pointer to the next item to be accessed from the list, and the base/modulo of the file are saved in a special workspace.

Once the list has been saved, it may be accessed via the user exit "u1195"

u01a2

executes a computed goto branch based on the value in the current output buffer.

"Index" is the value to compare against one or more arguments.

"Arg1" through "argn" are values to compare against the index. If a match is made, control passes to the n+1th line below the user exit call.

Syntax

u01a2

index arg1 ... argn

(jump here if index = arg1)

:

(jump here if index = argn)

(continue here if no args match)

Example

```
001 pq
002 o  Demonstration of the computed-goto
003 o  Enter: YES, NO, or MAYBE
004 st on
005 h1
006 in
007 a
008 h3
009 u01a2
010 2  YES  NO  MAYBE
011 g 100
012 g 200
013 g 300
014 x Sorry, only a YES, NO, or MAYBE may be entered.
015 100 x You entered a YES
016 200 x You entered a NO
017 300 x You entered a MAYBE
```

u01a6

performs cursor control functions. Nearly all the functionality of this user exit is available through the PROC "T" command, which should be preferred to this user exit.

control codes:

(col,row) positions cursor at specified column and row.

b Rings bell.

c clears screen.

in Outputs ASCII character with an integer value of "n".

xn Outputs ASCII character with integer hexadecimal value of "n".

"text" outputs literal text.

Syntax

u01a6

control.code{,control.code,}...

Example

```
U01A6
c
Clears the screen, and is the equivalent to:

T C
```

u01ad

retrieves a value from an attribute of an item in a file.

The next line of the PROC specifies the file information.

"file.name" is the name of the file. If file.name is prefixed with an asterisk (*), the DICT portion of the file will be accessed.

"item.name" is the item name.

"attribute.ref" is the attribute name (or number)

"code.number" is:

a = Outputs to alternative output buffer.

p = Outputs to Primary input buffer.

s = Outputs to current output buffer.

t = Outputs to terminal.

v = Verifies that the file exists

va= Verifies that the attribute exists.

This function allows "indirect" references to values from the input and output buffers by using the following special symbols:

"%" indicates the current input buffer value.

"#" indicates the current output buffer value.

Syntax

u01ad

file.name item-id attribute.ref code.number

Example

```
Example of direct reference:
  pq
  u01ad control-file co.name 1 s
  xcan't find co.name or control-file

Reads attribute 1 of the item called "co.name" in the file called "control-
file" and places it in the current output buffer.

Example of indirect reference:

  pq
  u01ad %2 %3 %4 p
  xsomething went wrong...
  d0
Reads the attribute number as defined in the fourth location of the input
buffer, using the item-id from the third location and the filename from the
second location.
```

u01b8

returns and (optionally) formats a value from a file or a text string.

u01bc

performs an "n"-way branch.

u1191

releases previously-locked execution lock number 9. This lock typically would have been set by a call to user exit "u0191".

If the execution lock was locked by some other port.number, then it will not be unlocked.

u1193

swaps the first two values of the date.

This allows for conversions of "European" date displays to "American" date displays, or vice-versa.

u1195

retrieves the next item in an active list.

u11a2

pads the current input buffer with enough leading zeros to make the value "number.of.digits" digits wide, and returns the resulting string to the secondary input buffer.

Syntax

u11a2

number.of.digits

Example

```
009 in
010 u11a2
011 4

Input values of "1", "01", "001", or "0001" will all be replaced by "0001".
```

u11aa

Converts the current value in the pib into the PROC secondary input buffer.

Example

```
U11AA
O MD2,$
```

u11ad

retrieves all multi-values from an attribute.

The following line of the PROC specifies the file information, and a code.number.

"file.name" is the name of the file. If file.name is prefixed with an asterisk (*), the DICT portion of the file will be accessed.

"item.name" is the item name.

"attribute.ref" is the attribute name (or number)

The "code.number" specifies where the data is to be placed.

a = Outputs to alternative output buffer.

p = Outputs to Primary input buffer.

s = Outputs to current output buffer.

t = Outputs to terminal.

v = Verifies that the file exists.

va= Verifies that the attribute exists.

Syntax

u11ad

file.name item.name attribute.ref code.number

u11bc

pads the value in the secondary input buffer with zeros.

The next line in the PROC specifies the number of zeros.

Syntax

u11bc

number.of.zeros

u20d7

initiates a restore of the entire file system from "file-save" media.

u218d

disables the break key.

u2191

unconditionally releases execution lock number 9.

This execution lock is typically set with user exit "u0191".

u2193

sets the current position of the assigned print queue as "top-of-form" for subsequent printer output.

u219b

kills the PROC stack flag

Syntax

u219b

u21a2

resets both output buffers, just like the "ro" command.

u21ad

retrieves all values of an attribute from an item in a file, but does not advance the multi-value counter.

The following line of the PROC specifies the file information, and a code.number.

"file.name"

is the name of the file. If file.name is prefixed with an asterisk (*), the DICT portion of the file will be accessed.

"item.name"

is the item name.

"attribute.ref"

is the attribute name (or number).

"code.number"

specifies where the data is to be placed.

a = Outputs to alternative output buffer.

p = Outputs to Primary input buffer.

s = Outputs to current output buffer.

t = Outputs to terminal.

v = Verifies that the file exists.

va= Verifies that the attribute exists.

Syntax

u21ad

file.name item.name attribute.ref code.number

Example

```
See "u01ad".
```

u21bc

deletes entries from the output buffer.

The next line in the PROC specifies the "delete.code":

0 = Deletes all entries.

1 = Deletes the last entry only.

Syntax

u21bc

delete.code

u318d

enables the break key.

u3191

releases an execution lock if it was locked by this port.number.

This user exit performs the same function as "u1191" with this exception: no error path is take if execute lock is locked by another process.

u3193

directs all subsequent output to the terminal.

Example

```
001 pq
002 u0193
003 hlist md
004 p
(all output goes to the terminal)
005 u3193
006 hlist md
007 p
(all output goes to the printer)
```

u31a2

flags a statement for later transfer of control.

u31ad

returns the current port number.

The "code.number" following the user exit specifies where the value is to be placed:

a = Outputs to alternative output buffer.

p = Outputs to primary input buffer.

s = Outputs to current output buffer.

t = Outputs to terminal.

Syntax

u31ad

code.number

u31bc

implements a PROC subroutine call.

"item-id" is the subroutine to call in the given "file.reference".

Syntax

u31bc

file.reference item-id

u401c

returns the current abs frame number.

u4193

disables output to the spooler, and redirects output to the terminal.

Called from PROC, this performs the exact same process as "printer close" in FlashBASIC. Its function is to 'close' the currently open print job and direct subsequent output the the terminal.

u41a2

returns control to the statement immediately following the most recently executed subroutine call.

u41ad

replaces the string in the primary input buffer with a new string, placed in the next PROC line.

Syntax

u41ad

string

u41bc

is a PROC Subroutine return.

u5193

sends the Bell character ("beep") to the terminal.

u51bc

is a PROC Subroutine return.

u61a2

toggles a system-level flag which controls visible output to the users terminal.

When terminal output is suppressed, this is equivalent to the PROC "ph" command.

The reverse of this operation would be "u3193", which sets terminal output ON regardless of its current setting.

Example

```
pq
u61a2
hsselect md
ston
hsave-list mx<
ston
p
u61a2
```

u61bc

disables terminal output.

This is the reverse of "u3193" and is the equivalent of the PROC "ph" command.

u713c

returns the system serial number.

u71a2

returns the port number of the current process, and places it in the output buffer.

u8193

directs all subsequent output to the printer.

u91a2

transfers control to the specified PROC name in the primary output buffer.

The primary input buffer is unaffected.

Both output buffers are cleared.

Syntax

proc

u91a2

Example

```
001 pq
002 st off
003 hnew.proc
004 u91a2
```

u91bc

transfers control (chains) to another PROC, without affecting the contents of the input buffers.

The target PROC name is listed on the line above the User Exit.

Syntax

hproc.name

u91bc

ua1a2

deletes the data in the primary input buffer from the current pointer to the end of the buffer.

ua1bc

moves entries from the current position in the input buffer to the end.

ud070

returns the current process' account name into the active input buffer.

If "charge-to" is active, the account name is not included.

Example

```
001 pq
002 ud070
003 d0

This displays the current account
```

ue070

returns the current tape reel number into the active input buffer.

user exits, PROC

allows direct access to system functions. The following is a list of user exits commonly used from within Proc.

x

serves two purposes. In a "mainline" (PROC) routine, it functions as a "stop" command and halts the PROC. In a "subroutine", it acts as a "return" statement, and returns control to the next executable line after the one which invoked it.

Syntax

x {text}

u419b

returns the number of ACCESS items selected into a proc parameter.

Processing Codes

describe data base links and data manipulation rules, in dictionaries. The D3 dictionary defines file layouts and data structures. Processing codes are used by various D3 System processors, such as the Update processor (UP), to transform the value of data.

Processing codes are stored in attribute-defining and file-defining items in file dictionaries. The processing codes are found in:

Attribute 7 -- Output Conversion

Attribute 8 -- Correlative

Attribute 14 -- Input Conversion

Output conversion codes manipulate the display or format of data. They are applied just prior to output from UP, the list processor or a FlashBASIC call.

Correlative codes are used by all the system processors. Correlative codes for attributes are applied by the sort or select processor when building a list of item-ids and by UP and the list Processor prior to calling the output conversions. Correlative codes in file-defining items are called when the item is filed, especially from UP. Some correlative codes, such as stamps, item-id generation, and FlashBASIC subroutines, are only called from UP.

More than one processing code can be specified in an attribute by separating the codes with value marks (<Ctrl>v in UP).

Input conversion codes are used by UP to validate data as it is entered. These codes include:

Algebraic functions

Edit patterns

D3/Assembly code executions

FlashBASIC calls

Range limitations

Required fields

Input conversion codes define single or multiple file indexing. This allows cruising through both the current file (that is, the file where the item resides) and other linked files.

Reverse Polish Notation

is a written syntax where the operators follow the operands. For example to total, one and two would be written as: 1 2 +

This notation is used by Hewlett Packard calculators and the D3 f processing code. To total one and two in an "F-correlative", it is written as f'1';'2';+

This notation is easier for a programmer to program, but harder for a user to use. So the "A processing code" was created to support embedded operators, providing the syntax of a'1'+'2'

a (algebraic)

recursive algebraic function which creates algebraic formulas and relational operations consisting of operands and operators; these expressions are used by AQL as well as the b-tree indices.

"a" processing codes are translated into "f" processing codes by the AQL compiler.

The functional operators and operands for the "a" processing code are the same as that of the "f" Processing Code, except as noted.

The "num.digits" argument is optionally used to indicate the number of decimal digits (in the range 0-4) to retain during calculations involving a mixture of whole numbers and numbers with implied decimals, along with "m" (mask) processing codes in the body of the function. The default "num.digits" is zero.

The expressions formed are like FlashBASIC expressions and consist of operands, operators, conditional statements and special functions combined together to yield a single resulting value.

D^3 algebraic processing uses a "last-in first-out" stack. Operands are pushed onto the top of the stack and all existing stack entries shift down. Operations operate on the top first, second, or third stack entries, depending on the operation. Results processing, including additional conversions, take the top value off the stack.

The available operands include:

number — An integer number reference to an attribute count ("ac").
For example:
a3*5
This multiplies the value in attribute 3 by the value in attribute 5.

numberr — A reference to an attribute count, with a "repeat" code, meaning that the first value of the attribute is to be used (repeated) when using this attribute against another multi-valued attribute.
For example:
a3r*5
This multiplies the 1st value in attribute 3 by the 1st value in attribute 5, and returns the result as the first value. Then, it will multiply the 1st value in attribute 3 by the 2nd value in attribute 5, and return it as the second value, and so on.

number(processing.code{]processing.code...})

The "number" is an integer reference to an attribute count ("ac"), which may be immediately passed through any other valid processing code, except another "a" or "f" processing code or an index processing code. Multiple processing codes may be specified and each separate processing code must be delimited by a value mark (represented in the syntax with the "]" character).
For example:
a1(t1,40]mct)

This retrieves attribute 1 of the current item, extracts the first forty characters, and converts each word to upper and lower case.

9998 — Used as an "ac" specification. Returns the sequential item counter.

9999 — Used as an "ac" specification. Returns the size of the item in bytes.

"text" — Any quoted text or numeric string, which is treated as a literal.
For example:
a"Attention: ":1
This concatenates the literal, "Attention: ", to the contents of attribute

1 of the current item.
An example of a numeric constant is as follows:
a3*"12"
This multiplies the value of attribute three of the current item by the constant "12".

n(attrname){(processing.code...)}

In this case, "n" is a literal and indicates that the attribute-defining item within the following set of parentheses is to be retrieved. This is sometimes called an "indirect" reference. This allows a reference to any valid attribute-defining item in the dictionary of the file being accessed. The string returned with this operand may subsequently be passed through one or more other valid processing codes (except another "a" or "f" processing code or an index processing code) as long as each is separated by a value mark. This is where recursion occurs, as the attrname being referenced in the processing code may in fact reference an item which also contains an "a" processing code. For example:

an(city):", ":n(state):" ":n(zip)

This concatenates the contents of the attribute defined as "city" with a comma and a blank, then joins the value of the attribute defined as "state" with a blank and then appends the attribute defined as "zip" to the end of the string.
For example:

an(city):", ":n(state)(tstates;c;;1):" ":n(zip)

This is like the previous example, but this time the value in the "state" attribute is "translated" to the "states" file, returning attribute 1 of the corresponding item.

d — Returns the system date in internal form. For example:

a(d-n(invoice.date)) — This subtracts the value of "invoice.date" from the current system date, which determines the number of days that have elapsed since the date of the invoice.

nb — Returns the number of the break level. There is a maximum of 255 break levels (0 - 254). The 255th level is reserved for the grand total. This works in attribute 7, the "conversion" attribute, only.

nd — Returns the number of detail lines. This works in attribute 7, the "conversion" attribute, only.

ni — Returns the sequential number of the item being processed. This works in attribute 7, the "conversion" attribute, only.

nv — Returns the sequential number of the value being processed. For example:

nv=s(ac#'!')

ns — Returns the sequential number of the subvalue being processed.

t	Returns the system time in internal format, meaning the number of seconds past midnight.
The available operators include:	
Arithmetic Operators:	
+	add
*	multiply
-	subtract
/	divide (note that division operations are rounded down.)
Concatenation Operator:	
: (colon)	Concatenates two operands.
Boolean Operators:	
AND	Logically AND operand one to operand two.
OR	Logically OR operand one to operand two.
Arithmetic Functions:	
r(expression1,expression2)	Returns the remainder of expression1 divided by expression2.
s(expression)	Sums multi-valued results.
Substring References:	
string.expression[begexp,lenexp]	Used to extract a fixed number of characters from a string expression. The "begexp" (beginning position expression) specifies the beginning character position and the "lenexp" (length expression) specifies the length, or number, of characters to retrieve. The begexp and lenexp may be quoted numbers or any expressions which derive numeric results. For example: a":n(phone)["1","3"]:") ":n(phone)["4", "3"]:"-":n(phone)["7","4"] This takes a phone number stored as "7145551212" and outputs it as "(714) 555-1212".
Relational Operators:	Relational operators produce a numeric result. These results are either 1 (one) for a true condition or 0 (zero) for a false condition.
=	Equal to. If operand1 = operand2 then 1 else 0
#	Not equal. If operand1 # operand2 then 1 else 0 < Less than. If operand1 < operand2 then 1 else 0 > Greater than. If operand1 > operand2 then 1 else 0
<=	Less than or equal to. If operand1 <= operand2 then 1 else 0
>=	Greater than or equal to. If operand1 >= operand2 then 1 else 0
Conditionals:	Conditionals are constructed using a syntax very similar to the FlashBASIC "if...then...else" construct and have the general form:

If condition then algebraic function { else algebraic function } {end}

if condition else algebraic function {end}

if condition then algebraic function {end}

If the condition evaluates to zero, it is considered false and, if present, the "else" clause is taken. If the evaluation is false, and the "else" clause is missing, a null is returned.

If the condition evaluates to non-zero, it is considered true, and, if present, the "then" clause is taken. If the evaluation is true, and there is no "then" clause, a null is returned.

If END is not specified, the ELSE is paired with the nearest previous IF. The END is used to close off the current IF and associate the ELSE with the prior IF.

For example:

if 1 then if 2 then "1 and 2 true" end else "1 false"

Note: if 1 is true and 2 is false, null is returned.

Conditional options:

"condition": An algebraic function (a processing code).

"algebraic function": An algebraic function (a processing code).

Examples:

if 1 > 2 then n(attr1) else n(attr2) end

if 1 > 2 then if 3 then 3 else 4

if (if 1 then 2 else 3) then "true"

Syntax

a{num.digits}{;}expression{s}

algebraic function

see the "a" processing code.

b (bridge)

maintains a horizontal relationship (bridge) between an attribute in one item and an item in another file (or the same file) on the system level.

"file.reference"

refers to the file to which the bridge is being built.

"ac1"

contains the attribute in the current item that contains the item-id(s) of the item(s) in the "bridged-to" file. This attribute may be multi-valued in the current file. If it is multi-valued, all items are updated in the "bridged-to" file.

"ac2"

contains the attribute in the "bridged-to" item that is referenced by the bridge. Unless the optional "operator" parameter is specified, this attribute contains the item-id of the current item as a cross-reference to the current file.

If the optional "operator" parameter is specified as an arithmetic operation with an operand "ac3", the attribute contains the result of the arithmetic operation.

"ac3;+|-"

adds or subtracts the value of the attribute specified by "ac3" in the current item to the value in the attribute "ac2" in the "bridged-to" item. The inverse operation is performed on a delete. No backward pointer is maintained in the bridged-to item.

If the current item is deleted, the inverse operation is performed to automatically return the value in "ac2" to its original value.

"d"

is used as an override to delete the current item even if "ac1" is not null. Without this option, the current item may not be deleted if the attribute specified by "ac1" contains a value.

All values of "ac1" in the current item are processed. If a value in "ac1" is changed, the value of "ac2" in the original item specified by "ac1" is deleted, and this value is added to attribute "ac2" of the new item pointed to by "ac1".

The advantage of a bridge correlative is that it performs a function which previously took significant FlashBASIC coding. When an item is added, changed, or deleted, all the bridge correlatives in the file-defining item of the file dictionary are processed for that item.

Syntax

bfile.reference;ac1;ac2{;{ac3;+|-}|d}

Example

```
bentity;1;5
This example places the item-id of the current item into attribute 5 of the
"entity" file. Attribute 1 in the current file contains the item-id in the
entity file.
bproduct;2;4;3;+
This example adds the value of attribute 3 in the current item to the value in
attribute 4 of the bridged-to item (product file). The result of the addition
is stored in attribute 4 of the bridged item.
```

bridge processing code

see the "b" processing code.

c (concatenate)

concatenates elements for output.

Each element may be delimited by a character to appear on output. The ";" (semi-colon) has a special meaning; it does not appear on output.

The elements can be:

- Numeric constants, enclosed in quotes.

- An Attribute position, referenced by its numeric attribute count.

- An alphanumeric character string enclosed in quotes, double quotes, or backslashes. When delimited by semi-colons, strings must be enclosed in quotes. Otherwise, alphanumeric strings are not separated from the other entries by semi-colons (;).

- '*' (asterisk). The asterisk is a special character used to concatenate the resulting value from the last processing code operation.

In the third form listed under syntax, the "x" specifies the separator to appear between the concatenated elements. The separator may be any non-numeric character including a blank or semicolon, but cannot be a system delimiter. The semicolon is a special separator that specifies the elements are to be concatenated with no separation.

Syntax

celement{;element...}

celement{/*}

celement{xelement}{...}

Example

```
c;3;", ";4;" ";5
Concatenates the current item's value of attribute three with a comma and a
blank, attribute four, a space, and attribute five.
```

call

calls a FlashBASIC subroutine from the file-defining item or from an attribute-defining item.

The syntax for calling a subroutine does not specify any parameters, but the subroutine is passed a parameter anyway, so one parameter must be specified in the "subroutine" statement. However, when a subroutine is called from the file-defining item, the entire item body is passed to the subroutine. When a subroutine is called from an attribute-defining item, the value of the calling attribute is passed to the subroutine.

Subroutines can be called from the "correlative", "input-conversion", or "output-conversion" attributes of file-defining items or attribute-defining items.

When a subroutine is called from the "input-conversion" of the file-defining item, control passes to the subroutine before the operator gets into the file via the Update processor. That is, after the UP command has been executed (:u filename) and before the user gets access to the item.

When a subroutine is called from the "correlative" field of the file-defining item, control passes to the subroutine at file time.

Subroutines called from the "output-conversion" attribute are executed on output for redisplay after entry. Subroutines called from the "correlative" dictionary are executed prior to processing. Subroutines called from the "input-conversion" dictionary are executed after input.

Data may be interrogated and modified in the subroutine using the "access" statement.

Error conditions can be set if an "inputerr" statement is executed in the FlashBASIC subroutine.

Syntax

call {cataloged.subroutine}

Example

```
Assume that there is an attribute-defining item that contains a call to a
subroutine called "st.name":
subroutine st.name(st)
states = "alaska,washington,oregon,california,idaho"
codes = "al,wa,or,ca,id"
convert "," to char(254) in codes
locate(st,codes;x) then
st = field(states,",",x)
end else st = "unknown"
return
In this example, the value is passed from Access into the variable, "st". If
```

```
the corresponding string is found, "st" is loaded with the appropriate "code"
and control returns to Access.
```

callx subroutine

a FlashBASIC subroutine call which is specified on attribute 8 (the "correlative" attribute) of the file-defining item and is used to maintain data integrity.

Subroutines invoked with the "callx" processing code are executed when an item in D3 is updated, regardless of which process does the updating, i.e.: the Update processor, FlashBASIC "write" statement, TCL "copy" command, or the "t-load" command.

This processing code differs from the "call" subroutine in that it is called any time an item in the file is updated, versus being called when the item is updated from the Update processor.

The FlashBASIC subroutine must adhere to the following:

The parameter passed is the entire item, which may be updated by the subroutine.

access(3) is identical to the parameter upon entry to the subroutine and CANNOT be updated by the subroutine.

access(1) is the file.variable for the data portion of the file.

access(2) is the file.variable for the dictionary portion of file.

access(10) is the item-id and can not be changed by the subroutine.

access(11) is the name of the file.

Note: access(11) returns the name of the data portion of the file only. If the data file name is different from the dictionary name, the designer must program in the dictionary name in order to use this function.

access(12) is the deleted item flag.

access(16) is the new item flag.

access(20) is the "item changed" flag.

Syntax

callx {cataloged.subroutine}

character update

see the "cu" processing code.

concatenate

see the "c" processing code.

cu (character update)

allows data in a defined attribute with an output-conversion or correlative to be changed character by character. The "cu" processing code is placed in the "input-conversion".

If "cu" is not specified, existing data is erased and nulled when editing is started.

The "cu" processing code is required ONLY when there is another conversion present in the attribute-defining item. For example, having a "cu" processing code within an attribute which contains a date and a corresponding "d" output conversion code in its attribute-defining item allows changing PART of the date field, rather than having to re-enter the entire date.

d (date)

invokes the date conversion function, to convert an external date to internal format, or to convert an internal date to one of many available output formats.

By convention, day "0" (in "internal" format) on the D3 calendar is December 31, 1967. It was selected in 1968 and has never been changed, thereby retaining compatibility. For example, June 26, 1987 has an internal value of 7117, which is the number of days since 12/31/67. It follows that 7118 is June 27, 1987. Dates before 1968 are stored as negative numbers. For example, December 30th, 1967 is "-1".

"yr.pos" specifies the number of display digits for the year. For example, if the internal date is 7117, "d2" returns 26 Jun 87. Valid entries are 0, 1, 2, 3, or 4. The default is 4.

The parameters specified in "skipdel" and "#skips" are functionally equivalent to the "g" (group extract) processing code. They define the position of the date value portion in a multi-part data value and for the most part, are unused.

The "outsep" (output separation character) parameter specifies the character that delimits the month, day and year values. Popular characters to use are "-", as in "12-23-1955", or "/", as in "1/16/1997".

If the "outsep" parameter is omitted, then a " " (space) is assumed.

The output order is: 2-digit day, 3-digit month abbreviation and year. The number of year digits depends on the "yr.pos" parameter (dd mmm yyyy or dd mmm yy).

The following formats can be used to input dates:

Format Description

mm/dd/yy Default - any separator may be used (e.g., 6*26*87).

mm/dd Year is same as system date.

mmdd Year is same as system date (when in "standard" format)

dd Month and year are same as system date. On releases 6.1.0 and above, it is necessary to use the "dn" variant to get this functionality. The normal "d" conversion returns null in this case for compatibility with R83.

dd/mm/yy "European" format.

dd/mm "European" format.

ddd Julian date (1-366).

yyddd Julian date (five digits).

yymmdd Military date if in US mode.

ddmmyy Military date if in European mode.

If the date processing code is used as an input-conversion in the Update processor, a period (".") may be entered to duplicate the last date entered. The "/" (slash) by itself may be used to insert the current day's date.

If the "d" processing code is specified as an input-conversion, no codes are required for input date conversion except for the Julian and European formats, since the format is obvious in all other cases. If the "n" character is added after the "d", then the user can pass the numeric day of the month, and the function returns the correct internal date assuming the current month and year.

When the date processing code is used as an output-conversion, it converts the internal date to an external format as specified by one of the following optional character codes:

The "char.code" may be any of the following:

d Numeric day of the month (1-31).

f Full external date with alphabetic month, day of the month and four-digit year (e.g., October 9, 1990).

i This is a special case used when data is stored in its internal representation. It is typically used on the correlative attribute to convert an external date to its internal equivalent in the pre-processing phase of AQL. Any legal date conversion may be specified on the output-conversion attribute prior to displaying the date.

j Julian day of the year (1-366).

m Numeric month (1-12).

ma Alphabetic month (January, February, ...)

q Numeric quarter (1-4).

w Numeric day of the week (1=Monday, 2=Tuesday, 3=Wednesday, ...).

wa Alphabetic day of week (Sunday, Monday, ...).

y Year (default is 4 digits).

The date code with no parameters converts legal date formats into the internal date (number of days from 12/31/67).

When used in a BASIC program in an oconv() statement, the results may be in either all uppercase, or in mixed Upper/lowercase. If casing is off, then the output will be in Upper/lowercase. If casing is on, then the output will be in Uppercase only.

Syntax

d{n}{yr.pos}{skipdel #skips}{outsep}

d{n}{yr.pos}{char.option}{outsep}

Example

```
data     code  sample output
10594    d     01 Jan 1997
10594    d2-   01-01-97
10594    d-    01-01-1997
10594    d0    01 Jan
10594    d0-   01-01
10594    dd    1
1/1/97   di    10594
10677    dj    84
10594    dm    1
10594    dma   January
10594    dq    1
10594    dw    3
10594    dwa   Wednesday
10594    dy    1997
10594    d2y   97
-10594   d4    29 Dec 1938
```

date

see the "d" processing code.

f (function)

defines functional expression.

f-correlative

used in the conversion, correlative, and input-conversion attributes of attribute-defining items, to create and modify the associated attribute value using various mathematical and logical operations.

Elements in an f-correlative are operands and operators in "Reverse Polish Notation" separated by semicolons. As each operand is encountered scanning from left to right, its value is "pushed" onto a stack. As an example a "push-down" stack is usually compared with the spring loaded mechanism that loads, stores and dispenses saucers in a restaurant.

Operators operate on the top entries in the stack. For example the addition operator "+", "pops" and adds the top two entries in the stack together and then pushes the result back onto the stack.

Operands and operators may appear in any order and after all the operand pushes and operator operations have been processed and the right end of the f-correlative has been reached, what's left on the stack is the result.

Operands:

ac{r}	Attribute value of the attribute whose attribute count is "ac". "r" specifies that the first value or subvalue of an attribute is to be used repeatedly when using it against a multi-valued value or subvalue.
"string"	Operand is the immediate literal string.
cn	Numeric constant "n".
d	System date in external format (dd mmm yyyy)
lpv	Last processing code value (result of the previous processing code).
nb	Break level number. On detail lines, 0 is returned. The break level number has a value of 255 on the grand-total line and a value of 0 (zero) on detail lines. The lowest level control-break in the sentence has a value of 1.
nd	Pushes the number of detail lines since the last "break-on" on the stack. On a detail line, "nd" has a value of 1. On a grand-total line, "nd" equals the item counter; ("nd" can be used to generate averages in conjunction with control breaks).
ni	Pushes the current item counter (number of items listed or selected) on the stack.
ns	Pushes the current sub-multi-value counter on the stack (for columnar listing only).
nv	Pushes the current multi-value counter on the stack (for columnar listing only).
t	Pushes the current time (in "00:00:00" format) on the stack.
Operators:	The term, stack#, is used in the following descriptions to indicate the stack level where the data resides (level 1 being the top, 2 being the next level, etc.).

*{n}	Multiplies stack1 by stack2. If the optional "n" is used, the result is divided by 10 to the power of (n-1). (This is not available as an a-correlative operator. Use the division operator with the appropriate literal value, i.e. /"100".)
/	Divides stack2 by stack1, and returns quotient to stack1. Results of division operations are rounded down.
r	This is the same as "/", except the remainder of the division is returned to stack1.
+	Adds the top two entries in the stack and returns the sum to stack1.
-	Subtracts stack1 from stack2 and returns the result to stack1.
:	Concatenates the string value in stack1 to the end of the string value in stack2 and returns the result to stack1
[ ]	Extract a substring from string value in stack3, using stack2 as the starting character position, and stack1 as the number of characters to extract. The result is placed in stack1.
s	Adds multi-values in stack1 (if any) and returns the sum to stack1.
_ (Underscore)	Exchanges the top two positions in stack.
p	Pushes the top stack value back on the stack, resulting in the same value in stack1 and stack2.
(processing.code)	A standard processing code such as "d" (date), "g" (group), etc., may be specified. It operates on the top stack value and the result replaces the original top stack value.
Relational Operators:	Relational operators compare stack1 to stack2 entries. A "1" (one) is returned to stack1 if the condition evaluates to true. A "0" (zero) is returned to stack1 if the condition is false. The following relational operators are available:
=	true, if stack1 equals stack2
<	true, if stack1 is less than stack2
>	true, if stack1 is greater than stack2
#	true, if stack1 does not equal stack2
=>	true, if stack1 is equal to or greater than stack2
=<	true, if stack1 is equal to or less than stack2
Branching:	
j ~label	branches to label
jf ~label	If the top stack entry is zero, it branches to the specified label.
jt ~label	If the top stack entry is non-zero, it branches to the specified label.
~label	The label is a "~" (tilde) followed by any ASCII characters other than a ";" (semi-colon), blank, Attribute Mark, or Value Mark, which ends the label.

Conditionals: Conditionals are constructed using a syntax very similar to the FlashBASIC "if...then...else" construct and have the general form:

if condition then f.code {else f.code} {end}

if condition else f.code { end }

if condition then f.code { end }

If the condition evaluates to zero, it is considered false and, if present, the "else" clause is taken. If the evaluation is false, and the "else" clause is missing, a null is returned.

If the condition evaluates to non-zero, it is considered true, and, if present, the "then" clause is taken. If the evaluation is true, and there is no "then" clause, a null is returned.

If "end" is not specified, the "else" is paired with the nearest previous "if". The "end" is used to close off the current "if" and associates the "else" with the prior "if".

Syntax

f;element{;element...}

g

extracts one or more groups of contiguous character substrings delimited by the specified character.

"skip.segments" argument specifies the number of delimited groups to skip before performing the extraction. It must be a single digit numeric, if omitted, zero is assumed, and extraction occurs from the beginning of the value.

"delimiter" may be any single printable character except numerics or the minus sign.

"get.segments" specifies the number of contiguous groups to extract from the value.

Syntax

g{skip.segments} delimiter get.segments

Example

```
The following examples illustrate the results obtained from various
combinations of the "g" code if the value being processed was the string:
ca*92714*1000:
g*1         ca
g0*1        ca
g*3         ca*92714*1000
g1*2        92714*1000
g2*1        1000
g2*2        1000
g3*1        (null)
```

group extract

see the "g" processing code.

i (index, File-defining Item)

used in file-defining items to maintain index keys.

The "root.fid" is inserted by the system and MUST not be altered.

When an index is created using the "create-index" verb, the system inserts the "i" code in attribute 8 of the file-defining item along with with the "root.fid" and the "a" processing code.

"root.fid" is the beginning (frame-id) address of the index. If the index is created before data is entered into the file, the "root.fid" is not assigned until the next item is filed.

If data is present and the index processing code is added directly to the file-defining item, the pre-existing items will not be added to the index until they are filed. Any "new" items or pre-existing items which are subsequently filed are automatically added.

If data is present and the "create-index" command is issued, the File-Defining item is automatically updated with the index processing code and each item is added to the index. During the creation of the index, a counter displays on the screen indicating the number of items added to the index.

"a.code" defines the method by which the index keys are generated. It must be a valid "a processing code" and reference at least one attribute count within the file.

Syntax

i{root.fid}a.code

i (index, local)

indicates that an index exists for the current attribute.

It must be defined in the attribute-defining item which corresponds to a "local" b-tree index previously created with a "create-index" command.

Utilizing D3's b-tree indices consists of a two step process:

The first step is the creation of the index. The "create-index" establishes a new B-tree index.

The second step is defining the attribute index correlative on the input-conversion attribute of the dictionary. There may be one (either "local" or "remote") or two (one "local" and one "remote") indexes defined in any given attribute-defining item.

To access data in other files, see "index, remote".

When editing data in the attribute using the Update processor, the operator can "cruise" forward and backward through the items in the file. <ctrl>+d accesses the previous item based on the index; <ctrl>+f accesses the next item. A bell signifies the beginning or end of the index.

There can be more than one index created on any given attribute. If more than 1 index exists, the desired format must be specified.

Example

```
Create an index on attribute 1:
  create-index filename a1
The "i" processing code must be added to the input-conversion attribute of the
corresponding attribute-defining item before the index may be used by the
Update processor.
Create an index consisting of attribute 1 concatenated with attribute 2.
  create-index filename a1:2
```

i (index, remote)

is used as an input-conversion and indicates that this attribute is based on data indexed in another file.

When editing data in this attribute, the user can cruise both forward and backward through the index in the specified secondary file. "<ctrl>+y" accesses the previous attribute value based on the index. "<ctrl>+u" accesses the next attribute value in the secondary file.

Data displayed is subject to the conversion codes in the primary file dictionary. If "a.code" is other than the item-id of the secondary file, a translation ("tfile") processing code must be specified to allow cruising through the secondary file index.

"file.reference" is the name of the secondary file to which the primary file is indexed.

"a.code" indicates the attribute count indexed to in the secondary file.

In the Update processor (UP), the index code allows "zooming". By by pressing <ctrl>+g when the cursor is at the end of the attribute, the user accesses the item addressed by the value of the attribute. A file exit command returns the user to the previous screen. The view displayed is subject to the processing codes in the dictionary of the secondary file. The secondary file attributes to be displayed when zooming can be specified in attribute 15 of the attribute-defining item of the primary file. If they are not specified, the default is to display the attributes defined on the macro dictionary of the remote file-defining item.

Both an "i" ("local index") and an "ifile.reference;a.code" ("remote index") processing code can be specified in an input-conversion. When these two processing codes are specified in the same attribute-defining item, <ctrl>+d and <ctrl>+f function differently depending on the order. If the "ifile.reference;a.code" code is specified before the "i" code, and if either <ctrl>+d or <ctrl>+f is pressed, UP accesses the index specified by the first code in the secondary file and gets the previous or next attribute value for the specified index key. UP uses the index in the secondary file as it would use an index in the primary file. If the "i" code is specified first, the index in the primary file is used when <ctrl>+d or <ctrl>+f is pressed.

Syntax

ifile.reference;a.code

id (assign id)

used by the Update processor (UP) to create new item-ids.

When UP is invoked without a specified item-id, UP creates a new item. When that item is filed, UP creates an item-id for it. If there is an "id" processing code, the item-id is created according to the id function indicated in the corresponding "subcode". If there is no "id" processing code, the item-id is created by concatenating the current date with a system-wide sequence number.

The "subcode" may be any one of the following choices:

"a.code" Uses the specified "a" processing code to create the item-id. Since an "a" processing code can call a FlashBASIC program, any user-defined item-id can be created.

n (integer number) Creates numeric item-ids, starting with "n". As a protective mechanism to avoid the case of "stepping" on existing item-id's, if the item-id already exists in the file, "n" is automatically incremented until a unique item-id is found. The system updates the value in the proccessing code with the latest value.

t Creates new item-ids by concatenating the internal date (currently 4 ASCII decimal numeric digits) with the internal time, measured in seconds. If two item-id's are assigned within the same second, an alphabetic character, starting with the letter "a", is appended to the item-id to ensure that each item-id is unique.

Syntax
id{subcode}

if

Used in dictionary attributes 7 & 8 (conversion and correlative) to generate values based on results of evaluation of expression according to the rules of a-correlative construction.

The if processing code is a stand-alone version of the if conditional allowable within an a-correlative. If the condition evaluates to true (non-zero) the a-correlative following "then" is executed. If there is no "then clause" a null value is returned on true.

If the condtion evaluates to zero (false) the else clause is taken. Multiple complex conditionals can be constructed by using the "end" to close off the current "if" and associate the "else" with the prior "if".

Syntax
if condition {then a-correlative} {else a-correlative} { end }

Example
```
if 2#4 then 4:'**':2
if 14["1","3"]='sco' then 17
```

index, file-defining item

See "i (index, File-defining item)".

index, local

See "i (index, local)".

index, remote

See processing code i.

l (length)

invokes the length function, either restricting processing to values that fall within certain length restrictions or simply displaying the length of a given value.

The "maxlength" parameter indicates that the value will be output only if the length of the string is less than the "maxlength" parameter, otherwise a null is returned. If "maxlength" is 0 (zero), the length of the data is returned.

Syntax
l{maxlength{,maxvalue}}

l0

Example
```
l0
Returns the length of field.
l9
Outputs data if field length is exactly nine characters.
l3,9
Outputs data if field length is greater than three characters and less than
nine characters.
```

m (mask)

a "masking" conversion used for both numeric and text string formatting.

This conversion is quite complex because nearly all of the elements available in its syntax are optional. The elements are listed in the order that the system evaluates the expression. If an element is going to be used, it must be in the same order shown. Each of the elements available is discussed below.

Note that the spaces in the syntax listing are just to provide visual relief -- there are no spaces allowed in the conversion, except for literal spaces (those between quotation marks or with a parenthesized format mask).

The elements include:

just indicates the justification of the string. May either be "l" for left, and "r" or "d" for right justified. Left justification is used primarily for text strings. Right justification is typically used to process decimal numbers.

precision indicates the number of decimal positions to print after the decimal point. It must be between 0 (zero) and 9. When omitted, this defaults to 0 (zero). If 0 (zero) is assumed or specified, no decimal point is printed.

scalefactor indicates the power to which the value is to be descaled. It must be between 0 (zero) and 9. There is no default value.

When a "precision" is specified without a "scalefactor", the "scalefactor" defaults to the "precision" value. ("mr2" is identical to "mr22").

z (a literal "z") suppresses leading zeros (i.e., 0003 becomes 3) and thereby supresses zero balance fields.

, (comma) inserts commas in the thousands and millions position on output.

signcodes alter the normal handling of negative numbers, with the exception of the "d" signcode. Their functions are as follows:

c negative values are followed by "cr".

d positive and zero values are followed by "db".

e negative values are enclosed in angle brackets "<value>". Zero values have a blank preceding and following them.

m negative numbers are followed by a - (minus) sign.

n suppresses (leading) minus sign on negative numbers.

$ appends a "dollar sign" to the value prior to justification.

format.mask may be any combination of literals and special "fill" operators listed below.

#n fills with "n" blanks.

*n fills with "n" asterisks.

%n fills with "n" zeros.

The format.mask may itself be enclosed within parentheses. When it is enclosed in parentheses, any alphabetic characters (and all punctuation characters except "#", "%", "*") may be specified as literal strings.

When masking decimal numbers, the "precision" and "scaling" can be used to round numbers. For example, an internally stored number is "56789". This number has 4 implied decimals. The mask, "mr4", indicates that both the "scalefactor" and "precision" are 4. This gives a result of "5.6789".

The "precision" and "scalefactor" need not be the same. Consider the situation where this number is to be scaled by 4 places, but the result needs 2 decimals of precision. The conversion to accomplish this is, "mr24". Passing "56789" through the conversion, "mr24", first scales by 4 and then rounds to 2 decimals resulting in the decimal number, "5.68".

Syntax

m{just} {{precision} {scalefactor}} {z} {,} {signcode} {$} {format.mask}

Example

```
This chart shows the effect of taking a data value and passing it
through the "oconv" function:
data        conversion          output data
12345       mr2                 123.45
12345       mr02                123
123456      mr2,$               $1,234.56
123456      mr02,$              $1,235
0           mrz                 (none)
0           mr2,z$              (none)
123456      mr2,e               1,234.56
123456      mr2,d               1,234.56db
-123456     mr2,e               <1,234.56>
-123456     mr2,c               1,234.56cr
-123456     mr2,m               1,234.56-
-123456     mr2,n               1,234.56
123456      mr24,               12.35
123456      mr02,$              $1,235
-123456     mr2,$               -$1,234.56
-123456     mr2,e$              <$1,234.56>
-123456     mr2,c$              $1,234.56cr
123456      mr2,c$              $1,234.56
123456      mr(%10)             0000123456
123456      mr2(%10)            0001234.56
123456      mr(*10)             ****123456
123456      mr2(*10)            ***1234.56
123456      mr2,$*12            ***$1,234.56
7145551212  ml((###)###-####)   (714)555-1212
7145551212  ml(###-###-####)    714-555-1212
7145551212  ml(#3-#3-#4)        714-555-1212
123456      mr2$(#10)           $  1234.56
123456      mr2($#10)             $1234.56
123456      mr(#3 plus #3)      123 plus 456
123456      mr24                12.35
```

mc (mask character)

invokes one of many available processing codes available for special processing on numeric and alphabetic strings of characters.

Multiple "mc" conversions may be placed on the same attribute in an attribute-defining item, provided that each is delimited by a value mark.

Syntax

mc{/}code

mc/a (mask character non-alpha)

retrieves only the non-alphabetic characters from a given data value.

Syntax

mc/a

Example

```
string = "1200 Main Street"
crt oconv(string,"mc/a")
This outputs '1200  ' ("1200" followed by two spaces).
```

mc/an (mask character not alpha-numeric)

retrieves only the characters which are neither alphabetic nor numeric from a given data value.

An alternate form of the conversion is "mc/na"

Example

```
string = "1200 Main Street..."
crt oconv(string,"mc/an")
This outputs "  ...".  All alpha-numeric characters are stripped.
```

mc/n (mask character non numeric)

retrieves only the non-numeric characters from a given data value.

Syntax

mc/n

Example

```
string = "1200 Main Street"
crt oconv(string,"mc/n")
This outputs ' Main Street' (note the space before "Main").
```

mc/na (mask character not alpha-numeric)

Mask character non-alpha-numeric

mca (mask character alpha)

retrieves only the alphabetic characters from a given data value.

Example

```
string = "1200 Main Street"
crt oconv(string,"mca")
This outputs 'MainStreet'. The numbers and embedded spaces are stripped.
```

mcan (mask character alpha-numeric)

retrieves only the alphabetic characters and numeric characters from a given data value.

An alternate form of the conversion is "mcna"

Example

```
string = "1200 Main Street..."
crt oconv(string,"mcan")
This outputs '1200MainStreet".  The spaces and the periods are stripped.
```

mcdx (mask decimal to hex)

converts all decimal numbers in a given value to their corresponding hexadecimal equivalent.

Example

```
number = 10
crt "internal = " : iconv(number,"mcdx")
crt "external = " : oconv(number,"mcdx")
This displays 'internal = 16' and 'external = A'.
```

mcl (mask character lower case)

converts all characters in the given value to lower case characters.

Example

```
string = "UPPER And lower"
crt oconv(string,"mcl")
Displays 'upper and lower'.
```

mcn (mask character numeric)

retrieves only the numeric characters from a given data value.

Example

```
string = "1200 Main Street"
crt oconv(string,"mcn")
Outputs '1200'. The letters and embedded spaces are stripped.
```

mcna (mask character alpha-numeric)

Mask character alpha-numeric

mcp (mask characters printable)

converts all non-printable (control) characters in a given value to periods.

Non-printable characters are those between the hexadecimal values x'00'-x'1f' and those between x'7f'-x'fb'. Note that the characters above x'7f' display, but actually have different character meanings as their high order bit is lost.

mcs (mask character sentences)

capitalizes the first letter in each sentence.

Example

```
string = "this is. a demo. of the. MCS code"
crt oconv(string,"mcs")
Displays 'This is. A demo. Of the. MCS code'.
```

mct (mask character upper/lower)

capitalizes the first character after any non-alphabetic character in a given value.

Example

```
string = "this is. a demo. of the. MCT code"
crt oconv(string,"mct")
This displays 'This Is. A Demo. Of The. Mct Code'.
```

mcu (mask character upper case)

converts all characters in the given value to upper case characters.

Example

```
string = "UPPER And lower"
crt oconv(string,"mcu")
Displays 'UPPER AND LOWER'.
```

mcxd (mask hex to decimal)

converts all hexadecimal numbers in a given value to their corresponding decimal equivalent.

Example

```
crt oconv("1AF2","mcxd")
This displays "6898".
```

mi (must input)

used as an input-conversion code and prohibits filing an item if any attribute defined with an "mi" code is null.

ml (mask left)

see "m (mask) Processing Code".

mp (mask packed decimal)

is the processing code for converting a decimal value to a packed decimal value or a packed decimal value to a decimal value.

Example

```
packed.decimal.val = iconv(decimal.val,'mp')
unpack.decimal.val = oconv(packed.decimal.val,'mp')
```

mr (mask right)

see "m (mask) Processing Code".

ms (mask alter sort)

used to alter the normal sort sequence of characters.

Used to sort an attribute in a sequence specified by the item called "seq" in the "messages" file. This is only used on an attribute defined as a sort key, thus, this code must be applied as a correlative (attribute 8). Not available for use in FlashBASIC.

The "seq" item must be manually added to the "messages" file using the following suggested format for the item:

seq (item-id)

001 !"#$%&'()*+,-./0123456789:;<=>?@ABC

DEFGHIJKLMN OPQRSTUVWXYZ[\]^_`abcde

fghijklmnopqrstuvwxyz{|}~

mt (mask time)

invokes the time conversion function to convert external time representations to internal format or to convert internal time values to one of a variety of external time formats.

The "internal" time format is the number of seconds from midnight. The "external" time is 24-hour military format (e.g., 23:25:59) or 12-hour format (e.g., 11:25:59PM).

For input-conversion, the time is entered with 'AM' or 'PM' immediately following the numeric time. If none is entered, 'AM' is assumed. On output, 'AM' or 'PM' is always printed immediately following the numeric time.

Note: "12:00AM" is considered midnight. "12:00PM" is considered noon. 'AM' and 'PM' are ignored on input if the "mt" code is specified. Illegal values are converted to null on input.

When used as an input-conversion, the "mt" code validates the time and converts it to the internal format of seconds from midnight. When updating a field controlled by an "mt" conversion, the "/" (slash) character can be used to insert the current time into the attribute.

"h" specifies a 12 hour clock. Default is (military) 24 hour clock.

"s" includes seconds in output value. If "s" is omitted, seconds are not listed on output.

Syntax

mt{{h}{s}}

Example

```
data   conversion   ouput
3600     mt         01:00
3600     mth        01:00am
3600     mts        01:00:00
3600     mths       01:00:00am
46800    mt         13:00
46800    mth        01:00pm
3630     mts        01:00:30
46800    mths       01:00:00pm
```

must input

see the "mi" processing code.

mx (mask ASCII to hex)

converts all ASCII characters in a given value to their corresponding ASCII-hexadecimal equivalent.

The characters are converted one at a time to 2-digit hexadecimal numbers.

The "my" code is the inverse of the "mx" code.

Example

```
The following table illustrates several mx conversions:
data      conversion   output
----      ----------   ----------
123          mx        313233
abc          mx        616263
a dog        mx        6120646F67
```

my (mask hex to ASCII)

converts all ASCII-hexidecimal characters in a given value to their corresponding ASCII equivalent.

The characters are converted two at a time to 1-digit ASCII characters.

The "mx" code is the inverse of the "my" code.

Example

```
--------  ----------     ------
313233        my          123
616263        my          abc
6120646F67    my          a dog
```

o (sort values ascending)

sorts the values in the specified attribute(s) in ascending order. The values are sorted when the item is filed.

While this CAN be used in the "correlative" attribute of an attribute-defining item, it causes the screen to refresh with each value added. To prevent this from occurring, it is advised that this processing code be placed on the "correlative" attribute of the file-defining item using the format listed below.

1) Attribute-Defining Items:

Syntax: o

Where Used:

correlative: used here

input-conversion: aborts with illegal use message

output-conversion: aborts with illegal use message

Function:

After a value is entered, the screen is redisplayed with the value placed in ascending alphabetic (a-z) order.

The sort does NOT take place until a value is added to an attribute. If the processing code is added to an existing attribute, the attribute can be displayed and filed without the attribute getting sorted. If a value is added, the attribute is sorted.

2) File-Defining Items:

Syntax: o{;}attribute.number{;attribute.number...}

In file-defining items, the "o" processing code may only be used in the "correlative" attribute. The first semi-colon is optional.

Function:

When the item is filed, the values in the attribute(s) specified by "attribute.number" is/are sorted and replaced in ascending alphabetic (a-z) order.

If an attribute is not already sorted, adding this processing code sorts the attribute the next time the item is filed.

Syntax

o

o{;}attribute.number{;attribute.number...}

p (pattern match)

tests a value and determines if it matches a pre-determined pattern "matchstring" composed of alphabetic, numeric or literal characters.

The "matchstring" may be a composite of literals and/or match operators, appended to length specifications. To be accepted for processing, the value must be the exact length of the length parameter. If the data does not match, null is returned.

The "p" processing code may be used as an output-conversion or input-conversion. As an input-conversion, the "p" code verifies input.

Match operators:

na "n" alphabetic characters only.

mn "m" numeric characters only.

nx "n" alphanumeric characters.

"text" any quoted text string. If a literal is specified, the "p" code tests for an exact match to that literal.

In all of the above cases which support a number, the exact number must be met or found to accept the data. "0" (zero) as the number allows for any number, including zero, to be accepted.

Multiple matchstring parameters will accept the data if any of the matchstrings are met. Pattern match operators must be separated by semicolons.

If the data matches the pattern exactly, the data is returned. If the data does not match, null is returned.

The "p" code can be used as an output-conversion or input-conversion. As an input-conversion, the "p" code verifies input. Multiple pattern match.strings must be separated by semicolons.

Syntax

p(matchstring){;(matchstring)...}

pattern matching

see the "p" processing code.

r (range)

invokes the range function, and restricts processing to those values which fall within given minimum and maximum range restrictions.

Multiple range specifications allow display of the data value if the data value falls between any of the range pair sets. In using multiple range set specifications on negative values, the ranges should be in ascending sequence, beginning with the lowest.

Syntax

rmin,max{;min,max...}

range

see the "r" processing code.

s (substitution)

substitutes either a data value from the referenced attribute or literal text when the current data value is null or zero.

The "s" processing code returns the value defined by "attrnum2" if the value for conversion is null or zero. Otherwise, the value of "attrnum1" is returned. "attrnum1" and "attrnum2" may both be an attribute number or text string.

The "*" is used to display the original data value, if it is not null or zero. Otherwise, the second argument is displayed. This feature may be embedded within an "f" processing code in the form: fattrnum(s;*;'text') It also may be embedded within an "a" processing code.

Syntax

s;attrnum1;attrnum2

s;'text';'text'

s;*;'text'

Example

```
Substitution is used in an "a (algebraic)" processing code as an alternative
to the THEN/ELSE construct.
The following A correlatives produce identical results:
aif n(sales)>="1000000" then "made it" else "trouble"
a(n(sales)>="1000000")(s;"made it";"trouble")
The following checks for the existance of a shipping address in attribute 20.
If not there, show the billing address in attribute 10.
a(20)(S;*;10)
```

substitution

Substitution function.

t (text extraction)

provides a facility for extracting any number of characters from an attribute, starting at the position specified in "startcolumn", for a length of "length".

If the startcolumn parameter is not specified, the extraction begins from the same end of the string as the justification attribute (9), either l (left) or r (right).

Columns are always counted from left to right, regardless of the justification specified in the attribute-defining item.

Both parameters must be numeric.

Syntax

t{startcolumn,}length

Example

```
t3,10
This extracts 10 characters, beginning from the 3rd position of the string.
```

t (translate)

uses the presented value as an item-id and attempts to retrieve the corresponding item in another file to obtain a value from one of its attributes to substitute.

The attribute referenced in attribute 2 of the attribute-defining item is assumed to be an item-id in the specified file.reference, except in cases where the attribute count is "derived", such as in the case "f;3;2;*(tabc;x;;1)".

"file.reference" is the name of the file used for the translation.

"subcode" designates how to behave if the requested item is not on the "lookup" file. See available "subcodes" listed under "options".

"vc" (value count) is used with multi-valued attributes, and specifies the multi-value position to be returned. For example, if an attribute contains six multi-values, and "vc" is 4, the fourth multi-value is returned. If the specified multi-value contains subvalues, the subvalues are returned concatenated with blanks. Alternately, an "*" (asterisk) may be specified as the "vc", when following the "c" or "x" subcode. The "*" is used when translating a multi-valued attribute into a multi-valued attribute. For example, when processing the third value of a multi-valued attribute, the third value of the translated attribute is returned. If the corresponding value does not exist, a null is returned.

If "vc" is not specified, all the multi-values in the attribute are returned, concatenated with blanks.

"inac" (input attribute count) specifies the attribute number used for input translation. The data value is used as the item-id in the file specified by file.reference, and the translated value is retrieved from the attribute specified by "inac". If "inac" is not specified, no input translation takes place.

"outac" (output attribute count) specifies the attribute number used for output translation. When a listing is generated using AQL, the attribute values specified are looked up in the file specified by "file.reference" and the attribute specified by "outac" is listed instead of the original value.

"breakac" (break attribute count) if specified, is used instead of the "outac" during the listing of break-on and total lines.

Syntax

t{dict }file.reference;subcode{[vc|*]}; inac;outac{;breakac}

Options

c Converts, if possible. If the item in the translate file does not exist or the specified attribute is null, the original value is used.

i Verifies input only. This subcode functions as a "v" subcode for input, and as a "c" subcode for output.

o Verifies output only. This subcode functions as a "c" subcode for input and as a "v" subcode for output.

v Specifies that a conversion item must exist and the specified attribute must have a value, otherwise an error message displays. This is for verification only, and the item-id is the only data returned.

x Converts, if possible. Otherwise, it returns a null value.

text extraction

text extraction function.

translate code

see "t (translate)".

u (user exit)

see "user exits, AQL".

v (value limit)

specifies the maximum number of values allowed in an attribute when as an input-conversion.

"number" specifies the number of values allowed. "number" may be any integer.

If "number" is "0" (zero), data cannot be entered and any data present will be nulled.

If the "v" processing code is not defined, there is no limit to the number of values available in an attribute.

Syntax

vnumber

Example

```
v3
Allows only three values to be entered in this attribute.
```

v (within)

used by the "within" connective and must be placed in the correlative attribute (8) of the file-defining item ("d-pointer").

Syntax

v;;n

Example

```
see "within" connective.
```

x (display only)

used as an input-conversion and indicates that the specified attribute is for display only. The attribute cannot be updated.

x (update stamp)

provides an audit trail for items that are updated.

type specifies the type of stamp to use, and may be any of the following choices:

"a" places the name of the user in the specified attribute when the item is filed.

"d" places the current date in the specified attribute when the item is filed.

"s" adds the total time (in seconds) the item was in UP into the specified attribute. The value is cumulative. Each time the item is updated, the current usage is added to the previous time.

"t" places the current time in the specified attribute when the item is filed.

"ac" specifies the attribute count into which the stamp is to be placed.

"v" specifies that each time the item is filed, the stamp is to be inserted as a new value. If "v" is not specified, the current stamp overwrites the previous entry. The "v" has no effect with the "s" option.

The audit stamp codes may be combined into more complex expressions, such as "xa1d2s3v". This example will stamp attribute 1 with the user, attribute 2 with the date, and attribute 3 with the time, adding subsequent stamps as multi-values. When stamp specifications are combined, the system will not add another stamp if the user and date are the same. For example, if a stamp already exists for user 'dm' on 09/28/94, and the same user updates the item on the same date, another stamp will not be added. This does not apply when specifying stamps separately.

Syntax

xtype ac{v}

Example

```
1)  xa14
This places the user-id in attribute 14.
3)  xa3d4t5
This complex expression adds the user-id in attribute 3, the current date in
attribute 4, and the current time in attribute 5.
```

xc

centers the input string in a field of spaces.

If column.width is specified, then the "xc" code will center the text within a field of column.width characters. If no column.width is specified, then the processing code looks forward to the attribute length specifier in the attribute defininig item.

Syntax

xc{column.width}

xi

forces any further conversions to be treated as if they were an input conversion.

xo

forces all following conversions to be treated as output conversions.

xr

produces a running sub-total of an output column on an AQL listing or report.

The attribute definition must have "xr" specified as an output conversion only. In order to get a running total, the attribute containing the "xr" conversion code must be "total"ed. if not, the conversion has no effect.

The sub-total may be restricted to any number of control break levels by specifying the optional break.level parameter.

Syntax

xr{break.level}

Example

```
Assume that the following processing code was present in the output-
conversions of the attribute-defining item myattr":
 xr1
This will cause any access statements using this attribute in a "total" clause
to reset the running total to 0 after the first and higher break levels.
```

xs

see x (update stamp)

xt

toggles the conversion direction from an input to an output conversion or vice-versa.

xt

see x (update stamp)

y

stamps user, pib, and/or time/date update information into an internal item header from all processors.

To be effective, the "y" must be followed by one or more of the following letters to stamp information at update time:

u User ID

p User Pib

t Time/Date Information

Unlike the "x" code, the "y" code updates stamp information when an item is updated from any processor. Additionally, this stamp information is stored outside the normal item body in a special textual header.

To read the stamp information generated by the "y" code, prefix the filename with the word "hdr:". The header information can then be examined from FlashBASIC, AQL, or any other processor. Note that the stamp information CANNOT be written outside of the normal update mechanism.

The "save" verb saves all header information on the save tape, and "restore" automatically restores this information. However, "t-dump" and "t-load" do not save/restore the header. Instead, "t-load" generates new header information as if the items were being updated normally.

Syntax

y{u}{p}{t}

za

displays a D^3 item in a format compatible with assembly source code:

{label} opcode {arguments,,,} {Comment}

zc (zip code)

displays the city, state, postal (zip) code and country during zip code entry or output.

In order to use the "zc" processing code, there must be three files defined. The files used by this code are : "zcf", "state", and "country".

zip code

see the "zc" processing code.

Runoff

facilitates the preparation and maintenance of textual material such as memos, manuals, etc.

The "runoff" command invokes the output function of the "Runoff Processor". Text stored with embedded commands is formatted for output to a terminal or printer. See "commands, runoff".

Runoff source text contains commands which control justification, page headings and footings, numbering, spacing and capitalization.

Textual material prepared with Runoff may be easily edited and corrected with the "line editor" or "Update processor" and then reprinted with Runoff.

Runoff also provides the capability of combining separate textual material into a single report and inserting duplicate text into different reports.

Multiple input items are treated as a single source text file.

A source text item may contain a command which causes Runoff to "chain" to another file item. This makes it possible to "link" file items together without doing a "select" or "sselect".

Items included in "itemlist" may chain to other items within the same file. When the "chain" ends, processing continues with the next item from the "itemlist".

A source text item may also contain a command which causes Runoff to "read" a second file item and then resume processing of the first item. This makes it possible to insert the text from a single file item in the output from many other file items.

Runoff commands are stored along with the textual material in the source file, and each Runoff command must be preceded by a period.

Syntax
runoff file.reference itemlist* {(options)}

Options
* see "options: Runoff".

c

suppresses the ".chain" and ".read" commands.

Syntax
runoff file.reference {itemlist*} (c)

Example
```
runoff doc chainletter (c
```

i

includes the item-id in the output.

This command is helpful for tracing chained sequences.

Syntax
runoff file.reference {itemlist*} (i)

j

suppresses highlights.

Syntax
runoff file.reference {itemlist*} (j)

n

prevents pausing at page breaks when outputting to the terminal.

Syntax
runoff file.reference {itemlist*} (n)

number

sets the number of times boldface or underscore characters are overprinted.

Syntax
runoff file.reference {itemlist*} (number)

Options: Runoff

alter the "normal" behavior of Runoff output.

Options are always presented at the very end of the Runoff command, and must be preceded by a "(" (left parenthesis). The ")" (right parenthesis) is optional. Multiple options may be specified; no delimiter is required between them.

Options
c Suppresses the ".chain" and ".read" commands.

i Outputs the name of the next item-id to be output.

j Suppresses hightlights.

n Prevents pausing at page breaks when outputting to the terminal.

p Directs output to system printer, via the Spooler.

s Suppresses boldface, underline and fonts and default headings.

u Forces all output to upper-case.

Example
```
runoff doc termination.notice (ps
```

p

directs output to the Spooler.

Syntax
runoff file.reference {itemlist*} (p)

s

suppresses boldface, underline and modes.

Syntax
runoff file.reference {itemlist*} (s)

u

forces the output to upper-case.

Syntax
runoff file.reference {itemlist*} (u)

&

underlines the following character.

The character to be underlined must immediately follow the "&" character.

If the "&" character is followed by a blank, the character is printed "as is", meaning, that it prints "&".

Syntax
&x

&

turns off the underline mode.

&^

puts Runoff in "underline mode" until an ampersand followed immediately by a back-slash ("&\") is encountered.

Example
```
&^Special Control Characters&\ are needed
```

*****

treats the entire line following the command as a remark. Any text following the command is not output by Runoff.

Syntax
.* {text}

Example
```
The following text is ignored. .*  comments begin here.
```

<

causes the following word to begin at the next defined tab stop.

The tab stop positions are defined using the ".set tabs" command.

Tab characters are only in effect in the ".nofill" mode.

>

causes the following word to end at the next defined tab stop.

The tab stop positions are defined using the ".set tabs" command. Tab characters are only in effect in the ".nofill" mode.

@

boldfaces the following character.

The character to be boldfaced must immediately follow the "@" character.

If the "@" character is followed by a blank, the character is printed "as is", meaning, that it prints "@".

Syntax
@character

@

turns off boldface mode.

@^

enables boldface mode.

Runoff remains in boldface mode until a "@\" is encountered.

The number of times the boldface letters are overprinted may be set by using the numeric option of the Runoff verb.

b

outputs a partially filled line before processing the next line.

Syntax

.b

begin page

forces a page break.

".bp", terminates the current page, prints the optional footing, ejects a page, increments the page counter, and prints the optional heading.

The text immediately following the ".begin page" command is printed on the next page. (see "nofill")

".begin page" must begin in position one of the attribute. If it is in the middle of an attribute, it is ignored and printed as text.

Syntax

.begin page

.bp

Example

```
.nf
This is page one.
.begin page
This is page two attribute 1.
This is page two attribute 2.
.bp
This is page three.
```

box

defines the left and right margin boundaries for a "box" around the text, and encloses the following text in a "box" of vertical bars on the output page.

The ".box off" command terminates the box.

Syntax

.box leftmargin,rightmargin

Example

```
.nf
.box 6,16
Note: Example A is shown on second page.
.box off
```

box off

turns off the "box" effect previously started with a ".box" command.

Syntax

.box off

Example

```
.nf
.box 6,16
Note: Example A is shown on page 2.
.box off
```

bp

"begin page".

break

creates a new paragraph.

The text following the command is output in the newly created paragraph.

This is a method of "flowing" text (DTP terminology), starting a new paragraph, without a new attribute. ".break" may appear anywhere within the document.

Syntax

.break

Example

```
For graphic representation see below: .break Example A
Produces the following output:
For graphic representation see below:
Example A.
```

c

"center".

capitalize sentences

capitalizes sentences following a period (.), exclamation point t (!), question mark (?), colon (:), or a semicolon (;) that is followed by one or more spaces.

If the first word of the sentence is already capitalized, it remains capitalized.

The ".nocapitalize sentences" command disables the ".capitalize sentences" command.

Syntax

.capitalize sentences

.cs

Example

```
.capitalize sentences
this is the first sentence.  this is the second. And, this is the third.
The output from this example is:
"This is the first sentence.  This is the second. And, this is the third."
```

center

outputs the next line of the document in "nofill" mode and centers it on the next line of output.

This command causes a "break" to occur.

Syntax

.c{enter}

Example

```
.center
Heading A
The words "Heading A" are centered.
```

chain

transfers control to the specified item-id.

If the file.reference is not specified, the current file is used. Control does not return to the source item.

This command is particularly useful for generating form letters. For example, it may be necessary to insert the name and address of each recipient of the letter from a separate file.

A "sselect" statement is used to extract the relevant data from the file and save it in a list. A series of ."readnext" statements inserts the data into the text of the letter. At the end of the letter, a ".chain" command may be used to restart the next letter. When the list is exhausted, Runoff stops.

Syntax

.chain {file.reference} item-id

Example

```
.chain next.doc
The item "next.doc" is processed at the end of the document which called it.
```

chapter

ejects a page, increments the page and chapter counters and outputs the literal, "Chapter", followed by the optional text, centered between the current margins.

The chapter, page number, and text are accumulated for the table of contents.

Syntax

.chapter text

contents

outputs the current table of contents, as defined by the previous ".chapter" and ".section" commands.

Syntax

.contents

Example

```
.chapter
Runoff
.chapter
OP
.contents
The output of this example is:
Chapter 1
Runoff
(page eject)
Chapter 2
OP
(page eject)
Table of Contents
section                                          page
1        Runoff                                   1
2        OP                                       2
```

control functions

are special characters available to the Runoff processor:

@, @^, @\, &, &^, &\, ^, ^^, \, \\, _, <, >.

Runoff features Special Control Characters for Upper/Lower Case Control, Underlining, Boldface Printing, Tabbing, and Special Character Override.

crt

directs text unconditionally to the terminal screen.

Syntax

.crt text

cs

"capitalize sentences".

ec

"end case".

end case

disables any previous ".upper case" or ".lower case" commands.

Syntax

.end case

.ec

Example

```
.upper case
this is the first sentence.
this is the second.
.end case
```

f

"fill".

fill

enters "line fill" mode.

Words are processed until there are enough to fill a line without overflowing it.

If justifiction mode is on, Runoff inserts spaces in the line at random to make the right margin line up.

".fill" is one of the ".standard" settings.

The ".nofill" command is used to terminate the ".fill" mode.

Syntax

.f{ill}

Example

```
.fill
This prints on line one,
and so does this.
The output appears as follows:
This prints on line one, and so does this.
```

footing

designates a text string composed of literals and special options to output at the bottom of each page.

If the line following the ".footing" command is null, the footing is suppressed.

Syntax

.footing {{text} {'options'}...}

Options

* see "options: footing".

Example

```
.footing "'lc'page 'pn' printed at 'tlc'"
```

heading

designates a text string composed of literals and special options to output at the bottom of each page. If the line following the ".heading" command is null, the heading is suppressed.

Syntax

.heading {{text} {'options'}...}

hilite

prints the optionally-specified character along the right margin.

If the character is omitted, or the word "off" appears after the command, the ".hilite" command is disabled.

Any character, other than a period (.), may be specified as the highlight character.

The ".hilite off" command terminates the ".hilite" command.

Syntax

.hilite {character}

.hilite off

Example

```
.hilite *
This produces an asterisk on the right margin.
```

hilite off

disables the ".hilite" command.

Syntax

.hilite off

Example

```
.hilite off
```

i

see "indent".

im

see "indent margin".

indent

indents the following text "number.spaces" spaces from the left margin.

If "number.spaces" is negative, the line begins to the left of the previously set left margin.

If "number.spaces" is not specified, it defaults to 1.

If the command is in the middle of an attribute, the preceding text is output and the next word begins a new paragraph that is indented "number.spaces" spaces.

All lines following the first line are output at the left margin. Only the first line is affected by the indent command.

Syntax

.i{ndent} {-}number.spaces

indent margin

increments the left margin by the specified number of spaces.

The "number.spaces" may contain a negative number to move the left margin farther to the left.

The alternate form is ".im".

Syntax

.indent margin {-}number.spaces

.im {-}number.spaces

index

places the text and the current page number into the document's index.

The text must be enclosed in double quotes if it contains embedded blanks.

Once a ".save-index" command is encountered, index references are written into a file and not saved in memory.

Syntax

.index text

input

accepts and processes text input from the terminal.

The text entered is output as though it had been stored in the document.

Syntax

.input

Example

```
Dear .input,
You are my only one.
Love,
Pat
 .chain love.letter
```

j

see "justify".

justify

enables ".fill" and ".justify" modes.

Runoff fills each output line by adding successive words from the source text until another word does not fit on the line.

The line is then justified by inserting blank spaces between words at random until the last word in the line exactly meets the right margin.

".justify" is one of the ".standard" settings.

The ".nojustify" command terminates the ".justify" command.

Syntax

.j{ustify}

lc

see "lower case".

left margin

sets the left margin at a specific number of spaces from the left edge of the paper.

The left margin is automatically set to zero (0) in the ".standard" command.

Syntax

.left margin marginsetting

.lm marginsetting

line length

sets the line length a specified number of characters from the left margin.

The "number.characters" plus the left margin setting, (minus one), denotes the right margin position.

Line length is automatically set to seventy (70) in the ".standard" command.

Syntax

.line length number.characters

lm

see "left margin".

lower case

outputs text in lower case characters.

The ".end case" command terminates the ".lower case" command.

Syntax

.lower case

.lc

Example

```
.lower case
THIS TEXT WILL BE PRINTED IN LOWER CASE LETTERS.

This produces :
This text will be printed in lower case letters.
```

lptr

directs output to system printer, via the Spooler.

Syntax

.lptr

lx

sets the "auto-tab" to the given "column.number".

Syntax

.lx column.number

ncs

see "nocapitalize sentences".

nf

see "nofill".

nj

see "nojustify".

nocapitalize sentences

deactivates the automatic capitalization of sentences mode.

Syntax

.nocapitalize sentences

.ncs

Example

```
.capitalize sentences
this is the first sentence.  this is the second. And, this is the third.
.nocapitalize sentences
this is the final sentence.
The output from this example is:
"This is the first sentence.  This is the second. And, this is the third."
this is the final sentence.
```

nofill

resets both the ."justify" and ".fill" modes.

Removal of extra spaces does not take place, and end-of-lines are not stripped from the input.

Output lines are not "filled" nor are right margins justified.

This command causes a ".break" (new paragraph).

Syntax

.nofill

.nf

nojustify

disables the justification mode.

The ".fill" mode remains in effect.

Syntax

.nojustify

.nj

nopage

outputs without pausing at the end of each page on output directed to a crt only.

The equivalent Runoff option is "n".

Syntax
.nopage

noparagraph

command turns off the ".paragraph" command.

Syntax
.noparagraph

p

see "paragraph".

page number

sets the current page number to a specified number.

If "pagenumber" is omitted, one ("1") is assumed.

Syntax
.page number pagenumber

paper length

specifies the maximum number of print lines per output page.

Syntax
.paper length paperlength

paragraph

specifies the number of spaces for paragraph indentation.

Any blank line or any line beginning with a blank is indented the specified number of spaces.

The "number.spaces" may contain a negative number to cause the paragraph to "outdent".

Syntax
.p{aragraph} {-}number.spaces

pfile

creates a separate Spooler entry containing the table of contents created with the ".contents" command.

Directs subsequent output to the specified "print.file.number", just like the "print on" statement in FlashBASIC.

Syntax
.pfile print.file.number

print

prints the next line of text on the terminal.

This command is used in conjunction with the ".input" command

Example
```
.print
Enter the subjects name:
.input
```

print index

prints the sorted index of words and page numbers.

The index is sorted in alphabetical order, and printed two columns per page.

This command changes the tab settings, and issues a ".begin page" command.

read

transfers control to the specified Runoff item for output and returns control upon completion.

This command causes Runoff to read the "item-id" indicated. If no "file.reference" is given, the item is read from the same file as the item being processed.

The text input from this item is processed and output without any parameter or mode changes. After processing this item, Runoff resumes input with the next line of the current source of input.

This command does not cause a "break".

The ".read" command scans the string following the command, looking for an "item-id" or a file name. The legal delimiter for the "item-id" or "file.reference" is a blank or a period.

If there is more than one blank-delimited field following the ".read" command, the next-to-the-last field is taken to be the file name, and that file is opened. The last field delimited with a blank is considered the "item-id", and it is retrieved by the Runoff processor to be executed next.

If the statement is a ".read", the processor eventually returns to the current item and continues processing at the beginning of the next line in the item. No statements which occur after the ".read" statement in the line are executed.

A comment statement may be included after the ".read". The line is considered exhausted when the processor encounters an end-of-line mark, or when it encounters a period preceeded by a space.

Syntax

.read {file.reference} item-id

readnext

extracts the top entry of an active list created by one of the list-generating commands prior to output and inserts the value into the text of the item.

This command is used to read data from an "active list". It has an effect only if, prior to entering Runoff, a "select", "sselect", "qselect" or "get-list" statement has been entered, which selects a list of values. See "active list".

Each ".readnext" command in Runoff extracts the next value from the active list and places it in the text stream.

".readnext" does not cause a "break".

If there is no "active list", or when the list is exhausted, the ".readnext" command causes a termination of Runoff, and a return to TCL.

Runoff commands

available Runoff commands. For consistency, all Runoff commands are typed in lower case letters.

save index

saves entries defined in previous ".index" commands in the specified "file.reference", using the item-id of the current Runoff item.

The "file.reference" of the index must be different than the current "file.reference" to avoid destroying data.

This command causes chapter and page number information of indexed data in a text to be saved in a separate file.

Each index entry is stored as an individual item using the index entry as the item-Id, the chapter (where that index entry is referenced) as the first attribute and the page number as the second attribute.

Multiple values are stored in these attributes as multiple references to the same index entry. The resulting file may then be operated on by the AQL processor to generate listings for the chapter and page number information of all indexed data in a text.

The "file.reference" is the name of the file in which the chapter and page information is to be stored.

The ".save index" command is placed in the text item itself and must precede the ".index" commands.

In short, only that indexed data which has been preceded by the ".save index" command is saved in the specified file.

Syntax

.save index file.reference

section

handles automatic chapter section numbering and formatting, in conjunction with the ".chapter" command.

The ".section" command automatically starts the next section at depth "sect.number", where "sect.number" is the range 1-5.

The text is printed following the section number and a ".skip" occurs. The text is recorded as the section heading in the Table of Contents.

If no text appears on the ".section" command, no ".skip" occurs and the section is not recorded in the Table of Contents.

Section numbers are incremented automatically and the section number is printed in the form "1.2.3.4.5", with "n" digits printed.

Conventionally the ".section" command is followed by a blank line before the next paragraph starts.

Since the ".section" command causes a ".break" which terminates the preceding paragraph, and since the text following the ".section" command is placed immediately into an ouput line and output prior to a consideration of the next line, the blank line after the ".section" command can be avoided by not indenting the first line of the next paragraph. That is, if the processor does not know that the next line starts a paragraph, it will not skip a line. It may be necessary to use an ".indent margin" if paragraph indentation is desired, however.

Syntax

.section sect.number text

set tabs

clears previous tabstops and specifies tabstop locations at specified column position{s}.

The tabstops (up to 30) must be greater than zero and in increasing order. They indicate tabstop positions relative to the left margin.

Tabs are only in effect in ."nofill" mode.

The left-tab character "(<)" causes the next word to start at the next tab position. The right-tab character "(>)" causes the next word to end at the next tab position.

If a tab character appears at a point in the line where no further tab stops have been set, the tab character is ignored.

Syntax

.set tabs tabstop{,tabstop...}

sk

see "skip".

skip

outputs a specific number of blank lines on a page, taking into account any previously specified ".spacing" commands. If "number.lines" is omitted, "number.lines" defaults to one ("1").

Syntax

.sk{ip} {number.lines}

sp

see "space".

space

outputs a specific number of blank lines on a page, regardless of any previously specified ".spacing" commands.

If "number.lines" is omitted, "number.lines" skip defaults to one ("1").

Syntax

.sp{ace} number.lines

spacing

specifies inter-line spacing to the specified number of lines (e.g. ".spacing 2" sets double spacing). See the ".skip" command.

Syntax

.spacing number.lines

standard

defines the default parameter and mode settings for Runoff text.

Default parameters:

".capitalize sentences", ".crt", ".fill", ".footing (null)", ".heading (null)", ".justify", ".left margin 0", ".line length 70", ".paragraph 5", and ".upper case".

Syntax
.standard

test page

tests the number of lines left on the current page and determines whether text is output on the current or next page.

If the "number.lines" parameter is less than the count returned, the following text is output on the current page. Otherwise, a page is ejected and the text is output on the new page.

This prevents blocks of text from being split across a page boundary.

Syntax
.test page number.lines

.tp number.lines

tp

see "test page".

uc

see "upper case".

upper case

outputs text in upper case characters, unless specifically altered by the lowercase special control character, "(\)".

The ".end case" command terminates the ".upper case" command.

Syntax
.upper case

.uc

Example
```
 .upper case
this is the first sentence.
this is the second.
 .end case
```

****

prints the following character in lower case.

The character to be printed in lower case must immediately follow the "\" character.

If the "\" character is followed by a blank, the character is printed "as is", meaning that it prints "\".

Syntax
\x

****

turns off upper case mode.

"\\" causes the text to switch to lower case in the same way that ".lower case" causes the switch, except that "\\" may be imbedded in a line.

Turning off this function requires the ".end case" command.

^

prints the following character in upper case.

The character to be printed in upper case must immediately follow the "^" character.

If the "^" character is followed by a blank, the character is printed "as is", meaning that it prints a "^".

Syntax

^character

^^

activates the upper case mode.

"^^" causes the text to switch to upper case in the same way that ".upper case" causes the switch, except that "^^" may be imbedded in a line.

Turning off this function requires the ".end case" command.

_

treats the following character as a text character, rather than as a control character.

Example

```
This prints one underscore: __.
```

Spooler

controls all output that is sent to a printer.

SPOOLER is an acronym derived from Simultaneous Peripheral Output and On-Line Error Recovery sub-system.

Any system which allows multiple users to generate printed output and direct it to one or more logical or physical printers must have a spooler. The Spooler controls the creation of each print job and sees that it is either sent to a printer or held in a storage area. It also controls the relationship between physical hardware and logical printers. On some implementations, the Spooler controls the parallel printer(s) as well.

Depending on the printer assignments and the status of the printer, the output may be printed immediately, sent to tape, placed in a queue for later printing, or placed in a hold file.

There are three major divisions of interest in the Spooler: form queues, logical printers and physical printers. Each user is assigned to a form queue, either by direction or by default. When the user creates a report, it becomes an element in a table of currently active reports for the assigned form queue. There are status flags which indicate if the report is to be printed, suppressed, held, sent to tape or left open at the end-of-job.

The relationship between the physical printers (hardware) (see "startptr") and the form queues is established when a logical printer is "started". At that time, the user designates the name (number) of a logical printer, the physical hardware device it will run on, and the form queue(s) that it will service.

The Spooler directs the items in the queue which are scheduled to be printed to the printer(s) as the printer(s) become(s) available. The Spooler can be directed to output report items to printer(s), tape, or to a specified file in "runoff" format.

locks, Spooler

four locks specifically associated with the spooler: the "master", "input", "form" and "permanent element (or entry)".

If a lock is not set, the literal "###" appears in the location.

If a lock is set, the port.number of the process holding the lock is displayed by the "list-locks" command.

"mq" is the "master queue" lock. It is locked by the Spooler process when altering any of its tables, such as when a printer is being started.

"iq" is the "input queue" lock. It indicates the maximum number of jobs that may be open at once. The limit is about 650 "permanent" (hold-type) entries.

"fq" is the "form queue" lock. It is locked whenever a form queue is attached to a printer.

"peq" is the "printer element queue" lock. It is locked briefly by each user as the first byte of a Spooler entry is produced.

options: Spooler

can be specified to most spooler commands. Some have numeric arguments.

To keep these options clear for the options interpreter, it is recommended that numeric options be separated with blanks as in the following example:

sp-assign 3 hs f1 ?

This sets parameters for hold file output, suppresses print out at the end-of-job, routes output to form queue 1, and 3 copies.

The spaces in this option string are optional, in this case, since there are no instances of "back to back" numeric parameters. Options may appear in any order. Consider the case:

sp-assign f1 3 hs ?

Without the space between the "1" and the "3", one copy of the output is directed to form queue 13, rather than 1.

spooler

account holding a file called "Peqs" which is used to reference spooler jobs as normal D3 items.

The data section of this file is a "QS" pointer to the "peqs:" driver. The "QS" signals the save processor to dump both the spooler header information and the spooler jobs to tape. In general, this is the only account that should contain a "QS" pointer to the peqs: driver. All other accounts should contain simple "Q" pointers.

At the present time, the spooler account does not serve any purpose beyond this. The "QS" super-q-pointer was not put into dm because confusion could result if the peqs file was restored accidently.

It is now possible to restore spooler jobs from previous file-saves. Two options are available; individual items or the entire file can be restored to a temporary file, then copied to the spooler as needed, or the entire contents of the previous spooler can be overlaid on the current spooler. Note that the save must be done with the "E" option to save spooler jobs.

:startspooler

see "startspooler".

assignfq

assigns a printer device driver to a Spooler formqueue.

This printer driver is then used by the Output processor (OP), or by FlashBASIC "@" functions to generate special formatting commands like boldface, cursor positioning, or underlining, when output is directed to the spooler.

Using a "?" as a parameter produces brief on-line "help".

Multiple formqueue's can be updated in the command, but each queue must have an associated device name.

Printer drivers are located in the "dm,devices," file. The current driver assignments may be displayed with the "listabs" command.

A printer may have more than one formqueue attached to it. (See the "startptr" command).

On Unix-based implementations, the spooler has the ability to transfer data directly from the D3 process to a Unix-based operation such as a spooler, file, communications link, hardware device, etc. The relationship between D3 and the Unix process is established by the "assignfq" command. Simply place the Unix command string in parentheses immediately after the "device.name" field. Assignfq recognizes the parentheses to mean a Unix process and sets up the proper linkages. (See the "startshp command.)

Syntax

assignfq formqueue,device.name{(Unix.command)}{,
formqueue,device.name{(Unix.command)}{,...}} {(option}

assignfq ?

Options

c "Compiles" devices file item into cursor control block (ccb).

The Cursor Control Block is a binary item, (sort of like a system-level dynamic array) which contains the codes and control strings pertinent to the device in question. Each system functions (@(-(n))) has its own array position, and the system cursor function searches this item for control strings such as clear-screen, clear to end of screen, etc.

Example

```
assignfq 3,hp-lzrii
This assigns driver "hp-lzrii" to formqueue 3.
assignfq 3,b(cat>/file1)
sp-assign f3?
Line# Status Copies Form# Device
    7  p          1     3  b(cat>/file1)
list only md (p
The above assignfq command establishes a relationship between the D3 spooler
form queue 3 and the Unix process "cat > /file1".
Then, the sp-assign command tells the system to attach to form queue 3. Note
that the sp-assign is necessary even if the line is already assigned to form
queue 3 so that the process is informed that form queue 3 should now output to
a Unix command. Finally, the list command will create the Unix file "/file1"
with an image of the listing.
The following FlashBASIC program creates a Unix file containing the text "line
1".
execute "assignfq 0,ibm3151(cat>/file1)"
execute "sp-assign f0"
execute "!rm /file1"; * Clear old one out
printer on; * Start printer output
print "line 1"
execute "!ls -l /file1"
printer close; * Close output
execute "!ls -l /file1"
execute "!cat /file1"
The output of the program is:
Assigned form queue device 0, IBM3151(cat>/file1)
-rw-rw---- 1 pick  pick  0 Apr 23 11:15 /file1
-rw-rw---- 1 pick  pick  7 Apr 23 11:15 /file1
line 1
Special attention must be paid to the size of the Unix file. Note the size
printed on the Unix "ls" command (0 and then 7). Even after the "line 1" text
has been printed, the Unix file size is still zero.
Only after the print file has been closed does the output appear in that file
(as shown by the new size of 7). Because of this buffering effect, printer
jobs must be closed before examining the Unix output.
```

list-ptr

see "listptr".

Syntax

see "listptr"

listabs

displays the current Spooler assignment parameters for each process on the system.

If a port.number, or range of port.numbers is specified, the report is limited to the given port(s).

The report has the following format:

line# status copies form# device

"line#" port.number of each process. (see "who" or "listu")

"status" current output options (see "sp-assign").

"copies" current number of copies (see "sp-assign").

"form#" current form queue (see "sp-assign").

"device" printer device name attached to the form queue (see "assignfq").

Syntax

listabs {port.number{-port.number}} {options}

Options

* see "options: TCL".

Example

```
listabs (p
This directs the output of "listabs" to the spooler.
listabs 5
5    p           1     11  hp-lzrii
listabs 3-12
Line#  Status   Copies  Form#  Device
3    p           1      3  hp-lzrii
4    p           1      0  dp.lzr
5    p           1     11  hp-lzrii
6   hs           1     11  hp-lzrii
7    p           1     11  hp-lzrii
8    p           1      0  dp.lzr
9    p           1      7  hp-lzrii
```

listpeqs

displays the status of all Spooler controlled print job elements.

Outputs a list of spooled entries. The following column headings appear in the report:

"job#" is the element (job) number assigned by Spooler.

"stat" is the "encoded" print file status (see "status").

"lnk". In a linked form queue, this is the job# of the next output job after this one.

"line" is the port.number that generated this queue element.

"status" is the assignment parameters on the print file:

a: available; may be edited, spooled or deleted.

c: closed; "spooling", job is ready to de-spool to device.

g: align; phantom "alignment" print job (see startptr)

h: hold file; see sp-assign.

i: immediate output; see sp-assign.

l: locked; see sp-edit, "l" option.

n: no close; see sp-open or sp-assign "o" option.

o: output is currently being generated by the user.

p: assigned to printer output.

r: re-queued; has been edited/spooled with sp-edit.

s: spooled; linked to an output queue.

t: assigned to tape output.

x: aborted; has been aborted by sp-kill.

"copies" is the number of copies remaining to print.

"form" is the assigned output form queue.

"frames" is the number of frames in use by the print file. If the word "open" appears here, the printfile is still being spooled. If the print file was created with the "c" ("choke") option, the number of frames used reflects the number of frames left to print when the print file was closed, instead of the actual number of frames used.

"date" is the date the print file was added to the job queue.

"time" is the time the print file was added to the job queue.

"user" is the user-id of the user who created the print file.

"Frms" is the size of the print file. A trailing "+" indicates an increasing file, a trailing "-" indicates a decreasing file. A blank entry indicates a file build is being done, or no data has been spooled out. See option "e".

"CurFID" is the current frame-id being output (in hexadecimal). See option "e".

"curpos" is the current position of the despooling process. See option "e".

"begfid" is the beginning frame-id of the start of the print file. See option "e"

Syntax

listpeqs {options}

lq

Options

'account.name' This displays the Spooler entries created by the specified account name. Single quotes are required around the account name.

'user-id' In D3, Spooler entries are stored under the user-id of the user who generated them. This displays the Spooler entries created by the specified user-id. Single quotes are required around the user-id.

n{-m} (integer number(s)). Displays element "n", or elements "n" through "m". The current maximum is about 600.

a Outputs the status of all reports created by the current account.

c Outputs only the count of spooled entries and total storage (frames) used.

e Outputs print file absolute (disk) location. When this is in effect, the display has two additonal columns of output: "curpos" indicates the current frame and displacement into that frame being printed. "begfid" is the first frame of the printfile. Both frame references are indicated as hexadecimal addresses. Under the "frames" column next to this printfile, if a "+" character follows the number, it is still being Spooled; if a "-" follows, it is despooling (printing).

f{form.queue.number}

Outputs status of jobs queued for output in form queue order (0-125). The optional "form.queue.number" limits display to the specified form queue.

h Outputs hold files only.

l Displays status of elements which have been deleted, as well as those which are active.

p Directs output to system printer, via the Spooler.

Example

```
listpeqs 'mlb'
listpeqs (e)
listpeqs 'jack' (n
listpeqs 13-22
```

listptr

displays the printer control block status for all printers.

The report has the following format:

Page Dev or

Type Number Output Queues Skip Line # Status

"printer type" is either serial or parallel.

"number" is the printer number. See "startptr".

"output queues" are the queues assigned to this printer. (3 maximum)

"page skip" is the number of pages to skip between print files.

"dev or line#". If the device is a serial printer, this displays the port.number. If the device is a parallel printer, it displays the printer number.

"status" is the current printer status: "active" means the printer is printing, initiating or terminating a report. "inactive" means the printer is inactive. If it is also stopped, the "startptr" verb may be used. "stopped" means the printer is set to stop. "Unallocated" means that the printer has never been started, or has been deleted by an "sp-kill (d)" command, or has been lost due to a control block error. The printer may be started using the "startptr" command.

The "list-ptr" verb is a synonym for "listptr".

Syntax

listptr {options}

Options

printer.number{-printer.number} Specifies beginning printer number, or beginning through ending printer numbers.

b Lists both allocated and unallocated printers. An unallocated printer is a printer "gpiocb" (general purpose input/output control block) with no printer assigned to it. (An empty table entry).

n Activates nopage function on output to terminal.

p Directs output to system printer, via the Spooler.

Example

```
listptr
Page  Dev or
Type Number Output Q's Skip Line Status
serial   0   0          1    13  inactive
```

```
serial   1   3   4   5   1    11  inactive
serial   2  11  12       1    15  inactive
serial   4   6   7       1    14  inactive
serial   7  13           1    99  active
serial   8   1  98       0    74  inactive
serial   9  99           0    19  stopped

listptr 2-5b
Page  Dev or
Type Number Output Q's  Skip Line Status
serial   2  11  12       1    15  inactive
serial   3   0   0   0   0     0  unallocated
serial   4   6   7       1    14  inactive
serial   5   0   0   0   0     0  unallocated
```

lq

see "listpeqs".

peqs

a super-Q-pointer used to access spooler jobs. The "peqs" super-Q-pointer allows access to spooler jobs as if they were standard files. The following translations are applied on both retrieval and update (note that a DLE is character 16):

Raw Job	Simulated File Item
CR,LF	AM
DLE	DLE,DLE
SM	DLE,'_'
AM	DLE,'^'
VM	DLE,']'

Note also that all trailing null characters are removed when reading a job.

This translation makes it easy to read, modify, and update jobs using BASIC, ED, or the Update processor.

The "COPY" verb can be used to copy jobs into standard files for archival purposes.

The "DELETE" and "CLEARFILE" operations are supported, but only affect unlocked hold files (as with the associated SP-EDIT operation).

An update of an existing, unlocked, hold job updates the job contents without changing the job number. An update of a non-existent job creates a new spooler job with the current SP-ASSIGN options. An update of a locked or non-hold job fails and does nothing.

When reading an item for later update, it is suggested to use a READU. This operation will fail if the job is locked or not a hold file.

If the user updates the "peqs" item "0", then the job is re-spooled with the current sp-assign options to the first available entry number. The user may optionally override the current sp-assign parameters by directly following the "0" in the item id by the new options. Note that there can be no spaces in the item id and that the sp-assign options "c","i","o","rnnn","t" and "?" are ignored.

When using the Update processor to edit "peqs" items, it is suggested to use the "(r" option to better handle control characters and to prevent truncation of trailing attribute marks.

The following pseudo-attributes are defined off of the peqs file. These attributes actually translate from the peqs item-id into the header section of the peqs file (called "hdr:peqs"):

Job The job number

Size The size of the spooler entry

User The user who last modified the job

Pib The PIB of the user who last modified the job

Line The PIB of the user who last modified the job

Date The date of last modification

Time The time of last modification

Permissions Indicates ability to read or write a job

Form The form queue number

Copies The number of copies set

Status The job status information (similar to list-peqs)

Prio Indicates printing priority within a form queue

The "SPOOLER" account holds a super-Q-pointer with the file definition attribute set to "qs". This signals the save processor to dump the both the spooler header information and the spooler job itself to tape. For more information see the spooler.account entry.

Example

```
:list peqs
Page 1    Peqs           10:57:12 12 Sep 1994
peqs
1
2
3
[405] 3 items listed out of 3 items.

:u peqs 3 (r
peqs '3' size = 167
01 .:list md md (p
02 Page 1 md           10:57:59 12 Sep 1994
03
04 md..................
05
06 md
07
08 [405] 1 items listed out of 1 items.
09
peqs '3' size = 167 filed
```

pr-spool-job

re-spools a spooler hold job. The user can select a one-time sp-assignment for this job by placing the desired sp-assign options after the open parenthesis at tcl. Some sp-assign options are not valid in this context, they are "c", "i", "o", "rnnn", "t" and "?". See filename.peqs for more information.

Syntax

pr-spool-job spooler.job {(assignment.options}

Options

See tcl.sp-assign for options with the caveats described above.

Example

```
pr-spool-job 3 (f2,4
This will re-spool spooler job number 3 to form queue number two. Four copies
of the job will be output.  Note that a comma is used to seperate the form
queue number "2" from the number of copies "4". Spaces cannot be used.
```

sp-assign

displays, changes or resets the current Spooler output assignment options for the current process/user. An "sp-assign" command with no options resets the assignment to the "default" state.

The default assignment is: output to printer, "normal" spooler entry (meaning, not a hold file), one copy, form queue 0, close files at end of process, queue job for output immediately upon closing.

"sp-assign" with no options also closes any currently open files assigned by the current process.

Syntax

sp-assign {options}

Options

number.copies (integer number in the range 1-125) indicating the number of copies. (default = one).

? Outputs resulting setting, or current setting when entered alone. The display shows the process port.number, current assignment parameters, and the device assigned to the currently assigned output queue. (See "assignfq").

a Ouput goes to the alternate port on the current terminal. To function correctly, the auxiliary port on and off commands must be defined for the current terminal in the devices file. Note that the a option does not suppress output to the system printer by default. To disable system printer output, use the "s" option as well.

c "Chokes" output. This stops the process from generating output when it gets 20 frames ahead of the printer. The "c" option is relevant only with output going to the printer and can only be used with an "i" option and without an "h" option. If a printer is not available or assigned to another output queue, the "c" option causes the process to wait until the printer is available. The "choke" option causes the copy count to go to 1 (one).

d Delays sending output to the queue. The output is delayed until the file is closed. This option must not be used with the "i" option. This option overrides the "i" option.

fnumber Indicates the output queue into which a print file is to be inserted. "number" must be between 0 and 125, inclusive. The default output queue is 0. Each output queue is assigned to a specific printer. See "listptr" and "sp-status.

h Spools output to a hold file.

i Spools output to the printer immediately.

m Suppresses the display of "[1151] Entry # n".

o Leaves the entry "open". The print file remains "open" upon completion of the process. This allows placing multiple print jobs in one physical Spooler entry. See also the "sp-open" and "sp-close" command. To close the entry, use the "sp-assign" command without options, or the "sp-close" command.

p Assigns output to the printer. This is the default.

rnumber Opens a print file with the specifications given by the other options in the string. "number" can be between 0 and 125 inclusive and is the number used in "print on 'n'" statements in FlashBASIC. CAUTION: If the "r" option is used to send two or more print files to tape during the same process, their outputs will be intermingled on the tape. If the "r" option and the "i" option are both specified, the printer starts to generate output immediately and does not wait for the print job to be completed before printing. There is no harm in this, except that no other print jobs can be serviced by that printer until the program has completed. "sp-assign" with the "r" option can be used to initialize print files until the capacity of the input control block is exhausted. There are about 60 input control block records available, depending on the system configuration.

s Suppresses output to printer.

t Assigns output to tape. "t" specifies that the print file goes directly to tape under the control of the generating process rather than under the control of the Spooler process. Therefore, the tape should be mounted before the job is started. The "t" option invokes the "t-att" verb, if necessary.

Example

```
sp-assign hs ?
Line#  Status   Copies  Form#  Device
39   hs           1      0  dp.lzr
"Line#" is the port.number to which this terminal is attached.
"Status" is the assignment options set by "sp-assign".
"Copies" is the number of copies to print
"Form#" is the form queue number(s).
"Device" is the  Printer device name. (See: "assignfq").
sp-assign 13 f11 ?
Line#  Status   Copies  Form#  Device
39    p          13     11  hp-lzrii
```

sp-close

closes all current Spooler entries previously opened by the current line with an "sp-open" command or an "o" option in conjunction with an "sp-assign" command.

Example

```
sp-close
[1140] your open files were closed.
```

sp-edit

accesses existing Spooler entries for processing or deletion. When options are omitted, all available (un-spooled) entries generated from the current user-id are retrieved for processing. Only files in the "hold" status are available. (See "sp-assign" and "sp-kill"). The first 500 bytes of the print job are available for display, but the job may not be accessed in any other manner.

The hold files may be left in the queue, output to tape, printed, deleted, or copied to data file items. (Hold files are generated by using the "h" option in "sp-assign" to send all subsequent printer output to a hold file, or by using "sp-kill" to change a currently spooled file to a hold file.)

With no options in effect, "sp-edit" returns all available hold files generated by the user-id which invoked the verb.

After the "sp-edit" is invoked, the following options are available depending upon the options previously selected:

display? (y/n/s/d/x/(cr))? -

y Displays the first 500 bytes. If the job is being created or output, and the "l" option is in effect, the prompt "another ?" displays. The only proper response to this prompt is "y" to re-display the first 500 bytes again. If the print file is not being created or output, the display will be followed by the prompt, "string-".

n The process skips to the "string" prompt.

s The process skips to the "spool" prompt.

d The process skips to the "delete" prompt.

x Terminates "sp-edit" and returns to TCL.

<enter>

Skips to the next print file.

Any other response skips to the "string" prompt.

string:

string<enter>

Scans the print file for the first occurrence of the given string, and begins output at the beginning of the line containing the string. If the string is not found, a message prints indicating this, and the operator is prompted again.

<enter>

Skips to the "spool" prompt.

spool:

y Enqueues the print job for output to printer or tape as specified by the "sp-assign" option in effect when the job was created. (See "r" option below). After the file is enqueued, "sp-edit" attempts to process the next hold file, if any. If neither the "sp-assign" nor the "sp-edit" specifies printer or tape as the output destination, a message displays.

n (or <return>) Skips to the "delete" prompt.

t Directs output to the terminal, then returns to the "spool" prompt. At the end of each page of terminal display, press any key to view the next page or <ctrl>+x to stop spooling to the terminal and return to TCL.

tn Directs output to the terminal without pausing at the end of each page, then returns to the "spool" prompt.

f Copies to data file item(s) and prompts for additional information. When used with the "c" option on the TCL line, this option suppresses the embedded (OP/Runoff) ".CHAIN" commands and multiple item-ids for a print job, creating one item only.

The OP (and Runoff) commands ".bp" and ".nf" are inserted at the beginning of the data file item before the hold file is copied. Any trailing blank lines in the hold file are not copied to the data item. To have trailing blank lines copied, use the "v" option when initiating the "sp-edit" verb. It is necessary to be logged onto the user-id which created the hold file in order to copy it to a data file. After the item is written, the delete prompt is displayed. "y" deletes the entry.

n (or <enter>, or any other response) Skips to the next print file or TCL, if no more print files exist.

Spooling Hold Files to Tape

If the process is to copy the hold file to tape, the tape is checked to determine if it is attached. If the tape is not attached and it is available, the tape is attached using the system default block size (16384 for SCT; 500 for diskette; 8192 for half-inch tape). The following message displays:

"tape attached block size: nn"

If the tape was already attached, the same message is output, except the block size used is the size set at "t-att" time.

If the tape is attached to some other line, the following message displays:

"tape attached to line nn"

If the "w" option is in effect, the process waits until the tape becomes available.

A tape label is written on the tape which includes the word 'spooler', and the account.name. To suppress the label, include the "h" option (or "hdr-supp" modifier) with the "sp-edit" verb. The tape label can also be suppressed by including the "hdr-supp" modifier (or "h" option) with the verb that originally creates the hold file.

Syntax

sp-edit {options}

Options

'account.name' This accesses the Spooler entries created by the specified account name. Single quotes are required around the account name. "sys2" privileges are required to use this option. The account name option overrides the "u" option.

'user-id' In D3, Spooler entries are stored under the user-id of the user who generated them. This accesses the Spooler entries created by the specified user-id. Single quotes are required around the user-id.

file.number{-file.number} Designates print file, or a range of print files, if available. Must be within the range of 1 through 600, inclusive. When a range of print files is specified, the second number must be higher than the first.

b Suppresses the ".bp" and ".nf" (Runoff) commands in each item when spooled to a file with the "f" option. See "warnings".

c Places the entire print job into one item when spooled to a file, rather than breaking each page of the report into a separate item and inserting (Runoff) "chain" statements at the end of each item to chain each item to the next item.

d Deletes print file(s), when used in conjunction with a print file number or a range of print file numbers.

fnumber{-number} (integer number(s)) Edits specific hold file, or a range of hold files. Must be in the range of 0 through 125.

h Suppresses the tape label if the hold file destination is tape. The option is effective with either the "sp-assign t" specification or the "sp-edit t" specification.

k Keep raw characters. This option removes the conversion of non-displayable characters to periods.

l Displays an enqueued print file.

m Suppresses remaining prompts when used with the "d" or "s" options.

n No pause. Suppresses the pause at the end of each page on the terminal.

o Looks at the print file that is being output. This must be used with "l" option.

p Sends the print file to its assigned printer. This option overrides the current "sp-assign" parameter(s).

r Uses the current "sp-assign" parameters for the form number and copy count when an entry is spooled. The form number and copy count are changed permanently.

s Spools the print file(s).

t{w} Directs the hold file to tape. This supercedes a "p" option. "w" waits for tape to be readied, if it is not already.

u Edits all hold files. This option requires sys2 privileges.

v Converts a hold file to a data file. It does not remove any trailing blank lines at the bottom of a page.

Example

```
sp-edit mso
Obtains all print files created by the current user-id, and views (observes)
them.
sp-edit msp1-5
Obtains all print files created on the current user-id, and sends jobs 1
through 5 to the printer.
sp-edit nd1-10
Obtains print jobs 1 through 10, and deletes them.
sp-edit 10-20
Obtains available hold files with entry numbers 10 through 20 generated by the
current user-id.
sp-edit ru3
Edits entry# 3 (generated by any user-id), and uses the current sp-assign
output queue and number of copies settings.
sp-edit t7-11
Sends to tape each print file with entry #'s 7-11 generated by the current
user-id.
sp-edit 4lu
Displays the first 500 bytes of print entry #4 (created by any user-id, if the
user has "sys2" privileges), and the print file is not being output.
sp-edit f3 ms
Obtains all print files created on the current user-id for output queue 3, and
spools them.
sp-edit 'pr' f4 p ms
If the current user-id has sys2 privileges, this spools all available print
files created by the "pr" user-id in output queue 4 to the printer, and ends
sp-edit.
sp-edit md
Deletes all available files created by this user-id.
sp-edit mud
Unconditionally deletes all print files.
```

sp-kill

stops spooler entry output or removes printers.

More specifically, it provides the following functions:

1) To abort the currently active print job(s) immediately.

2) To dequeue print file(s) on the optionally specified printer.

3) To remove printers from the Spooler system.

If no options are specified, "sp-kill" aborts the current print job on printer 0 (zero). Unless the user-id has sys2 privileges, only print files generated by the current user-id invoking the "sp-kill" can be aborted. "sys2" privileges allow termination of any print file.

Syntax

sp-kill {options}

Options

printer.number{-printer.number} Aborts jobs on a given printer number, or a range of printer numbers, between 0 and the maximum printer number.

a Stops Spooler entry or entries created by the current user-id.

b All entries. When used with the "d" option, this kills all printers. When used with the "f" option, this kills all open jobs. When used without any other option, this aborts all active printers.

c Kills all copies of the spooler entry or entries, rather than just the current copy.

dprinter.number{-printer.number} Deletes given printer(s) from the system. This requires a "sys2" privilege level. When used with the "w" option, it waits until the current job completes, rather than aborting it immediately.

f{file.number{-file.number}} Returns Spooler entries to available hold file status. The beginning and ending file numbers must be in the range (0-600).

n Suppresses abort message on output when directed to a printer. Also see "s" option.

o Used with the "f" option, this unlinks print file(s) already being output.

s Suppresses system messages. Also see "n" option.

u Used with the "d" option, this deletes printer 0 (zero) only.

w Waits until all processes have finished. Use of this option with "sp-kill" is identical to using the "stopptr dn", which causes printer "n" to die after the current job. Note that "sp-kill" and "stopptr" are not exactly synonyms, and "sp-kill" cannot be removed, as it has several other features which "stopptr" does not perform. These are two totally different commands which merely have overlapping options.

Example

```
sp-kill 1
Kills the print job on printer 1.
sp-kill of7
Kills print file 7, even if it is being output, and returns it to an
available, hold file status.
sp-kill
Kills the current print file on printer 0.
sp-kill a
Kills all print files currently being output, that were generated by the
current user-id.
sp-kill a2
Kills the print file on printer 2.
sp-kill 3-5
kills the print file(s) going to printers 3, 4, and 5.
sp-kill b
Kills all print files going to all printers.
sp-kill f5-10
Changes print files 5 through 10 to hold files.
sp-kill bf
Changes all print files going to all printers into hold files.
```

```
sp-kill d1
Deletes printer 1 from the system.
sp-kill d3-5
Deletes printers 3, 4, and 5 from the system.
sp-kill bd
Deletes all printers from the system.
```

sp-open

leaves the next Spooler entry generated by the current process "open" upon completion of the output generating process. Any subsequent (printer) output will be attached to the end of the "open" entry, until an "sp-close" command or an "sp-assign" command is issued.

Example

```
sp-open
block-print "this is " (p
entry # 133
block-print "a test" (p
(no entry number appears -- this is "attached" to "open" entry)
sp-close
[1140] Your open files were closed.
```

sp-status

displays the status of all Spooler-controlled devices.

If no options are specified, the status of the Spooler and all allocated printers is displayed.

The printer status may be one of the following:

"active" means the printer is currently printing a report.

"inactive" means the printer is currently idle.

"stopped" means the printer has been stopped with a "stopptr" command, or is set to stop once it becomes idle.

"unallocated" means the printer has never been started, or has been deleted from the spooler with an "sp-kill" command using the delete option. The printer may be restarted with the "startptr" command.

Syntax

sp-status {options}

Options

printer.number{-printer.number} Designates a specific printer, or range of printers, to display. Must be between 0 and the maximum printer number, inclusive.

b Outputs status of all printers, allocated and unallocated.

n No pause; suppresses pause at end of page on terminal display.

p Directs output to system printer, via the Spooler.

Example

```
:sp-status
The spooler is inactive.
Printer #  0 is serial, inactive, and on line.
The printer is running on line  13.
Assigned output queues: 0 .
The number of inter-job pages to eject is 1.
```

sp-tapeout

retrieves (inputs) reports previously spooled to magnetic media and directs output to the Spooler.

The Spooler redirects the file according to the options currently specified in "sp-assign".

Syntax

sp-tapeout {options}

Options

a Converts EBCDIC format to ASCII format.

l Treats each tape record as a line, if the tape record length is less than 140. This assumes the record is right-padded with blanks, and without carriage-return, line-feed sequences embedded in them. Trailing right blanks are removed and a carriage-return, line-feed sequence is inserted at the end of each tape record.

u Converts all lower case characters to upper case.

Example

```
sp-tapeout
Spools the file (at the current location of the tape) to the
destination specified by the current sp-assign command.
sp-tapeout ua
Reads the file (at the current location of the tape), converts the data from
EBCDIC to ASCII, converts all characters to uppercase, and spools the file to
the destination specified by the user's "sp-assign".
```

startptr

activates and initializes printers.

Up to fifty printers, although only up to four parallel printers are supported on PC systems. The actual number of printers supported depends on the hardware configuration.

If the status of the printer is active, "startptr" has no effect.

The following parameters can be specified:

"number" is the printer number between 0 and the maximum printer number, which is determined by the actual number of physical ports on the system, plus 4. If no other options are specified, this restarts a "stopped" printer. (See "stopptr").

"queue1, queue2, queue3" are output queue numbers between 0 and 125, inclusive. "queue1" is given first priority for spooling, "queue2" is given second priority, etc. At least one form queue must be specified, but not more than three.

"page.eject" is the number of pages to eject at the end of each print file, and must be between 0 and 9, inclusive. Also see "s" option.

"type/address" is the printer type and port number. The printer type may be "p" for parallel, or "s" for serial printers. The "address" is determined by the type. For serial printers the address references the port number. On a parallel printer, the address is the address of the physical device. In the case of a parallel printer, the printer number and the physical device number should be the same. The system uses the printer number as the device number, but requires the device number for syntax reasons. On D^3 Unix implementations, all printers are considered "serial" because of the unique relationship between Unix and D^3.

"a" initiates the alignment process. The system will prompt for the number of lines, and then print a sample alignment. After each trial, it prompts again for either a "y" to try the alignment

again, "t" to terminate the alignment process, or "n" to input a different number of alignment lines. During the alignment process the printer is not yet considered attached to the spooler, so it cannot yet receive print jobs.

Syntax

startptr number{,queue, page.eject, type/address,{a} {(options)}}

startptr number, (queue1{, queue2{, queue3}}), page.eject, type/address, {a} {(options)}

Options

b The serial printer is assigned the line feed and form feed delays of the initiator.

d Enables debugger on the current port. Used primarily by developers to test spooler code.

s Inhibits the initial page-eject command at the start of a print file. Also see the "page.eject" parameter.

x{n} Indicates that the serial printer does not recognize an ASCII form feed character as a page-eject command. The printing process must count lines within the page and send the correct number of blank lines when a page-eject command occurs. "n" indicates the number of lines per page. If the "x" option is specified and there is no numeric specification, the page length defaults to 66 lines.

Example

```
startptr 0,0,0,p0
This starts printer 0 on form queue 0, skipping 0 pages between jobs, on
parallel port address 0.
startptr 1,1,0,s12
This starts printer 1 on form queue 1, skipping 0 pages between jobs, on
serial port 12.
startptr 9,(98,99),1,s19
This starts printer 1 on form queues 98 and 99, skipping 1 page between jobs,
on serial port 19.
startptr 1,(3,4,5),1,s11
This starts printer 1 on form queues 3, 4 and 5, skipping 1 page between jobs,
on serial port 11.
```

startshp

starts a shared printer under Unix. The output of the D^3 printer process is routed to the Unix spooler.

This command combines the "startptr" command and the necessary "assignfq".

Keeping track of shared printers can be achieved by the "shp-status" TCL command. Terminating a shared printer should be done by the "shp-kill" TCL command.

Once a job has been submitted to Unix, the (Unix) "lpstat" command may be used to keep track of the running jobs. The printer devices associated with the form queues must have a special interjob sequence so that the Unix spooler can distinguish the end of the job.

bhe following parameters can be specified:

"ptr.number"

The (integer) printer number.

"q1", "q2", and "q3"

The output queue numbers.

"eject.pages"

An integer number which specifies the number of pages to eject at the end of each job.

"type/address"

Printer type and port number. The printer type can be "S" or "P." All printers are treated internally as serial devices, even when (as in the case of a shared printer) the 'device' is a pseudo device.

"pd1", "pd2", and "pd3"

These indicate printer devices. To each form queue, a printer device, defined in "dm,devices," must be associated to define the printer characteristics as well as the interjob sequence. The interjob sequence is defined in the function "@(-269)". If the number of printer devices is less than the number of form queues defined, the last device is used for the remaining form queue(s).

lp This is a Unix command used to spool the D3 print job. If not specified, lp is used by default. If the Unix command has more than one word (separated by spaces), the command must be enclosed in parentheses.

When the printer is successfully started, a Unix log file is created in the directory "/tmp/lppick" which is used to keep track of the shared printer activity. The name of the file in this directory is formated as follows:

vmn _ prt _ port

Where:

vmn : Virtual machine name,

prt : D3 printer number,

port : Port number.

The log file contains the following elements, separated by attribute marks:

pid ^ cmd ^ eoj ^

Where:

pid : PID of the lppick process,

cmd : Unix command used to send data to the spooler,

eoj : End of Job string.

Optionally, this file also contains trace information, if the (T{n}) option is used or if an error is encountered. The level of trace is coded as follows:

1 (default) Record only beginning and end of jobs.

2 Record, in addition to level 1, each write to the lp command.

3 Record, in addition to level 2, all data which is exchanged. Non printable characters are replaced with a '.'.

4 Record, in addition to level 2, all data which is exchanged. All characters are replaced by their hexadecimal value coded on two ASCII characters.

The trace is stopped only when the printer is stopped with the 'shp-kill' TCL command.

The log file is used by the "shp-status" TCL command to keep track of printers shared with Unix and the processes involved, and by the "shp-kill" TCL command to find the processes to terminate.

The "startshp" command supports all "startptr" options, plus the one listed below.

Syntax

startshp ptr.number,(q1{,q2{,q3}}), eject.pages,type/address, (pd1{,pd2{,pd3}}) {,(lp)} {(options}

Options

a Initiates alignment process. Note this is an option, while on the startptr verb, it is an argument. When this option is used, each alignment attempt is a Unix job. The Unix printer should have been set up so that there is no banner, no interjob page eject.

c Compiles the printer devices. This option must be used whenever the printer definition item is changed.

t{n} Trace shared printer activity. Even when the trace is not activated, errors are recorded. 'n' is a trace level. See above text for a description of each level.

v Verbose. Various information is displayed on the terminal. This option should be used only to diagnose problems. It displays the amount of data that is transferred to the Unix spooler.

Example

```
startshp 3,(2,3),1,s32,lp.unix (s
Starts printer 3, serving form queues 2 and 3, both associated to the same
printer device "lp.unix", with one page eject after each job (required for a
laser printer), on port 32.
startshp 4,7,1,s33,lp.unix,lpr (s
Starts printer 4, serving form queue 7, associated to the printer device
"lp.unix", with one page eject after each job (required for a laser printer),
on port 33, using the Unix command lpr to access a remote printer.
startshp 5,(8,9),1,s34,(lp.unix,lp.lzr.ux),(lp -n 2 -o nobanner) (s
Starts printer 5, serving form queues 8 and 9, respectively
associated to the printer devices "lp.unix" and lp.lzr.unix", with one page
eject after each job (required for a laser printer), on port 34, using the
command "lp -n 2 -o nobanner" to generate 2 copies of each job and to remove
the banner.
startshp 6,10,0,s35,lp.unix,(cat >> /tmp/print) (s
Starts printer 6, serving form queue 10, associated to the printer device
lp.unix, with no page eject after each job, on port 35, using the command "cat
>> /tmp/print" to send all print jobs into a regular Unix file.
startshp 7,99,0,s16,lp.unix (a
Starts printer 7, serving form queue 99, associated to the printer device
"lp.unix", initiating the alignment process. Jobs must have been queued on the
specified form queue for alignment to be attempted.
startshp 8,97,0,s17,lp.unix (t2
Starts printer 8, serving form queue 97, associated to the printer device
"lp.unix" with a level 2 trace.
```

startspooler

initializes the Spooler sub-system and returns it to normal operation.

This process is usually invoked when the system is powered on (see colstart, system-coldstart or user-coldstart)

The Spooler controls the printing process and is normally started during coldstart. When necessary, "startspooler" can be used to execute specified portions of the Spooler initialization process without reinitializing the entire spooler sub-system.

Using "startspooler" without options makes no changes in the spooler sub-system, but simply "wakes up" the printers (if any are started). The printers then poll their respective form queues for scheduled jobs to print.

Syntax

startspooler {port.number} {(option{s})}

Options

port.number Starts Spooler process on given port.number. If "port.number" is not specified, the Spooler is started on the first port following the last physical port on the system. Used primarily by developers to test Spooler code.

c Starts Spooler, clears the form queue assignments, kills the printer, clears input and output queues and converts any existing print files to hold files. This should be used as the "second to last" option for resetting the Spooler.

d Used in conjunction with the "port.number" option above to enable the debugger on the port.

i Starts the Spooler and initializes control data with new overflow frames. The disk space used previously by the Spooler is lost until the next file restore (approximately 5-9 frames). This is the last resort in resetting the Spooler.

stopptr

stops the specified printer at the end of the current print job.

If the printer is inactive, it stops immediately. If it is active, it stops at the end of the current print file. If the current print file is specified to print multiple copies, the printer stops after the completion of the current copy.

"sys2" privileges are required to use "stopptr".

Executing this command without specifying a printer attempts to stop printer 0.

Syntax

stopptr {options}

Options

printer.number{-printer.number} Designates the printer number, or range of printer numbers to stop. Must be between 0 and the maximum printer number. The maximum printer number is the actual number of ports on the system, plus 4.

b Stops all printers.

s Suppresses display of error messages.

w Waits until the printer has completed printing any print file and is inactive. This option can be used, for example, when the "stopptr" verb is issued from a macro, followed by a "startptr" verb. This ensures that the printer is stopped before "startptr" is executed.

Example

```
stopptr 1
printer # 1 set to stop
stopptr b
printer # 0 set to stop but is still active
printer # 1 set to stop
stopptr 1w
Sets printer #1 to stop, then waits for printer #1 to become inactive before
returning to TCL.
```

System Debugger

used for D3 assembly code debugging, has the capability of accessing and changing data directly on the disk.

A thorough familiarization with the virtual assembler is recommended prior to using the debugger, as it can be very destructive.

Symbol file definitions:

"gsym" The permanent symbol table.

"psym" A q-pointer to "gsym".

"osym" The object symbol table.

"tsym" The temporary symbol table.

Note that these are the "standard" tables for "classic" Pick. These files are not always provided with each release of D3 and may actually go by different names. Usually, however there is a "psym" file provided with the assembler account, which may have to be obtained separately.

The "set-sym" command sets the symbol file pointer for use by the system debugger:

set-sym gsym (t

Constructing address specifications:

To address any disk location, three pieces of information are usually required:

The "data format specification".

The "data reference specification".

The "data window specification".

The debugger prompt is a "!". If the user types break in the debugger, the debugger debugger is entered. In this case, the "d!" prompt is displayed. In this case, type "g" to continue.

data format specification

specifies the output display format mode of operation to use.

It precedes the address of the data to access. Once set, the "data.format.specification" remains in the designated mode until explicitly changed and is applied to subsequent data displays which do not specify a different data format.

The "data format specification" is one of the three elements used in referencing disk locations within the system debugger; it specifies the output format: either "integer", "character" (ASCII) or hexadecimal.

The data format specification is optional. If unspecified, the previous format specification is used for the current data display. If no format has been assigned, hexadecimal format is the default.

All of the following are invalid with the use of the "a" or the "l" commands, and will generate error messages.

"data.format.specification" options:

c Character format.

i Integer format. The maximum window specification is 6. Larger numbers give unpredictable results.

x Hexadecimal format.

w Hexadecimal Word format. This is identical to the "x" format, but data is displayed in 2-byte words meaning that the output will be un-byte-swapped on Intel platforms.

Example

```
!x/mbase<return>     <- Enter this.
!x/mbase .00017A5B=  <- This displays.
!c/mbase<return>     <- Enter this.
!c/mbase ...[=       <- This displays.
!i/mbase<return>     <- Enter this.
!i/mbase 108379 =    <- This displays.
```

data reference specification

references a disk location or D3 Virtual Address. The data format and window specifications, if unspecified, are dictated by previous settings.

The "data reference specification" is one of the three elements used in referencing disk locations within the system debugger; it specifies either a "direct" or an "indirect" address.

Address specifications are represented by the character preceding the address, as follows:

".address"

Indicates the address is a hexadecimal number.

",address"

Indicates the address is a decimal number.

The formats illustrated use the default hexadecimal representations.

Direct reference :

!fidaddr.dataaddr or !dataaddr

"fidaddr" is the frame-id (fid) of the virtual frame, and "dataaddr" is the offset into the virtual frame specified by "fidaddr". If "fidaddr" is not specified, the current process' pcb is the referenced frame. Either of these parameters may be preceded by a period (.), indicating that the address is a hexadecimal value, or a comma (,), which indicates that the address is a decimal value.

Indirect (and symbolic) references :

!symbolname or !/symbolname or !*symbolname

Where "symbolname" is the name of the symbol, as defined in either the "psym", "gsym" or "tsym" files. This displays the register number, displacement, format type, and window of the symbol. The symbol table must be "set" prior to using the indirect reference to a symbol name. See "set-sym".

The content, rather than the definition, of the virtual memory area defined by the symbol is displayed in the window equal to the size specified in the symbol definition or by a window specification.

Indirect references:

/symbolname

This displays the contents of the given symbolname.

!*symbolname

This references the data pointed to by the address contained in the symbol.

!*fidaddr.dataaddr or !*dataaddr

This treats the specified location as a storage register which is used to reference data. The displacement, "dataaddr", is added to the frame address to get the address of the storage register.

Double Indirect References:

!**symbolname or !**fidaddr.dataaddr or !**dataaddr

This is one more level of indirection from the single "*" addressing. The specified location is assumed to contain a storage register which is pointing to another storage register which is pointing to the displayed data.

For "*" and "**" addressing, if the first byte of the medial storage register is a x'82', the element is interpreted as a basic indirect string element and the storage register is taken from two bytes beyond this location. If any of the data fields are invalid as storage registers, the message "addr" appears.

System Debugger functions

briefly discusses the functions and commands that are available using the replace or assignment function, "=".

Once a valid address has been specified, the debugger accesses the requested location and displays its current contents. The cursor is positioned to the right of an "=" sign, indicating that the debugger is ready to either accept any of the following special functions or change the contents of the window.

/ctr1 .0120 =

The following commands are available in response to the "=" prompt character that appears as the result of a legal debugger command:

/ctr1 .0120 =<return>

(carriage return). Returns control to the debugger command processor (!). This leaves the window unchanged.

/ctr1 .0120 =<lf> .0043 =

(linefeed). Displays the next "window" of data, on the same line.

/ctr1 .0120 =<ctrl>+n

1890.94 .0043 =

Displays the address and the contents of the next window on the next line which includes decimal frame and hex offset.

/ctr1 .0120 =<ctrl>+p

1890.90 .0020 =

Displays the address and the contents of the previous window on the next line which includes decimal frame and hex offset.

c*bmsbeg;10 employees ='string

Places the characters "string" at the beginning of the displayed window for the length of string, which may not exceed forty bytes. The string must terminate with a <return>, <enter>, <lf>, <ctrl>+n, or <ctrl>+p.

/ctr1 .0120 =decimal number

Places the value of the decimal number in the displayed window, filling from the right, if the window is 1, 2, 4, or 6 bytes in length, and does not cross a frame boundary. The string must terminate with a <return>, <enter>, <lf>, <ctrl>+n, or <ctrl>+p.

/ctr1 .0120 =.hexadecimal string

Places the value of the hexadecimal string in the displayed window, filling from the left. The string must contain an even number of hexadecimal characters, and may not exceed 38 hexadecimal characters. Notice the "." prefix. This distinguishes between hexadecimal and decimal entries. The string must terminate with a <return>, <enter>, <lf>, <ctrl>+n, or <ctrl>+p.

/h0 =b<return>

1891.f:0 =<ctrl>+n

1891.f:1 ={1 or 0}

The b shows data in binary mode, 1's and 0's are the only legal values allowed to be entered and must terminate with a <return>, <enter>, <lf>, <ctrl>+n, or <ctrl>+p. Notice the most significant bit is bit zero and least significant is bit 7 in a displayed byte.

/ctr1 .0121 =0

(zero). Clears the window to hex zeros. The string must terminate with a <return>, <enter>, <lf>, <ctrl>+n, or <ctrl>+p. Any size window can be cleared with a single zero.

/ctr1 .0120 =a

1891.92 .0120 =

Redisplays the address and contents of the last window.

/ctr1 .4445 =c

1891.92 DE=c4

1891.92 DEFG =c.2;2

1892.94 FG =

These forms change the display type, window and offset if specified, and redisplay either the original field with the new type and/or window specification, or the resulting field if the offset is changed. The string must terminate with a <return>, <enter>, <lf>, <ctrl>+n, or <ctrl>+p. "c" is for character mode display and ";" is the window width. "c.2;2" moves the window over bytes and displays a window of 2 bytes.

:

pushes a level from within the system debugger.

When a <return> is issued at the ":" prompt, control returns to the "!" prompt.

Syntax

:<return>

Example

```
!:<return>
::
::who<return>
54 dm dm
::<return>
!
```

```
The example above assumes that you were at level 1, you pushed a level to
level 2, and then returned to level 1.
```

a

displays the address of the instruction currently being executed.

Example

```
a
Will display:
!aau.input:018
This indicates the process is in the "au.input" subroutine, at relative
address x'018' within the mode.
```

addd

adds the two decimal integer numbers immediately following in the command line and displays the result.

Syntax

addd number number

Example

```
!addd 12 13<return>  25
The result displays on the same line as the command.
```

addx

adds two hexadecimal numbers together and displays the result a hexadecimal number.

Syntax

addx hex.number hex.number

Example

```
!addx 1F 33<return>   52
```

b

establishes a "breakpoint" condition which causes the system to enter the system debugger when the condition is met.

The optional "offset" sets a particular breakpoint within the mode, at the specified "offset". If no "offset" is specified, the debugger "breaks" at every available breakpoint within the mode. Every assembler instruction is an available breakpoint.

The "data.reference.specification" may contain a maximum of two numeric parameters, hence no window specification is expected.

A plus (+) character is displayed for each breakpoint successfully entered into the table, until the table is full. The maximum number of breakpoints at one time is four.

Syntax

b{data.reference.specification}{.offset}

Example

```
b/5.157
Sets a breakpoint at offset x"157" in frame 5.
b/5
Breaks at every available breakpoint in frame 5.
b/au.input.0
Breaks at the first entry point of the "au.input" mode.
```

c

displays debugger data in character (ASCII) format.

Syntax

c{data.reference.specification}

Example

```
c*bmsbeg
This makes an indirect reference to "bmsbeg" and displays a four-byte window
by default.
c16274.c;6
Will display:
!c16274.c;6 s^2322=
 This will display, in character format, the contents of frame 16274, offset
x'c', displaying a window of 6 bytes.
```

d

displays the table containing current breakpoints, traces, data breakpoints (y-trace tables), and frame replacement specifications.

Example

```
d
brk tbl au.input
trc tbl
c*R13;30 : 151036.4C9
c*R14;40 : 449076.724
chg tbl
i/R0.C;4 : 1538.C  -65308
The above shows a breakpoint set at mode "au.input", character traces on
registers 13 and 14, and a change-trace ("y-trace") on the process' internal
accumulator, D0.
```

data window specification

number of bytes to display at the given address. The "window" is one of the three elements used in referencing disk locations within the system debugger.

The "data.window.specification" indicates the number of bytes to display and has the general form:

;window

This represents the number of bytes to display. The maximum is limited to the size of the requested data.

Offset specifications (explicit):

;offset,window or ;offset.window

This designates the offset and window, where offset is a positive or negative number indicating the offset from the "data.format.specification", and window is a positive number as above. This form works for traces, except in the case that the location is an indirect reference from a storage register whose location is specified by the form: !fidaddr.dataaddr.

Offset specifications (implicit):

;coffset or ;coffset.window or ;coffset,window

Where offset and window are functionally the same as above, and the "c" (code) designates the number of fields. If the window specification is not included, the implicit window derived from the field type is used, otherwise the specified window is used.

The possible designators for the c parameter are as follows:

b field width = 1 bit.

c field width = 1 byte.

d field width = 4 bytes.

f field width = 6 bytes.

h field width = 1 byte.

r field width = 8 bytes.

s field width = 6 bytes.

t field width = 2 bytes.

Or:

;c

Where the "c" parameter is as above, but in this form, an offset of zero is assumed, along with the implied window and dfs.

In the specific case of bit (b) display, the general form:

;boffset.window or ;boffset,window

This form specifies "bit display", starting at bit zero, the offset from the addressing base, for a width of "window" bits. Bits and bit fields may be traced with either trace. The displacement specified by a symbolically-addressed bit is in bits. Therefore, the form "fidaddr.dataaddr" will treat the dataaddr as a "bit" count in the direct-reference form.

divd

divides the first integer number provided by the second integer number provided and displays the result as an integer quotient and remainder.

Syntax

divd number number

Example

```
!divd 52 24<return>  2 4
The result displays on the same line as the command. "2" is the result, with a
remainder of "4".
```

divx

divides the first hexadecimal number provided by the second hexadecimal number provided and displays the result as a hexadecimal quotient and remainder.

Syntax

divx hexnum hexnum

Example

```
divx 7F D<return>   9 A
The result is displayed on the same line as the command.
```

dtx

converts a given decimal number to its corresponding hexadecimal equivalent.

Syntax

dtx decimal.number

Example

```
dtx 1024<return>  400
The result displays on the same line as the command.
```

e

toggles the system debugger single-stepping feature on or off.

Example

```
e1<return>  <- Enter this.
!e  on      <- Now in single-step mode.
!<ctrl>+j     <- Executes one instruction.
E au.rtn.level:00C => 15  bbs ilk.fail,item.lk
(above line is execution address.)
!e  off     <- Cancels single-step mode.
!<ctrl>+j     <- Returns to normal processing.
```

end

terminates the current process and returns either to TCL (if at the first level), or to the previous level

Example

```
select entity    <= Invokes a lengthy select process.
<break>          <= Interrupts process.
!                <= Debugger prompt appears.
!end<return>     <= Finishes "select" ...
:                <= ... and returns to TCL.
In this example, the assumption is made that the <break> key is defined to
"break" into the debugger. See "brk-debug", "brk-level",
"esc-level", and "esc-debug".
```

g

transfers program execution to the address specified within the currently executing virtual space (abs).

When used alone, the "g" command continues processing from the current program location.

Syntax

g{data.reference.specification}

i

displays debugger data in integer format.

The "i" command, followed by a <return>, displays subsequent output in integer format.

Syntax

i{data.reference.specification}

Example

```
i172628.b8;2
 This displays the contents of frame 172628 at offset x'b8',
displaying a 2-byte window in integer format.
```

k

"kills" one or all breakpoints previously set with the "b" command, and removes the entry or entries from the breakpoint table.

See the "b" command for limitations of breakpoint addresses.

The "k" command, followed by a <return>, kills all current breakpoints in the table. A minus (-) character is displayed for each breakpoint entry successfully removed from the table.

If a data.reference is included, only the breakpoint corresponding to that reference will be "killed".

Syntax

k{data.reference.specification}

l

displays the frame linkages for the given fid.

If the fid is not specified, the links for the frame most recently implied by a data reference specification are displayed.

A data format specification may not be used with this command and will generate an error message.

See "<ctrl>+n", "<ctrl>+p" in "Functions: System Debugger".

Frame linkages may be replaced. On the right side of the "=" sign, the values of the four elements may be changed using the following format:

lfid.address ..... =n1,n2,n3,n4

Where "n1", replaces the number of contigous "forward" links; "n2" replaces the "forward" linked frame; "n3" replaces the "backward" linked frame; and "n4" replaces the number of contiguous "backward" links.

Syntax

l{data.reference.specification}

Example

```
L109681<return>     <= Enter command.
!L109681  0  :   109684   109514  :  0 =   <= This displays.
This indicates that frame "109681" has a forward link to frame "109684" and a
backward link to "109514".
In this example, "109681" is the current fid. The first "0" is the number of
contigous "forward" links. "109684" is the "forward" linked frame. "109514" is
the "backward" linked frame. The last "0" is the number of contiguous
"backward" links.
```

m

toggles the modal trace function on or off. When it is "on" the debugger is entered each time a mode boundary is crossed.

Syntax

m

md

enters the monitor debugger from the system debugger.

Example

```
debug
This enters the system debugger from TCL
md
Now, within the system debugger, the "md" command enters the monitor debugger.
```

me

assigns all PCB and symbolic reference data specifications to the specified port. This provides the invoking line with the ability to use the debugger and symbolic references of the requested port's workspace area.

The "me" command, followed by a <return>, resets the pointer to the current virtual process' pcb.

Syntax

me{port.number}

Example

```
me5
Any PCB elements displayed will be those of pib 5.
i/t0
 Displays the contents of T0 belonging to pib 5.
me
Resets to reference the port's own PCB elements.
```

muld

multiplies the first integer number provided by the second integer number provided and displays the result as an integer number.

Syntax

muld number number

Example

```
muld 14 78<return>  1092
The result displays on the same line as the command.
```

mulx

multiplies the first hexadecimal number provided by the second hexadecimal number provided and displays the result as a hexadecimal number.

Syntax

mulx hexnum hexnum

Example

```
mulx F A<return>   96
The result displays on the same line as the command.
```

n

ignores any previously designated debugger breakpoint conditions for a specific number of occurrences.

This command prints the instruction address and other characteristics for the number of times specified in the tally reference counter prior to returning to the debugger command level. If a real error is encountered, entry into the command level is automatic.

The "n" command, followed by a <return>, cancels this function. Thereafter, all breaks will re-enter the debugger command level.

Syntax
n{tally reference counter}

off

stops processing at all levels on the current port and returns control to the initial logon prompt.

p

toggles the display of terminal output either on or off.

r

displays the data referenced through a specific register number.

Syntax
r{regnumber}

subd

subtracts the second integer number provided from the first integer number provided and displays the result as an integer number.

Syntax
subd decimal.number decimal.number

Example
```
subd 9766 2234<return>  7532
```

subx

subtracts the second hexadecimal number provided from the first hexadecimal number provided and displays the result as a hexadecimal number.

Syntax
subx hexnum hexnum

Example
```
subx 7F D<return>   72
```

t

sets a trace table entry, indicating to the debugger to display the specified data element, along with the contents of the break and trace tables, on each break.

If the data reference specification is omitted, the "t" command toggles the trace function on or off.

The system displays the appropriate condition.

A plus (+) character is displayed for each trace table entry successfully entered into the table, until the table is full. The maximum number of trace table entries is four.

Syntax
t{data.reference.specification}

time

displays the current system time and date.

Example

```
time<return>   08:44:43  25 Feb 1992  Tuesday
The time/date display appears immediately to the right of the command.
```

u

removes trace table entries previously specified with the "t" command.

A minus (-) character is displayed for each trace table entry successfully removed from the table.

The "u" command, followed by a <return>, clears all commands previously specified with the t command.

Syntax

u{data.reference.specification}

w

displays subsequent output in hexadecimal words.

The "w" is a formatting specification in the system debugger similar to the "x" format. However, the "w" format prints the data as 2 byte "words" instead of bytes. This means that on Intel platforms, data will be un-byte-swapped so that it can be displayed in a human-readable format.

To indicate that the "w" format is in effect, brackets are displayed around the data.

To force input data to be entered by words, use the "[" modifier before the data.

When the "w" format is in effect, the "[" modifier is assumed when entering hexadecimal input.

The "w" format may also be specified for traces.

Syntax

w{data.reference.specification}

Example

```
 Assume the following occurs on an Intel platform:
 debug
!/r11 .00002c010000151e=w
1557.158  [.0000012c00001e15]=
!x/t0 .e400=w
1557.e  [.00e4]=[.0102]
!x/t0 .0201=w
1557.e  [.0102]=.0304
!w/t0  [.0304]=
!tw/t0+ 1557.e
!e on
!g
E tcl1:01c
 1557.e [.0102]
!end
```

x

displays subsequent output in hexadecimal format.

Syntax

x{data.reference.specification}

xtd

converts a given hexadecimal number into its equivalent decimal format.

Syntax

xtd hexnum

Example

```
xtd 1a<return>   26
The result is displayed on the same line.
```

y

sets a y-trace table entry indicating to display the specified data element, along with its address, each time the data element changes.

If the "data.reference.specification" is omitted, the "y" command toggles the function on or off. The current value of the data, stored in aligned words, is kept with the address data, so that the table element size will change with varying sizes of data. A plus (+) character is displayed for each entry successfully entered into the table, until the y-trace table is full. The maximum number of entries is four.

Syntax

y{data.reference.specification}

z

cancels "break" conditions previously set with the "y" command.

The "z" command, followed by a <return>, cancels all data breaks set by a previous "y" command. A minus (-) character is displayed for each entry successfully removed from the y-trace table.

Syntax

z{data.reference.specification}

[

indicates the entry of hexadecimal words at a system debugger data input prompt.

See the system debugger "W" command for more information.

Tape

multi-user tape

A port can now attach more than one device to itself and ports can independently access different tape devices simultaneously. This expands the functionality of t-det. It now requires a device number or else it will detach all tape devices held by the port. T-det (u will release all tape devices on the system.

The old t-att command is now used primarily to change block sizes.

A new command t-act is available to switch between devices if more than one is attached to a single port. Remember, you can only issue one tape command at a time.

tape format error

see restore errors (Files, Definition)

tape label

defines a D3 "tape" label.

Traditionally, D3 "tapes" have an identifying "label" written on the first 80 bytes of the tape.

```
Tape     Element Element
Position Length  Description
-------- ------- ---------------------------------
01         01    Label Identifier. Always an "L".
02         01    (space)
03 - 06    04    Record Length (in hexadecimal)
                 "01F4" = 500-byte record
                 "03E8" = 1000-byte record
                 "4000" = 16384-byte record
07         01    (space)
08 - 15    08    System Time at time of tape-writing.
                 "00:00:00" format
16 - 17    02    (spaces)
18 - 28    11    System Date at time of tape-writing.
                 "MMM DD YYYY" format
29         01    (space)
30 - 77    48    Label Header Text.
78         01    (space)
79 - 80    02    Tape volume (reel) number.
-------- ------- ---------------------------------
```

If the tape is created by a "t-dump", the label contains the source filename and text specified in "t-dump"s "heading" option (if any).

If the tape is created by a "file-save", "account-save", or "save", then it contains the word "DATA", and optionally, the information entered at the "file-save tape label:" prompt.

If the tape is created as Spooler output, the label contains the word "SPOOLER", and optionally, the account name and user-id of the creator.

chg-device

changes a tape device.

Devices which can be used as tapes are listed by the "list-device" command. Some characteristics can be changed to either modify the behavior of a device or to replace a device by another one.

The changes made with this command are not permanent and are lost when the virtual machine is rebooted. If no option is specified, the current value is retained. The following options and values are recognized:

name=newname

This specifies a new device name. Since this is likely to completely change the device, all other options should be specified as well. Spaces are not allowed. The length of the new name cannot exceed 63 characters.

type=[floppy|halfinch|8mm|sct]

Specifies the type of device.

density=[360K|1.2M|1.44M|pseudo]

This specifies the density for a floppy. "pseudo" indicates an 'infinite' density device, such as pseudo floppy on hard disk.

density=[1600|3200|6250]

Specifies the density for a half inch device. This field is essentially for information purposes, since, on most Unix implementations, the density of the drive is encoded in the device name (as a suffix, for instance).

density=[low|standard|high]

Specifies the density for a SCT. This field is essentially for information purposes, since, on most Unix implementations, the density of the drive is encoded in the device name such as a suffix.

label=[80|512]

Specifies the label block size. Valid for 8mm tapes only. The D3 label is always 80 bytes long, but some Pick implementations write it in a 80 byte block, while D3 writes it in a 512 byte block. To be able to exchange 8mm tapes between systems, it might be necessary to change this option.

blksize=default block size

Specifies the default block size. Usually, this is 16384 for Native systems, and 512 for hosted Unix systems.

Syntax

chg-device device.number option=value { option=value ... }

chg-device ?

Example

```
chg-device ?
USAGE: CHG-DEVICE devnum option=value {option=value ..}  {(option}
options:
name=newname
type=[ floppy | halfinch | 8mm | sct ]
density=[360K|1.2M|1.44M|pseudo] (floppy)
        [ 1600 | 3200 |6250 ]  (1/2 inch)
```

```
        [ low | standard | high ]   (sct)
label=[ 80 | 512 ]             (8mm only)
blksize=newblksize
 Displays the help.
chg-device 7 name=/tmp/pseudotp
Changes the name of the Unix file or pipe used as a pseudo tape.
```

set-8mm

sets the 8mm Data Cartridge as the tape device, by invoking the "set-device" verb and passing in the argument "8mm" and an "i" option.

The default block size is 16384.

The list of installed devices is displayed with an "*" in front of the selected device.

If more than one 8mm tape is present on the system, this command selects the first tape in the configuration file. To select another device, use "set-device".

Syntax

set-8mm {(option}

Options

c Changes tape block size. The current tape block size is displayed and the user is prompted for the new block size (0 or 512). A tape can only be read with the same block size at which is was written. Normally, D3 uses a block size of 0, which means variable length blocks, but this option is provided to read tapes written on an earlier version of Pick, or for compatibility with other systems. This option might not be available on all implementations.

set-half

attaches the half-inch tape drive to the current process for subsequent input or output activity. The following "t-att" command allows changing the default block size from its default of 8192.

set-sct

attaches the streaming cartridge tape drive to the current process. This must be executed before any other tape handling commands.

Specifies a 1/4-inch streaming tape unit as the peripheral storage device. "set-sct" automatically specifies a tape-attach command (t-att) and a rewind (t-rew).

"blocksize" may be any value between 512 and 16384. The default is 16384. The block size must be a multiple of 512.

Make sure the hardware is present for a 1/4-inch drive. Otherwise, the port may wait forever.

If only one density is available then it is used. The default is high density.

To improve reliability of Streaming Cartridge Tapes (SCT's), it is strongly advised to retension the tape prior to performing any tape input or output operations. (see "t-ret" or "t-reten").

On D3 Unix systems, this verb is provided for compatibility. (see "set-device").

Syntax

set-sct {(blocksize)}

Options

l Low density, if available.

h High density, if available.

m Assigns number of buffers using the total real memory minus 256K or 768 buffers, or 384K on a 640K system. Pick Systems strongly advises using this option to increase tape streaming and decrease the number of stop/starts, which decreases tape reliability.

r Indicates that the tape is in "old" format, meaning that it was created on a release prior to 2.2.

s Standard density, if available.

set-sct-dma

specifies the DMA (Direct Memory Access) channel for the 1/4-inch streaming cartridge tape unit.

Syntax

set-sct-dma channel.number

Options

n (integer number) Specifies a DMA channel number between 1 and 7.

I set to interrupt mode (requires IRQ be set to 5).

N set to non-interrupt mode

set-tape-type

specifies the type of half-inch tape drive being used.

"Pick-style" tapes use an 80-byte tape label. "Microdata-style" used a 50-byte label.

Syntax

set-tape-type {(option}

Options

c Sets Cipher-type half inch tape. This puts the tape drive into start-stop mode, allowing the Cipher tape drive to properly sense end-of-tape. Cipher tape drives can run in streaming mode (the "k" option) provided that it does not stream past the end-of-tape marker. This causes the tape drive to get an end-of-tape error which can be cleared by turning the tape drive off and back on again.

k Sets Kennedy-type half-inch tape. This allows the tape to stream when searching for end-of-file or end-of-tape.

m Microdata format. This sets a flag is set in the TCB (tape control block) so that all subsequent tape reads will be performed in Microdata format.

p Pick format. This resets the "m" option.

t-act

Selects a previously attached tape device as the currently active device to read and write to.

D3 releases that have multi-user tape not only can have many users using many tapes simultaneously, a single user can have multiple tapes attached to himself. This command allows the switching between tapes.

Syntax

t-act device.num

Example

```
set-device 1
set-device 2
t-dump md
```

```
t-act 2
t-dump md
t-act 1
t-dump bp
t-act 2
t-dump bp
t-det
```

t-att

attaches the tape unit or floppy disk drive to the current process unit and optionally assigns the blocksize to the tape i/o buffer.

"blocksize" is an integer number indicating the number of bytes in each block.

Floppy disks may be set at any number between 20 and 512, but are usually set to 500 or 512. The default is 500.

Half-inch tapes may be set to anything between 512 and 16384, inclusive. The default is 8192.

Streaming cartridge tapes (SCT) may be set to anything between 2048 and 16384 and must be a multiple of 512. The default is 16384.

8-millimeter tapes may be set to anything between 512 and 16384 and must be a multiple of 512.

After attaching the device to the current process, all of the regular "tape" handling verbs, like "t-rew" and "t-fwd", are available, even when using floppy diskettes. Diskettes must be "rewound" before writing to them or reading from them. Usually, this doesn't take long.

If the tape unit is attached to another line, the process displays the port that has it attached. The "u" option in the "t-det" command attaches the tape "unconditionally", regardless of what it may be doing. This may be necessary if the transaction logger is enabled. (see "t-det").

The "t-att" verb should be used before any tape manipulation process, such as executing tape control verbs, generating print file output to the tape using the "t" option in "sp-assign" or "sp-edit", executing tape reads and writes in FlashBASIC, or generating tape output using the "reformat" and "sreformat" verb.

All tape manipulation processes on the system check for attachment, attach the tape if possible, generate the required message, and terminate if the tape is not available.

The implied "t-att" uses the current tape block size specification and remains "set" until one of the following events occur:

1) using "t-att" with a numeric argument.

2) using "t-att" without a numeric argument.

3) using any tape verb which checks for tape attachment.

4) executing the "t-rdlbl" verb when a labeled tape is mounted. The tape block size is stored in labels written to tape. By reading a tape label through "t-rdlbl" or "t-read", the current blocksize is changed to the size stored in the label read.

Syntax

t-att {blocksize} {(options}

Options

u Unconditionally attaches the tape. It is strongly advised to verify that the tape is not actually being used before stealing it from another process.

z Unconditionally attaches the tape, except if the tape is attached to the transaction logger.

t-bck

"backspaces" the tape to the end of the previous file or a specified number of blocks.

Syntax

t-bck {number}

Options

number Indicates the number of blocks to backspace. If "number" is not specified, the tape backspaces to a position immediately preceding the previous eof mark, or to load point. Before reading the next file, a "t-fwd" must be issued to position the tape after the eof mark.

t-bsf

"backspaces" a given number of files.

If "number" is omitted, this command behaves like "t-bck".

Syntax

t-bsf {number}

t-bsr

"backspaces" a given number of records.

If "number" is omitted, this command behaves like "t-bck".

Syntax

t-bsr {number}

t-cascade

Cascades or "links" one tape device to another tape device, either in a chain or a loop. Instead of prompting for the next reel, the system will use the forwardly linked device specified.

Tape attachment is not checked. This verb simply updates the links of the tape devices.

When all the devices have been used in a chain, the system will remain on the current device and prompt for the next reel, if necessary.

If the links are set up as a loop, the system will never prompt for the next reel. It will be up to the operator to mount the next reel properly.

All tape drives involved in the cascade must be at the same block size or be able to be changed to the starting drives block size automatically by the system for the cascade to succeed. Remember, certain tape types like floppy drives are limited in their block sizes available.

Syntax

t-cascade device.num, device.num, ... {options}

Options

c Clears all links for the first device number specified or all devices if none is specified.

Example

```
t-link 6,8
Tape device 6 will cascade into tape device 8 if the end of reel is
encountered on device 6. No prompt for the next reel will occur.
t-link 2,3,2
Tape device 2 and 3 are linked as a loop, 2 is linked to 3 and 3 is linked
back to 2.
```

```
t-link 2,3,4,5
Tape devices 2 to 5 are all linked or cascaded together. Only, after device 5
is filled, the system will prompt for the next reel.
t-link 4 (c
Clears the links associated with tape device 4. This device will now prompt
for next reel when the end of tape is reached.
```

t-chk

checks a tape for parity errors.

When the "a" option is omitted, the current tape file is checked.

At the end of the process, the number of files checked is displayed.

See "dummy restore" for validating backup media.

Syntax

t-chk {(options)}

Options

a Checks to end of data.

l Displays label data if one is found after an end-of-file mark.

p Directs output to system printer, via the Spooler.

t-det

detaches the tape from the current process, or from another process.

If no device.number is specified, all devices currently attached to this process are detached.

On D3 Unix systems, this command detaches the tape from the current port, so that other processes, other Unix users, or other D3 virtual machines may use it.

The device.number must be specified or all tape devices attached to this port are detached.

On D3 Unix systems, the "u" option has to send a Unix signal to the process which owns the tape to force it to close the device. This is done through the 'trap' mechanism (see the TCL command 'trap'). The default trap for this signal is the TCL command 't-det'. If the System Administrator disables this trap, or uses another command (like displaying a message to ask the user to detach the tape instead of taking the tape forcefully), the command fails with the message "The tape is not available for use now". The "z" option works only if the owner is the transaction logger.

Syntax

t-det {device.number} {(option}

Options

u Detaches the tape unit from any port except the Spooler, which detaches automatically at the end of each job. This requires a sys2 privilege level. The device.number must be specified or all tape devices are detached.

z Unconditionally detaches the tape, except if the tape is attached to the transaction logger.

t-eod

positions the tape between the two end-of-file (eof) marks after the last file on the tape.

The last file on tape is marked with two eof marks, and is considered end-of-data (eod).

Syntax

t-eod {(options}

Options

l Displays label data if one is found after an end-of-file mark. If the tape record size is different, it will change automatically.

p Directs output to system printer, via the Spooler.

t-fsf

forward a tape a given number of files. "forward space file".

If "number" is omitted, this command behaves like "t-fwd".

Syntax

t-fsf {number}

t-fsr

forward a tape a given number of records. "foward space record".

If "number" is omitted, this command behaves like "t-fwd".

Syntax

t-fsr {number}

t-fwd

moves the attached media forward to the next file, or moves the tape forward a specified number of blocks.

"t-fwd" stops automatically at an "end of file" (eof).

Syntax

t-fwd {number}

Options

number (integer number) Indicates the number of blocks to move. The maximum value for "number" is 32767. If "number" is not specified, the tape spaces forward to the position immediately beyond the next eof mark.

t-link

Cascades or "links" one tape device to another tape device, either in a chain or a loop. Instead of prompting for the next reel, the system will use the forwardly linked device specified.

Tape attachment is not checked. This verb simply updates the links of the tape devices.

When all the devices have been used in a chain, the system will remain on the current device and prompt for the next reel, if necessary.

If the links are set up as a loop, the system will never prompt for the next reel. It will be up to the operator to mount the next reel properly.

All tape drives involved in the cascade must be at the same block size or be able to be changed to the starting drives block size automatically by the system for the cascade to succeed. Remember, certain tape types like floppy drives are limited in their block sizes available.

Syntax

t-link device.num, device.num, ... {options}

Options

c Clears all links for the first device number specified or all devices if none is specified.

Example

```
t-link 6,8
Tape device 6 will cascade into tape device 8 if the end of reel is
encountered on device 6. No prompt for the next reel will occur.
t-link 2,3,2
Tape device 2 and 3 are linked as a loop, 2 is linked to 3 and 3 is linked
back to 2.
t-link 2,3,4,5
Tape devices 2 to 5 are all linked or cascaded together. Only, after device 5
is filled, the system will prompt for the next reel.
t-link 4 (c
Clears the links associated with tape device 4. This device will now prompt
for next reel when the end of tape is reached.
```

t-rdlbl

reads the label from the attached magnetic media and initializes the internal label storage area.

Syntax

t-rdlbl {(options}

Options

reel.number (hexdecimal number) Indicates reel number.

a Allows beginning from "any" reel number, rather than sequentially. The process attempts to read a label and then data. If the label is invalid, it backspaces over the data read.

t-read

reads from the magnetic media unit attached to the current process and displays the contents of each block read to the terminal or printer.

Any type of Pick-formatted media may be read with this command.

Syntax

t-read {(options}

Options

block.number{-block.number} Reads a specific number of blocks or a range of block numbers. block.number specifies number of blocks to read; block.number-block.number specifies a range of blocks. Blocks are counted from the current position of the tape. If the entire option is omitted, all tape blocks up to the next eof are read.

a Converts EBCDIC format to ASCII.

n No pause. Suppresses the pause at the end of each page on the terminal.

p Directs output to system printer, via the Spooler.

x Displays in hexadecimal format.

Example

```
t-read
L 01F4 16:12:28  16 Jan 1997   Tape Label
RECORD = 1
1 FORMATC^****...
51 ****...**^* THIS PROGRAM FORMATS A P
.
.
.
[94]  END OF FILE
```

t-rew

rewinds the attached media to the beginning, or "load point".

t-select

attaches a "tape" device to the current process.

"tape" devices include floppy diskettes, streaming cartridge tape (sct), half-inch, 2mm cartridge tape, 4mm dat devices and "psuedo-floppies" such as communication ports and inter-virtual-machine files.

This verb is the root command from which "set-floppy", "set-sct" and "set-half" operate.

If no options or parameters are specified, "t-select" displays all devices defined for the system and prompts for a selection.

"device.num" may be expressed as a numeric literal. This number must be a numeric integer between 0 and 15 and correspond to a predefined tape device. On Unix-based systems, these devices are defined in the system configuration file. If a comma separates more than one device.num then they are linked together. See t-link verb.

"keyword" may be any combination of characters (except numeric integers between 0 and 15) appearing in the "TYPE", "DENSITY" or "DEVICE NAME" columns of the "t-stat" verb. These keywords are used to try to pinpoint the desired device from a captured list. On some systems there may be more than one of a particular device (two 5-1/4" floppies, for example). If there is not sufficient uniqueness in the parameters, the process will always find the first floppy.

On Unix implementations, if there is more than one floppy drive, and one of them is a 5-1/4" floppy, the 5-1/4 is always designated as drive A. If no other parameters but FLOPPY are given, the verb will assume drive A, 5-1/4" high density floppy.

Syntax

t-select

t-select {?}

t-select {device.num} {,device.num} {(options)}

t-select {keyword} {(options)}

Options

3 3-1/2" floppy.

5 5-1/4" floppy.

9 9-track, 1/2" tape.

a 5-1/4" floppy.

b 3-1/2" floppy.

c Changes the 8mm tape block size. The current tape block size is displayed and the operator is prompted for the new block size (0 or 512). On D^3 Unix Systems, the default block size is set at "0" (zero) at install time. Zero means variable-length.

f Floppy (3-1/2" or 5-1/4" as default)

h High density (1.44M for 3-1/2" floppy, 1.2M for 5-1/2" floppy, 6250 bpi for 9-track tapes, 150M for Quarter Inch tapes).

i Suppresses display of devices after the command is complete (used in macros).

k Hard disk "pseudo" tapes. These devices are removable or fixed hard disks or regular Unix files. Their size is fixed and determined by the device or by the maximum file size for a user. The main usage for these devices is for small, fast t-dumps/saves, transaction logging or incremental saves.

l Low density (720K for 3-1/2" floppy, 1600 bpi for 9-track tapes).

m Medium density (720K for 5-1/4" floppy, 3600 bpi for 9-track tapes).

n Network "pseudo" tape. This device has an 'infinite' size. It is assumed another system is reading the data at the 'other end' of the network.

q Quarter Inch tape.

s Standard density (360 for 5-1/2" floppy, 60M or 120M for quarter-inch (SCT) tapes.

w Prevents an automatic rewind on the device. This option must be used to set the device to a floppy with an unformatted floppy disk in the drive.

x Default block size

Example

```
Assume the following device list for all examples:
Tape Status
#   Type                Density             Device Name
=============================================================
0   floppy              3-1/2" 1.44m        /dev/rfd0h
1   floppy              3-1/2"  720k        /dev/rfd0l
2   floppy              5-1/4"  1.2m        /dev/rfd1.15
3   floppy              5-1/4"  360k        /dev/rfd1.9
4   quarter inch        high (350m)         /dev/rmt2.1
5   quarter inch        standard (120m)     /dev/rmt2.5
6   floppy              infinite            /tmp/floppy
The following statements all attach the 5-1/4" floppy, high density:
t-select 2
t-select floppy  (by default and for compatability)
t-select floppy (a  (by Unix convention)
t-select 1.2m
t-select /dev/rfd1.15
The following attach the 3-1/2" low density floppy:
t-select 1
t-select floppy (b
t-select 720k
The following attach the psuedo floppy disk file called "/tmp/floppy"
on a Unix-based system:
t-select 6
t-select floppy infinite
t-select /tmp/floppy
```

t-space

moves the attached media forward a specified number of files, displaying each tape label as it is encountered.

If a number is not specified, the process stops and indicates that a "count" is required.

Syntax

t-space {number}

Example

```
t-space
no. of files? 4
Moves the tape device forward 4 filemarks. Each label is displayed.
```

t-stat

displays a report of the current attachment status for all tape devices defined on the system. It shows the device number and description, Unix process name, density and currently attached user.

Syntax

t-stat {(options)}

Options

p directs the report to the spooler.

Example

```
t-stat
 Tape        Status           16 Jan 1997  14:30:40
 #   Type             Density           Owner   Device Name
-------------------------------------------------------------------------
 0 | Floppy        | 3 1/2" 1.44M    | 191*+ | /dev/rpdsk/4
 1 | Quarter Inch  | High            |       | /dev/rmt/0n
 2 | 4mm DAT       |                 |       | /dev/rmt/1n
 3 | 4mm DAT       |                 |       | /dev/rmt/1un
 4 | 8mm Tape      |                 |       | /dev/rmt/2n
 5 | 8mm Tape      |                 | 141   | /dev/rmt/2un
 6 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/abs
 7 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/data
 8 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/ref
 9 | Network       |                 |       | /home/tmp/pipein
10 | Floppy        | Pseudo Floppy   | 191*  | /home/tmp/floppy
11 | Floppy        | Pseudo Floppy   |       | /n/dev/home/tmp/floppy
-------------------------------------------------------------------------
```

t-unld

see t-unload.

t-unload

rewinds and, depending on the device, unloads the attached tape.

On D3 Unix systems, this command does the usual 'tape unload'. Depending on the Unix version, this command may wait for the tape to be unloaded before returning.

t-verify

validates the integrity of "file-save" or "account-save" backups.

The process can compare the contents of the backup media with the corresponding data on disk, or simply check the media to ensure that it has the correct backup format.

This process first attempts to attach to the tape. If it is already attached to another process, the following message appears:

tape attached to line {port.number}

[1144] the tape is not available for use now.

"port.number" is the port with the tape attached. See the "where" command. The process with a "t" under the "stat" column has the media attached.

Items displayed are in file hierarchy format:

mds > account > dictionary > data section

By default, only account names are displayed. All errors found are displayed. The display pauses at the bottom of each screen and disk and tape comparisons are made.

The "t-verify" is an option provided during the "file-save" process. If it is selected, errors are logged to the "tv.log" file.

Syntax

t-verify {(options}

Options

number (integer number) Indicates "choke" count. This specifies the maximum number of errors allowed before abort. The tape is left at the abort point. By default, the errors never "choke" and just continue down the screen.

? Displays brief on-line help.

a Validates an "account-save". This verifies all accounts until the "end of save" mark on tape, by forwarding to the beginning of each account and verifying its contents. This option is useful for "file-save" reels other than reel 1 (one).

e Displays errors only.

f Displays all file names found on the tape in hierarchical format. If used with the "i" option, item-ids are also displayed.

g Suppresses group "hashing". Skips hashing test on all items. No errors will be found if items are in the wrong group.

i Displays all items found on the tape in file hierarchy format. The display is 'in place' for each file (i.e., all items for that file display on the same line). If used with the "f" option, items are displayed in the "form-feed" format, showing each item on a new line.

l Logs errors to a log file. Attribute 1 of the log file is the error message from the "messages" file. Attribute 2 is the element that caused the error, in hierarchical form. The "l" option prompts the operator for the log file name.

n Activates "nopage" function on output to terminal.

p Directs output to printer, via the Spooler.

q Queries operator on each error found. The operator gets a choice to either continue, quit or skip file system comparisons for the current file.

t Verifies contents of tape only.

x Displays index b-trees found on tape. This is useful for determining if the indices were saved with the file system. See the "b" option in the "save" command for saving indices.

t-weof

writes an "end of file" mark at the current position on the attached magnetic media. It also causes a flush of the last buffer written. This is important to ensure any partially filled buffer was written onto the tape.

Writing two eof marks in a row creates an end-of-data (eod) mark.

t-wtlbl {"label.info"}

```
writes a Pick-format (80-byte) tape label at the current position on the
attached magnetic media.
```

TCL

gateway to all D3 functionality.

The Pick System Terminal Control Language (TCL) is a system-level command language with system-defined or user-defined statements that can be executed individually or sequentially.

System-defined statements are called TCL verbs or commands. User-defined statements are: macros, menus, PROC's, and cataloged FlashBASIC programs. The first word of a TCL statement must be either a system verb, macro, menu, PROC or cataloged FlashBASIC program.

TCL commands can be typed in when the TCL prompt ":" (colon) (or ">", when a select list is active) displays. The completed command is entered for processing by pressing <return>, <enter>, <linefeed>, or <ctrl>+m. Because mistakes do occur, the TCL editor (UP) and TCL command stack facilities are provided. The UP/TCL editor allows corrections to commands after they are entered, but before they are executed. The "tcl-stack" file stores every TCL command, so that they may be recalled for correction and/or execution.

D3 also has Access Query Language (AQL) commands. AQL is a system-level information retrieval language that allows users to query data bases without writing complex programs. AQL uses the TCL commands as verbs with the addition of connectives to specify how the information is to be manipulated and output.

Another feature within the TCL commands are the Spooler commands. These commands control how information is output to the printer.

Example

```
who
create-file test.file 3 7
copy md * (t
compile bp *
sort entity by name name
```

Terminal Control Language (TCL)

The Pick System Terminal Control Language (TCL) is a system-level command language with system-defined or user-defined statements that can be executed individually or sequentially. System-defined statements are called TCL verbs. User-defined statements are: macros, menus and cataloged FlashBASIC programs. The first word of a TCL statement must be either a system verb, macro, menu or cataloged FlashBASIC program.

TCL commands can be typed in when the TCL prompt colon (":") displays. The completed command is entered for processing by pressing <Ctrl>+m or <Return>. Because mistakes do occur, TCL editor and TCL stack facilities are provided.

The TCL editor allows corrections to commands after they are entered, but before they are executed. The TCL stack each command entered at the TCL prompt and allows you to recall commands for correction and/or execution. Refer to the sections "Editing TCL Commands" and "TCL Stack" below and to the entries tcl stack and tcl edit commands for more information.

TCL commands include the AQL and Spooler commands. AQL is a system-level information retrieval language that allows you to query your data base without writing complex programs. The Spooler commands allow you to control how information is output to the printer.

Editing TCL Commands

The TCL editor uses the Update processor (UP) to enter commands, so the TCL editor commands are similar to the UP commands. A TCL command may be created and edited as if it were a paragraph. Pressing a <Ctrl>+m or <Return> key within the TCL command processes the entry.

The TCL editor is initially in the overtype mode. To toggle between overtype and insert mode, type <Ctrl>+r. The following commands function the same in the TCL editor as they do in UP. Refer to the keyboard template provided later in this section.

<Ctrl>+

b Move cursor up one line.

e Delete to end of sentence (command).

g Move cursor to end of sentence (command).

h Backspace and replaces character with space.

i Go to next tab position on line.

j Move cursor left.

k Move cursor right.

l Delete character.

m Insert mode: processes the entry when the cursor is at the end of a line; inserts a carriage return/line feed when the cursor is within the line.

n Move cursor down one line.

o Delete from cursor to end of word.

r Toggle between overtype and insert modes.

t Move cursor to beginning of command.

u Move cursor to next word.

w Insert single space.

x Exit TCL command and leaves just the TCL prompt.

y Move cursor back one word.

z z Undo last delete.

TCL Stack

When a command is entered at the TCL prompt, the system saves the command in the TCL-stack file of the dm account. In D^3, your stack is not terminal dependent. If you leave a terminal without logging off, another user can use the terminal under your user id. This causes the new user to "step-on" the your stack.

Changing any part of a TCL command in the stack, causes that stack entry to be moved to the top of the stack. This feature tends to keep the stack compact. However, the TCL stack does not have a maximum number of entries and can continue to grow indefinitely. Therefore, from time to time, the stack should be pruned either from TCL or by using the update command to modify the actual stack item (u dm,tcl-stack, user.name).

The following commands are used to move through a stack and to retrieve and run previously entered commands:

<Ctrl> +

a Searches for the entered string.

c p (cut and pop) Removes the current TCL command from its present position; places it at the top of the stack.

d Goes back to the previous command in the stack.

e If the cursor is on the first character, deletes the entry from the stack and displays the next command down the stack; otherwise deletes to the end of the command.

f Goes forward to the next command up in the stack.

p Moves a duplicate copy of the current TCL command at the current position in the stack to the top of the stack.

x Clears the displayed command from the screen and moves the pointer back to the top of the stack.

z Goes to the command at the top of the stack.

z a Same as <Ctrl>+a but searches to the top of the stack.

Pushing Levels

The execution of any command or program can be interrupted by pressing the <break> key. When a command or program is interrupted, the system stops execution and saves all parameters so that execution can be resumed exactly where it was interrupted. When a process is interrupted at the normal system level, the system prompts with two colons (::). At this point the command or program is said to be "pushed one level". Up to 16 levels can be pushed. The number of colons in the prompt indicates the number of levels pushed. The normal system level is 1.

To return to the previous level and continue execution of the process at that level, press <Ctrl>+m. To abort the process at the next lower level, use the end command. Refer to the entries level pushing and levels for more information.

Macros

A macro is a process that executes one or more TCL commands. Macros are stored in the master dictionary with the name of the macro as the item-id. The macro processor is provided for simple TCL procedures. In general, complex procedures should be written as FlashBASIC programs.

When a macro name is entered at TCL, it may be followed by any number of parameters. These parameters are added to the end of the first TCL command in attribute 2 as additional language elements and then passed for processing.

The first line of a macro must contain the character m (modify mode) or n (non-stop mode). Each subsequent line is considered a TCL command. If the m code is used in the first attribute of the macro, the TCL command is displayed so that changes can be made to it before it is executed. If n is used, the macro runs immediately.

A macro may be created with the Update processor or the create-macro verb. The create-macro verb takes the last statement entered at TCL and converts it to a macro. Enter the TCL statement to store as a macro and press <Ctrl>+m. When the cursor returns to TCL, enter:

create-macro macro.name

Note that create-macro sets attribute 1 of the macro to m. As long as attribute 1 equals an m, the macro name must be enclosed in double quotes when entered at TCL. Use the Update processor to replace the m with an n if immediate execution is desired. To create a macro using the Update processor, enter:

u md macro.name

For more information about using macros, refer to the entries macro and create-macro.

Menus

A menu provides a selection of processing choices. Menus are items in the master dictionary. The menu processor automatically formats the menu on the screen and you can then select one of the menu options for processing by entering the option number. The format of the menu item in the md is:

001 me {comments}

002 title

003 option 1

help 1

statement 1

004 option 2

help 2

statement 2.1

statement 2.2

...

Refer to the entry menus for more detailed information.

TCL Verbs

TCL verbs which operate exclusively on files and items use a consistent format to specify the file and items:

tcl-verb file.reference {item.list} { (options) }

The format elements are explained in the AQL section below.

AQL Verbs

AQL is a system-level information retrieval language that allows you to query your data base without writing complex programs. AQL uses TCL commands as verbs and displays the results on terminals or printers. AQL verbs operate on specified files and items based on optional criteria, specifications, modifiers, limiters, and options.

Often described as an ad-hoc data query language, the greatly expanded dictionary capabilities of D^3 offer the possibility of real nonprogrammer access to the data base. AQL, used in conjunction with the Update processor (UP), makes D^3 one of the most accessible data management system in existence.

Additional D^3 features enhance the already comprehensive query language. FlashBASIC calls from dictionaries are used for complex data calculations and output formatting. The ss (spread-sheet) connective allows the user to print out AQL reports in spread-sheet format. This is

achieved by adding the ss connective to the sort sentence and defining the desired range parameters. B-trees have increased the speed and performance of AQL.

AQL commands are entered at TCL and thus can be recalled, modified, or executed through the TCL stack. AQL sentences may also be stored and invoked through macros, menus, PROCs, and FlashBASIC.

An AQL statement has the following form:

verb file.reference {item.list} {selection criteria} {sort criteria} {output specifications} {print limiters} {modifiers} { (options) }

The verb and file.reference are required as operator and operand respectively. The verb must be the first word of the command. All other elements are optional and are used to modify either the operator, operand, or output. Selection criteria, sort criteria, output specifications, print limiters, and modifiers follow the item.list and may be in any order. Options, if used, must be placed last and must be preceded by a left parenthesis. The right parenthesis is optional.

Relational operators can be used with any of the elements of AQL sentences to allow exact specification of the conditions to be met. Refer to the entry relational operators for more information.

file.reference

Usually the name of a file in the md to which the user is currently logged. It can also be a synonym file name. The file name can be preceded by the literal dict to access the dictionary of the file instead of the data portion of the file. The default is data. In some cases, data may be specified to indicate only the data portion of the file.

To reference a file in another account or md from TCL, pathnames are used. The pathname may be used in place of the file.name in any TCL or AQL statement. A pathname may be entered in one of the following forms:

account.name,dict.name,file.name

account.name,file.name,

dict.name,file.name

item.list

Specifies one or more item-ids in the file defined by the associated file.reference. The item.list may be one or more explicit item-ids, a series of items separated by relational operators, an asterisk (*) to represent all the items in the file, or null.

If a select list is not active, a null item-id implies a new item for UP and all items for the other processors. Any command requiring a select list can obtain it from a previously selected list. (See the get-list, select, and sselect commands.) To cause a processor to use the select list, the item.list must be null. An item-id with the same name as a language element in either the md or the dictionary of the file must be enclosed in single quotes.

selection criteria

Limits the data by specifying criteria that must be met. Multiple criteria may be established to limit data selection to meeting a certain set of limitations. These sets are established by the logical relational connectives "and" or "or".

sort criteria

Connectives used to define the sort operation.

output specifications

Specifies the attributes to list. The selected attribute items or synonym labels are displayed in either a columnar or non-columnar format depending on the report width. The width of the report is the sum of the width of each attribute to be listed plus one blank separator between each attribute. If the width of the report does not exceed the page width as set by the term verb, a columnar format is generated.

The attributes for each item are displayed one under the other. If the requested output exceeds the page width, the column headings are listed in a non-columnar format down the side of the output with their respective values immediately to the right. In the non-columnar format, the column headings are listed only if there is a corresponding value. Item-ids are always displayed unless it is suppressed using the id-supp connective.

print limiters

Suppresses the listing of attributes within an item that do not meet specified limits.

modifiers

Control listing parameters such as double-spacing, control breaks, column totals, and suppression of item-ids, automatic headings, and default messages.

(options)

Used to tell the processor about special handling and tend to be processor specific. The options are single alpha characters and/or a numeric range specification as required by the specific processor. They are usually preceded by a left parenthesis. The right parenthesis is optional. When used, options must be the last element on the command line.

Wild Card Capability

Wild card characters may be used to select item-ids and attributes based on common characters. Wild cards can be used in selection criteria or complex item-lists as follows:

[(left bracket) Matches characters following the bracket. Ignores characters to the left of the bracket.

] (right bracket) Matches characters from the beginning of the string to the bracket. Ignores the characters to the right of the bracket.

^ (caret) Matches any character in the position occupied by the caret.

Retrieval of Items from Files

The AQL processor uses both the master dictionary and the file dictionary to determine the definition of the elements in the AQL sentence. The file pointer to the file dictionary and the connectives used in the sentence, for example, are found in the master dictionary. The file pointer to the data file and file-specific attribute definitions are found in the file dictionary. If an element is defined in both the master dictionary and the file dictionary, the definition in the file dictionary is used.

If the element is not found in either the master dictionary or the file dictionary, AQL creates a new element by concatenating the unknown element to a blank and the next element in the string. The processor attempts to look up this new element in first the file dictionary and then the master dictionary. If the new item-id is not found, an error message displays. The AQL

processor does not look up terms in the string that are enclosed in quotes, single quotes, or backslashes. These are assumed to be literals.

Default Output Specifications

In addition to explicitly listing attribute names as part of the AQL statement, there are three features that can be used to specify default output specifications. These specifications output the default attributes when attributes are not explicitly specified in the AQL statement and are listed below:

- The attribute names can be listed as a macro in the file-defining item.

- Default attribute items can be created.

- Temporary attribute items can be created.

Default Attribute Items

Default attribute items have numeric item-id starting with 1. These item-ids are used by AQL verbs as output specifications if no other output specifications are given. The numeric item-ids must be consecutive; that is, in order to have the third attribute list by default, attribute items 1 and 2 must exist, even if they are not needed for the listing. The attribute items for attributes that are not needed for listings can be given a d/code of x . For more information about default attribute items, refer to the entries default attribute items and default output specifications.

Temporary Attribute Items

Attribute items using special attribute names can be specified in an AQL sentence without actually existing in either the file dictionary or the master dictionary. The attribute name is of the form "Aac", where 'ac' is the attribute number. Temporary attribute items are created with a justification code (attribute 9) of lx (left justify and expand display field to fill report).

For example, even if neither the master dictionary nor the file dictionary for ent has an attribute-defining item a14, a statement such as "list inv a14" lists attribute 14 in the ent file where inv is the temporary attribute name.

Spooler

The D3 system Spooler controls all output that is sent to the printer. The term "Spooler" comes from the acronym SPOOL derived from Simultaneous Peripheral Output On-Line. Depending on the printer assignments and the status of the printer, the output may be printed immediately, sent to tape, placed in a queue, or placed in a hold file.

Most Spooler commands allow options, which sometimes have numeric arguments. To keep these options clear for the options interpreter, it is recommended that numeric options be separated with blanks as in the following example that assigns for hold file output to formqueue 1 with 3 copies:

:sp-assign 3 hsf1

Spooler options do not have to be enclosed in parentheses like options with other TCL commands. If numeric options are within parentheses, parameters outside of the parentheses will be ignored. Do not separate options with blanks.

The Spooler directs the items in the queue to the printer as the printer becomes available. The Spooler formats items in a hold file as if they were being output to the printer, but does not actually output them. The Spooler can be directed to output hold file items to the printer, to tape, or to a specified file. The following TCL commands are available to control the Spooler activity:

:startspooler assignfq list-ptr

listabs listpeqs listptr

sp-assign sp-close sp-edit

sp-kill sp-open sp-status

sp-tapeout startptr startspooler

stopptr

For more detailed information about these commands, refer to the individual entries for each command.

TCL Commands

The TCL commands are listed below. For a more detailed description of each command, refer to their individual entries.

```
!                   :absload            :bootstrap
:files              :reset-async        :scrub-ovf
:shutdown           :startspooler       :swd
:swe                :swx                :swz
:taskinit           =                   ?
abs-dump            abs.fid             absdump
account-maint       account-restore     account-save
add                 add-font            addbi
addd                addenda             addendum
addx                after               alarm
b/list              basic               basic-prot
beep                bformat             blist
block-print         bootstrap           break-key
brk-debug           brk-level           buf-map
buffers             bulletin.board      buffers.g
cal                 capt                capture-off
capture-on          case                case-file
cat                 catalog             cd
cf                  charge-to           charges
check-account       check-dx            check-files
check-sum           check-ws            checkfiles
chg-device          chksum              choose.term
cl                  clear-basic-locks   cleanpibs
clear-file          clear-index         clear-locks
clock               cls                 cmdu
coldstart           coldstart.log       color
comment             compare-list        compare
compile             compile-catalog     config
conv-case           converse             copy
copy-list           copydos             count
cp                  create              create-abs
create-account      create-bfile        create-file
create-index        create-macro        create-nqptrs
cross-index         cs                  ct
currency            date                db
dcd                 debug               decatalog
define-terminal     define-up           del-acc
delete              delete-account      delete-file
delete-index        detach-floppy       delete-list
detach-sct          dev-att             dev-det
df                  diag                dir
dir.pick            disc                disk-usage
diskcomp            diskcopy            display
div                 divd                divx
```

```
dl                  dm                  dos
dos.bridge          dos.shell           dos.video
download            dtr                 dtx
dump                ecc                 echo
ed                  edit                edit-list
el                  end                 env
environ             epson               esc-data
esc-level           esc-toggle          exchange
exec                exit                export
f-resize            fc                  fdisk
fid                 file-save           filecomp
find                fkeys               fl
flush               font-parms          format
frame-fault         free                fuser
get-list            get.pick            gl
group               halt-system         hash-test
help                hush                import
import.pick         indexer             init-abs
init-ovf            initovf             inputwait
inter               iselect             isselect
istat               item                k
kill                l                   ld
ldf                 legend              lerrs
lf                  lfd                 lfs
li                  link-pibdev         link-ws
list                list-abs            list-acc
list-commands       list-conn           list-device
list-dict           list-errors         list-file-stats
list-files          list-item           list-jobs
list-label          list-lines          list-lists
list-lock-queue     list-locks          list-logoffs
list-macros         list-menus          list-obj
list-pibs           list-system-errors  list-procs
list-ptr            list-restore-error  list-ports
list-users          list-verbs          listacc
listbi              listc               listconn
listdict            listf               listfiles
listfs              listprocs           listptr
listu               listusers           listverbs
ll                  lm                  load.mon
lock-frame          log-msg             log-status
logoff              logon               logto
loop                loop-on             lp
lq                  lre                 lu
maxusers            md-restore          message
mirror              mlist               mload
mmvideo             modem-off           modem-on
mono                monitor-status      move-file
msg                 mul                 muld
mulx                mverify             nframe-index
node                nselect             off
okidata             op                  overflow
ovf                 p                   pack
password            phantom-status      pc
pibstat             pick                pick-setup
pid                 pitch-compile       pitch-table
poke                povf                power-off
ppcp                prime               print-err
print-error         printronix          prio
prompt              psh                 psr
pverify             pxpcmd              qselect
r83.setup           reboot              rebuild-ovf
```

```
recover-fd          recover-item        reformat
rename-file         renumber            reset-port
reset-user          restore-accounts    ri
rmbi                rnf                 rtd
run                 run-list            s-dump
save                save-list           search
search-file         search-system       sel-restore
select              send-message        set-8mm
set-abs             set-batch           set-batchdly
set-baud            set-date-format     set-cmem
set-date            set-date-eur        set-break
set-date-std        set-device          set-dptr
set-esc             set-file            set-floppy
set-flush           set-func            set-half
set-imap            set-iomap           set-kbrd
set-keys            set-num-format      set-lptr
set-ovf-local       set-ovf-reserve     set-port
set-runaway-limit   set-sct             set-sct-dma
set-sym             set-shutdown-delay  set-sound
set-tape-type       set-term            set-time
set.lptr            set.time            setpib0
setport             setup-printer       setup.rtc
setup.sio           sh                  shell
shl                 shp-kill            shp-status
shpstat             shutdown            si
sl                  sleep               slice
sm                  sort                sort-item
sort-label          sort-list           sort-users
sortc               sortu               speller
sreformat           sselect             stack
start.rtc           start.ss            startlog
startsched          startshp            stat
status-port         steal-file          stoplog
stopsched           strip-source        sub
subd                subx                sum
system-coldstart    t-att               t-bck
t-bsf               t-bsr               t-chk
t-det               t-dump              t-eod
t-erase             t-fsf               t-fsr
t-fwd               t-load              t-rdlbl
t-read              t-ret               t-reten
t-rew               t-select            t-space
t-stat              t-status            t-unld
t-unload            t-verify            t-weof
t-wtlbl             ta                  tabs
tandem              tape-socket         tcl
tcl-hdr             term                term-type
termp               test-cursor         time
time-date           timedate            tlog-restore
to                  touch               trap
txlog               type                type-ahead
unix                unlink-pibdev       unlock-frame
unlock-group        unlock-item         unpack
update              update-abs-stamp
update-accounts     update-logging      update-md
update-prot         user-coldstart      useralarm
user-shutdown       verify-index        verify-abs
verify-system       vga.lcd             video.demo
what                where               which
which-line          who                 wlist
wselect             wsort               wsselect
x-ref               xcs                 xonoff
```

```
xref                xtd                 z
zh                  zhs
```

Connectives

Connectives are words in the master dictionary that are used to form the elements of AQL statements. They include relational operators and modifiers and are used to form sort and selection criteria and limit data to be processed by the verb with which they are used.

Relational operators are used to establish criteria based on the relationship of data to fixed values or other data. Relational operators would be used to select a range of zip code values within specified upper and lower limits. Refer to the entry relational operators for more information.

Other modifiers and options are listed below. For more information on each of these connectives, refer to their individual entries.

```
any                 before              break-on
by                  by-dsnd             by-exp
by-exp-dsnd         col-hdr-supp        dbl-spc
data                entry               det-supp            dict
duplicate           each                every
fill                footing             grand-total
hdr-supp            heading             header
id-prompt           if{each|every}      id-supp
if{no}              legend-supp         lptr
ni-supp             no                  nopage
only                or                  roll-on
sampling            spread-sheet        ss
supp                tape                tcl-supp
total               total-on            using
with                within              without
```

:

TCL prompt character.

@

indicates a D3 shell variable.

A D3 shell variable.name may contain any combination of alphabetic and numeric characters. The period and underscore characters are also valid.

Shell variables are pushed up and down all execute levels and persist until logoff. They are essentially the TCL analog of BASIC named common variables.

A D3 shell variable can be modified with the TCL command "set variable.name{=value}", and cleared with the command "unset variable.name". The current list of user variables can be displayed with the "penv" verb. The following user variables are examined by the system. Note that variables begining with the word "sys" are reserved for future system usage. The "$" character is optional, but is recommended for usage on all modifiable variables.

@$MESSAGES The contents of this variable are used to locate the system error messages. The format of this variable is the full path name of the desired alternate messages file. To allow the use of an alternate messages file, insert a line similar to the following, "set messages=dm,msg.file.name," as the very first line of the user macro. All subsequent error messages will use this alternate file until the user logs off the file.

@$PATH The contents of this variable are used to direct the system to default accounts for verbs, BASIC main programs, BASIC subroutines, and files. When the system fails to find one of these objects within the current master dictionary, it successively searches each of the account master dictionaries in this list. The list is of the following format: "account.name1,account.name2,account.name3...". Up to 8 account names may be listed. Note that there must NOT be a comma at the end of the list. When doing a "set" of this variable, the system pre-checks each of the account names. If an error is found, then the text of the list will be terminated with the "(ERR!)" string.

@$PROMPT The contents of this variable dictate the appearance of the TCL prompt. The system expects this variable to have the following format: "prefix,prompt1,prompt2". The "prefix" parameter is a textual indicator which is displayed before the prompt. The "prompt1" parameter is the prompt character to be used normally at TCL. The "prompt2" parameter is the prompt character to be used when a select list is active.

@$SYS.COMPILER The contents of this variable dictate the compatibility setting of the BASIC compiler. See the "$options" token for the BASIC compiler for more information about this.

Several variables have values which are dynamically computed at run-time. The currently supported list includes the following:

@AM	Attribute Mark
@FM	Attribute Mark (for licensee compatibility)
@VM	Value Mark
@SVM	SubValue Mark
@SM	SubValue Mark (for licensee compatibility)
@USER	User name
@WHO	User name (for licensee compatibility)
@ACCOUNT	Account
@LOGNAME	Account (for licensee compatibility)
@PIB	User Pib
@USERNO	PIB (for licensee compatibility)
@TIME	Current System Time
@DATE	Current System Date
@DAY	Current System Day of Month
@MONTH	Current System Month of Year
@YEAR	Current System 2-digit Year
@ID	Same as "*A0" (for licencee compatibility)
@SYSTEM.RETURN.CODE	Returns number of items processed by the most recent access statement.

If a shell variable occurs directly next to other text, the identifier (starting after the "@", but before the "$" if present) may be surrounded by the "{" and "}" characters.

The second version of the @ syntax executes the specified TCL command and replaces the @`...` sequence with the output of that command.

Substitution of TCL shell variables is active for all TCL commands including those issued from macros, paragraphs, interactive commands, and BASIC executes. Transitive expansions (those expansions whose values contain other variable definitions) are allowed except where noted below.

At TCL, several rules are in effect to prevent shell variables from interfering with previously existing code. If the "@" character appears within single quotes, then the corresponding variable is NOT expanded. Also, the sequence "@@" is expanded into "@". Finally, any shell variable identifiers which are not found in the current list are NOT expanded at TCL.

TCL shell variables are passed up and down between levels so they may be used freely across BASIC execute boundaries.

All shell variables are released and cleared when logging off of the system.

The conversion processor expands conversion values beginning with the "@" character to their respective values. Note that transitive expansions are not allowed in this case for performance reasons.

The menu processor expands all menu definitions based on the above mentioned rules for TCL.

Shell variables may be referenced and assigned within a BASIC program. Note that the TCL shell capability is not available from BASIC. Also, shell variables cannot be assigned directly by a complex BASIC statement (like "locate" for instance). They must be assigned with simple assignment statement.

Syntax

@{$}variable.name}

@`TCL.Command`

Example

```
The following sequence of commands shows some typical interraction
from TCL:
:set prompt=PROD,>,?
PROD>set myfile="dm,bp,"
PROD>who
70 john ba
PROD>penv
prompt=PROD,>,?
myfile=dm,bp,
PROD>select @$myfile sampling 5
[4042] 5 items selected out of 5 sampling items
PROD?list @$myfile heading "Report from @logname"
Report from ba
dm,bp,........
color
pick
set-imap
shp-kill
restore-accounts
[405] 5 items listed out of 5 items
PROD>display @system.return.code
5
PROD>display @who|@{time}xxx|@date
john|15:27:12xxx|17 May 1994
PROD>unset prompt
penv
myfile=dm,bp,
Shell variables can also be accessed from conversions.  Assume that a
```

```
file called junk has the following attribute-defining item:
DICT junk 'myattr' size = 45
dictionary-code   a
attribute-count   1
substitute-header
structure
output-conversion @$mycv
correlative
attribute-type    L
column-width      10
.....
Assume that the file contains one item called "itm1" which contains
the text "abc123". Notice how the conversion can be modified without
changing the attribute definition:
set mycv=mcu
list junk xx
Page   1    junk
junk..... xx.........
1         ABC123
[405] 1 items listed out of 1 items.
set mycv=mca
list junk xx
Page   1    junk
junk..... xx.........
1         abc
[405] 1 items listed out of 1 items.
Multi-lingual systems are facilitated by use of the "@$messages"
variable. Assume that a system exists with 30 English speaking users,
and 20 French speaking users. Assume that the standard English
messages are stored in "dm,messages" (which is the default), and that
the French ones are in "dm,messages,french". By default, all users
get their messages from "dm,messages,". To cause French speaking
users to use French error messages, it is a simple matter of
inserting the following line as the first thing in each French users'
user macro:
set messages="dm,messages,french"
```

dot stack

Describes the TCL dot stacker.

In D3, every unique command that is typed at the ":" (TCL) prompt is saved in a file on the "dm" account called "tcl-stack".

TCL stack entries may be recalled and edited using the dot stack commands:

.? Display help

! str Search stack for first entry containing 'str', display and execute

.L List stack

.L n List top n stack entries

.L m-n List stack entries m through n

.R Display top stack entry and allow modification

.R n Display stack entry n and allow modification

.R n/str1/str2 Replace str1 with str2 in stack entry n

.RU n/str1/str2 Replace all occurrences of str1 with str2 in stack entry n

.DE Delete top stack entry

.DE n Delete top n stack entries

.DE n/str Delete any of n stack entries containing 'str'

.X Execute top stack entry

.{X} n{,n} Execute stack entry n. X 'pops' entries to top of stack

.X name Execute macro created by .C command

.X file name Execute macro 'name' in file 'file'

.C n{,n} Create macro from specified stack entries

.CO n{,n} Create macro, overwriting

dummy restore

is a method of validating the restorability of a "save-type" (account-save, file-save, etc.) backup.

There are 3 methods to validate backup media:

1) The "t-chk" command, which looks for parity errors, but doesn't actually verify data.

2) Performing a "dummy restore" using the "sel-restore" with the "n" option without actually writing any items back to disk. This not only checks for parity errors, but also validates the "logical" format of the tape. If an unrestorable situation is encountered on the media, another backup can be made. See examples.

3) Using the "t-verify" command, which has the ability to not only do the same as steps 1 and 2, but also can compare each record from the media with its counterpart on disk to verify that they match. See "t-verify". This is the most reliable way to check tape integrity.

See the subject, "Tape Verify During Full Restore".

Example

```
:sel-restore md * (n
Restore from Full, incremental, transaction log (f/i/t):f
file#:99999
This will read every file in every account on the full save tape,
looking for file number 99999 and in the process read the entire
tape.
```

dummy save

performes the "save" command without actually writing to magnetic media.

By omitting the "t" option, nothing is written to a peripheral storage device.

This is one of the most useful mechanisms for determining if gfe's or other serious errors exist in the entire system, or in a particular account.

Example

```
:save (fd
This is the equivalent of a "full" dummy save (like a "file-save").
:save (fdi
This is the equivalent of an "account-save" (dummy).
```

itemlist*

used to indicate the choice of item-ids used within TCL2 commands. It may either be a single item-id, multiple item-ids separated by spaces, or "*" (asterisk), which indicates all items.

There is a valid "fourth" method of "itemlist*". This occurs when a list is already "active". The "active list" may be generated by processes such as "select", "sselect", "qselect", "get-list", "fl" ("form list"), etc.

When a list is active, the "itemlist*" is omitted from the (TCL2) verb:

select entity

17827971 items selected.

copy entity<cr>

If a specific list of item-ids (or "*") is provided while a list is active, the active list is ignored.

A list of TCL2 commands is provided under "see also" below.

In D3, if a valid TCL2 command is provided without a specific list of item-ids (or "*"), "all" items are implied. In other words, the "*" is optional.

Syntax

verb file.reference item-id

verb file.reference item-id item-id item-id ...

verb file.reference {*}

level pushing

a term used to describe the ability to interrupt a process, invoke a "new" TCL prompt, and execute any valid TCL command.

If a <return> is issued at the "new" level TCL prompt, control returns to the previous level, exactly where it left off.

It is possible to "logto" another account while at a "pushed" level. When a <return> is issued at the TCL prompt, the process automatically returns to the original account.

Any tape or peripheral storage devices attached to the process when logging to another account at another level remain attached in the new account.

options: TCL

discussion of the "p" (print) and "n" ("no pause") options, the only "standard" options available to every TCL command.

Options are special characters that alter the normal effect of a TCL command.

Options always appear at the end of a TCL command and must be preceded by a left parenthesis character. The right parenthesis is optional, as are "," (commas) between the parameters. This rule does not, however, apply to Spooler, dump, and where options.

Syntax

TCL.command ... (n

TCL.command ... (p

Options

n Activates the "nopause" function on output to the terminal.

p Directs output to the system printer, via the Spooler.

Example

```
copy dm,bp, term-type (p
block-print this is a test (n
```

paragraph

a macro language allowing execution of TCL statements, looping structures, data stacking, screen input/output, and branching.

A paragraph is an item stored in the current MD whose attribute 1 contains the word "PARAGRAPH". For compatibility with other vendors, "PA" and "T" are also accepted as paragraph indicators.

In its simplest incarnation, a paragraph is a list of TCL statements similar to a standard macro. However, paragraphs also allow other features which make it much more powerful.

LOOPING

Paragraphs allow simple loops by starting a block of commands with the "loop" command and ending that block with the "repeat" repeat. Note that such a loop will continue indefinitely unless terminated by a branch structure shown below.

DATA STACKING

As in MACRO, paragraphs can stack data by adding a second value after a command. Also, a seperate line can appear with the syntax "data {data to stack}". This works in a similar fashion to the BASIC "data" statement.

SCREEN INPUT/OUTPUT

It is possible to solicit input and display output in a paragraph. To do simple output, use the "display" statement with a string argument. To do simple input, embed the desired prompt within double greater than and less than signs. For example, the line "display <<Enter name>>" will prompt "Enter name>" and then print the name. To see the more complex versions of input, see the "<<" token.

PARAMETER MANIPULATION

Unlike standard macros, the parameters to the original TCL command line executing a paragraph are NOT directly passed to the first command. Instead, they are placed in a buffer for the paragraph programmer to access individually. To access the first parameter (word) in the original line (which is generally the name of the paragraph), use "<<c1>>". To access the second paramter, use "<<c2>>". To see more complex means of accessing parameters, see the "<<" token.

BRANCHING

For conditional branching, the IF-THEN-GO construct is used. Its syntax is "if [constant|<<input prompt>>] [conditional] [constant|<<input prompt>>] then go label." The conditional may be ">" or "g" for greater than, "<" or "l" for less than, "n" or "#" for not equal, or "=" or "e" for equal. The label at the end should be an alphanumeric label which appears elsewhere in the paragraph.

LABELS

Labels are used as the target of IF-THEN-GO statements and consist of an alphanumeric label name followed by a colon followed by AT LEAST ONE SPACE. Other paragraph commands may follow on the same line.

COMMENTS

Any line beginning with an asterisk is assumed to be a comment.

LINE CONTINUATION

Any line ending with the underscore character will be merged with the succeding line when processed by the paragraph processor. Note that leading spaces on the next line will be truncated.

Example

```
paragraph
* List a default file if not specified
if <<c2>> = "" then go default
list <<c2>> (<<Input options>>
if 1 = 1 then go done
default: list bp (<<Input options>>
done: * Done with paragraph
 This paragraph will list the default file of BP if no file is
specified.  When listing the file, it will first prompt for the
options.
```

tcl stack

describes the TCL command stacker.

In D3, every unique command that is typed at the ":" (TCL) prompt is saved in a file on the "dm" account called "tcl-stack".

TCL stack entries may be recalled and edited using the Update processor edit commands. The UP commands listed below are valid when editing items in the TCL-stack file.

Changing any part of a TCL command in the stack causes that stack entry to be moved to the top of the stack. This feature tends to keep the stack compact. The fact that only "unique" commands are saved also helps keep the stack compact. "Unique", in this context, means that there are never any duplicated commands in the stack. For instance, even if the "who" command had been used many times, it will only appear ONCE in the stack. Each time a command is found and re-executed, it is moved to the "top" of the stack.

There is no limit to the number of TCL-stack items, or to the number of attributes in each item. These items continue to grow indefinitely. Therefore, from time to time, the stack should be pruned either from TCL or by using UP to modify the actual stack item (u dm,tcl-stack, user-id).

A user's TCL-stack is not terminal dependent. It is user-id dependent. If a user leaves a terminal unattended, another user can use the terminal under the previous user's id. This causes the new user to "step-on" the previous user's stack.

Since the TCL-stack is an updated file, it is frequently locked and updated by the system. If two users share the same user-id, then while the first user types a TCL command, the second user is "locked-out" of the TCL stack, and their terminal will beep as long as the first user is still entering the command.

The stacker allows new commands even if the stack item is locked by another port.

It is possible to cut and paste when editing the TCL command stack item.

It is possible to stack data or additional commands under the TCL statement by using the <ctrl>-v command.

tcl verbs, format

operates exclusively on files and items use a consistent format to specify the file and items:

TCL-verb file.reference {item.list} {(options)}

tcl1 verbs

defines the class of verbs knows as TCL 1 verbs, one of the three distinct types of verbs in the D3 System. TCL1 verbs generally stand alone, meaning that they usually do not require a file.reference or an item-id list.

In D3, the "list-verbs" command produces an alphabetically organized list of all verbs in a given md, or the current md if one is not explicitly provided. Any verb with a "2" in the "A2" column is considered to be TCL1.

tcl2 verbs

one of the three distinct types of verbs in the D3 System.

TCL2 verbs all have the same syntax requirements, requiring the verb, a file.reference, and a list of item-id's (see "itemlist*").

In D3, the "list-verbs" command produces an alphabetically organized list of all verbs in a given md, or the current md if one is not explicitly provided. Any verb with a "3" in the "A2" column is considered to be TCL2.

verb

used to describe a process which may be invoked from the TCL prompt. For example "who" is a valid "verb".

"command" and "verb" are essentially the same thing when used in reference to TCL. For example, "who" is both a "verb" as well as a "command". Not all "verbs" are "verbs" by definition, but they are all commands.

verb classes

divide TCL commands into three different categories according to how they are invoked:

"TCL1" verbs typically require only the verb and a <return>. Most verbs, however, allow options. The common denominator to TCL1 verbs is that they do not affect files, with the exception of the "create-file", "clear- file", "steal-file", "move-file", "rename-file" and "delete-file" verbs.

Some examples of TCL1 verbs include "who", "time" and "ovf".

"TCL2" verbs require a filename, and usually a list of one or more item-ids. The only time that they do not require a list of item-ids is when a list is active. "Active lists" are created by verbs such as "select", "sselect", "qselect", and "get-list".

Examples of TCL2 verbs:

u entity 1000

In this example, an item called "100" is requested from the file called "entity".

u entity 1000 1001 1002

This example requests multiple items from the file. In TCL2 commands, each item-id is separated from the next by a space.

u entity *

In TCL2 commands, "*" (asterisk) is a special symbol used to request "all" items in a file.

:select entity = '100]'

1 items selected.

:u entity<return>

In this final form, a select list is built with the "select" verb. Note that the item-id list does not have to be specified.

AQL verbs: The third, and final class of verbs are AQL verbs.

These are, by far, the most sophisticated of all verbs and require at minimum a verb and a filename.

Examples of AQL commands:

sort customers by name name address csz lptr

sselect invoices with no payment.amount by date

sselect orders by ship.date

_

indicates that the current line should be merged with the succeding one when processed in a paragraph.

n

"nopage" option affects only output to the terminal, preventing the output from pausing at the end of each (terminal) page of output.

Syntax

TCL.command ... (n

Example

```
list md (n
```

p

directs output to the system printer, via the Spooler.

The output is directed to the printer most recently designated with an "sp-assign" verb.

Syntax

TCL.command ... (p

! (Unix)

creates a Unix shell and executes any Unix command.

"Unix.command" is any Unix command. A shell is created and the command submitted to it. If "Unix.command" is omitted, a 'shell is pushed'. D3 messages are held until "Unix.command" is complete. If more than one message is received, only the last one is displayed when returning to TCL.

Each time the "!" or "sh" command is invoked, a new shell is "pushed" on "top" of TCL. Control is transferred to this shell. "<ctrl>+d" returns control to TCL.

Note that the form "!Unix.command" does not require a space after the "!".

Macros in the master dictionary make almost all Unix commands available from D3. The following are a few useful tips on using this facility:

Avoid Creating an Intermediate Shell Process

The "sh" command uses the system(3S) Unix library call to execute the Unix command. This command creates a new process which executes the default shell /bin/sh and passes the string to this new shell. In turn, this new process creates another shell process which does an exec(2) of the Unix command. The intermediate process (the one created by system(3S) is normally useless. The following macro structure avoids the creation of a third process:

01 n

02 !exec command

The "exec shell command" replaces the second process.

Environment Variables:

Upon creation, a shell inherits environment variables from the D3 process. It is possible to change any variable in the macro itself. Consider the following example:

myshell

01 n

02 !cd /usr/mydir; TERM=ibm3151; export TERM; exec sh

This macro changes the default directory, sets a new (unix) term type and finally exec a shell. By typing 'env' in this new shell, the new setting will be shown. To permanently set environment variables, use the TCL "environ" command.

Syntax

!{Unix.command}

sh {Unix.command}

*****

implements a comment in a paragraph

Syntax

* comment

:apmenu

enhanced menu processor with user friendly features.

The :APMENU will run all menu items that have 'me' at attribute 1.

Supported keys and their functions:

To move up: Up arrow, Ctrl-B

To move down: Down arrow, Ctrl-N, Space-bar

To move left: Left arrow, Ctrl-J

To move right: Right arrow, Ctrl-K

To get back to a previous level or exit to TCL: ESC, q, x

An option can also be highlighted and executed by entering the option number followed by a carriage return.

Syntax

:apmenu {md} item-id

:bootstrap

is invoked by the program "bootstrap".

:ent-link

creates links and associations between entities in the system.

Links establish the communication path between entities in a given system. This command is the basic tool to create all sorts of association, or group of entities linked together. The TCL commands "tandem", "mirror", "converse" etc... are only special cases of useful associations. The ":ent-link" command is normally 'packaged' in more simple commands. It is not intended to be used routinely, since it is fairly complex.

cent Optional Controlling Entity ID. If specified, an association is created. The controlling entity MUST be a device (serial, telnet, etc..)

string Optional Termination String. If not specified, ESC'x' is used. The string can be expressed as any combination of characters (eg c'Ab'), hexadecimal values (eg x'1b') or decimal values, up to 4 characters, separated by commas (eg x'1b',c'x'). The string is associated to the controlling entity. When the controlling entity receives the termination string on its standard input, it terminates the association.

code Optional Association Code. This code is a decimal number from 1 to 255, which is given to the association as a mean of simple identification. The legal values are defined in the include "dm,bp,includes qcb.inc".

went Optional Wake up Entity ID. This entity must be a valid D^3 process entity ID (eg P12). If specified, the entity (possibly the current process) is deactivated when the association is successfully created. When the association is terminated, the entity is awaken. This option is useful to wait for the termination of an association.

mid Optional Master Association ID. If specified, the association is attached to the Master Association as a 'sub-association'. If the master association terminates, all the sub-associations are also terminated.

t Type of a link. This one letter code is 'o' for an output link, and 'f' for a forwarding link.

eorg Entity ID of the origin of a link.

edest Entity ID of the destination link.

Links to be created in the association are specified by a series of pairs of entities, eg. 'o[p12,s12]. When the association is terminated, the links are restored to their initial state. Links enclosed in the 'undo list', eg. 'u[o[o12,BB f[p12,NULL]]' are created when the association is terminated.

Syntax

:ent-link {c[cent] {s[string]} {a[code]} {w[went]} {m[mid]}} {t[eorg,edest] { ... }} {u[{t[eorg,edest] { ... }}]}

Example

```
 :ent-link c[s0] o[p0,p1]
  Create a pipe between the the D3 process 0 and the D3 process
1. After this command is executed, data printed by the D3 process 0
is sent as input to the D3 process 1. The serial device 0, normally
associated to the D3 process 0, is the controlling entity.
Therefore, hitting ESC and 'x' on the terminal associated to the D3
process 0 will terminate the pipe.
```

```
 :ent-link o[p1,bb]
  Sent all output of the D3 process 1 to the byte bucket. This is a
way of suppressing all output from a process. Note there is no
controlling entity, therefore the link will be terminated only by
explicitly linking  the output op 'p1' to something else.
```

:ent-list

lists all the entities currently existing in the system, with their main characteristics.

This commands displays:

Entity Id:

The entity Id in hexadecimal.

Lock ID:

The entity Id of the entity which has locked this entity. Entity locks are asserted only for a short instant, to prevent concurrent accesses.

Att ID:

The entity Id of the entity which has attached this entity. Entity attachment is performed by the TCL command "dev-att".

Nb Lnk:

Number of links, output or forwarding, pointing to the entity.

Fwd Link:

Entity Id of the forwarding link.

Out Link:

Entity Id of the output link.

Asociation ID:

Unique association Id, in decimal.

Association Code:

Association code type, in decimal.

Description:

Short note about the current status of the entity: unlinked, in tandem, in converse, etc...

Syntax

:ent-list {(options}

Options

H Supress header.

P Printer output.

:ent-mon

monitors the data sent to an entity in the system.

"id" is the list of entity IDs to be monitored. For example 'p12' is the D^3 process 12, 's45' is the serial port number 45, etc... See the section 'entity' for the structure of the entity IDs.

All data sent to the specified entity is also displayed on the m onitor screen, it is sent to the monitoring terminal. By entering a '~' (tilde) on the current terminal, ":ent-mon" enters a

Monitor Command Mode, where the user can enter either a TCL command, executed on the current process, or a Monitor command:

continue Exit Monitor Command Mode and return to Monitor mode. Or simply type <cr> at the "Monitor:" prompt.

esc Change the escape character to enter Command Mode from the Monitor Mode. The new escape character can be specified in decimal (eg 65), hexadecimal prefixed by a period (eg .41), or as a character between single quotes (eg 'A').

exit Terminate Monitor.

help Display a help.

mode Specify the display mode. The mode is specified by "normal" (data is displayed as is), "hex" (each character is displayed as two hexadecimal characters) or "text" (printable characters are displayed normally, D^3 system delimiters are displayed by their usual representation and other characters are displayed as dots). Without argument, the current mode is displayed.

search Specify a list of characters to be searched in the incoming data stream. A character can be specified in decimal (eg 65), hexadecimal prefixed by a period (eg .41), or as one or more characters between quotes (eg c'ABC'). More than one character can be specified by separating them with commas (eg .0d,.0a,'$01'). Any character found in the search list will be highlighted on the display. Without any argument, the search is deactivated.

This command requires a SYS2 privilege.

Syntax

:ent-mon id{,id{,..}} {(option}

Options

x Stop Monitoring the specified entities. This option can be used to terminate the monitor if the process doing the monitor was logged off, for example.

Example

```
:ent-mon p1,s2
  Monitor all data sent to Process 1 and Serial port 2.
```

:ent-status

displays the status of an entity, device or PIB, on the system.

"id" is the entity id to examine. See "entity" for a definition of the format of an entity id.

The following information is displayed for all entity types:

Entity ..... s15 Entity id ..... x'0003000F'

Name ..... 0000 Version 00.00

Lock owner ..... Attached by ....

Linked to ..... P15 Forwarding to ..

Association id . 0 Association code 0

I/O Buffer size 184 Nb of writes ... 0

Nb of reads .... 0 Nb of Errors ... 0

"Entity"

Entity name.

"Entity id"

Entity id in hexadecimal.

"Name"

Optional driver code name. Currently 0.

"Version"

Optional driver version.

"Lock owner"

Entity id of the entity which has this entity locked. Entity locks are set only temporarily to perform some special functions on a device. This field should normally be null. ":reset-async" resets this lock.

"Attached by"

Entity id of the entity which has attached this entity by performing a "dev-att" command.

"Linked to"

Entity id of the output link, where normal data is sent to. If empty, the entity has been 'unlinked' by the command "unlink-pibdev". Links are created by "link-pibdev", "tandem", "mirror", "converse", ...

"Forwarding to"

Entity id of the forwarding link, where data received by an entity is forwarded to. This field is normally set by commands like "tandem", ":ent-mon".

"Association id"

Unique integer representing an association, or set of links between entities in the system. "tandem" is an example of association where several devices and processes cooperate in an association to perform a function.

"Association code"

Type of the association. This code is used by "list-tandems", for example, to identify the association.

"I/O buffer size"

Size in bytes of the internal buffers.

"Nb of writes, reads, errors"

Statistics. Currently unused.

The following information is displayed for serial entities (some fields may be implementation specific):

Controller addr. 00000000 Device addr 00000000

Interrupt ..... 00000000 Baud rate..... 9600

Parity ..... 0 Stop bits 1

Bits/char ..... 8 Protocol 0

DTR status 00000000

VINTR ..... 0000001F VQUIT 000000FF

CLOCAL ..... 00000000 Device ..... /dev/pts/10

Syntax

:ent-status id {(option}

Options

V Verbose. Without this option, only raw hexadecimal data is displayed.

Example

```
:ent-status p15 (v
   Display the status of the PIB 15.
:ent-status s15 (v
   Display the status of the serial device 15.
```

:files

initiates a full restore as an alternative to reloading from the "options" prompt.

The proper media type must be indicated with the appropriate "set-" command prior to using this command.

First, the system goes into single user mode as if "maxusers (s" was entered. Then, each line is wrapped up as if a "shutdown" command was entered. All workspace is released and the overflow table is cleared. The user is prompted for reallocation.

Do you wish to disable file reallocation (y/<n>)?:

This question can be bypassed by using either ":files (n" or ":files (r".

Then, you are prompted to insert the media.

Load reel #1 and press return:

This prompt can be bypassed by using ":files (l".

Syntax

:files {(options}

Options

l Skip the "Load reel #1 and press return" message.

n Skip the "Do you wish to disable file reallocation" message answering "yes" so that files are NOT resized.

r Skip "Do you wish to disable file reallocation" message answering "no" so that files are resized.

z Skip Incremental and Transaction log prompts answering "n" to each which allows generating indexes to automatically take place.

Example

```
set-sct
block size: 16384
:files (nlz
All data will be lost, continue (y/n):
Wrapping up process on line 1
Wrapping up process on line 2
Wrapping up process on line 3
Building overflow list....
Diagnostics.. sucessfully completed
```

```
Block size: 16384
1 mds
1 mds > dm
1 mds > devices
1 mds > devices > devices
Generating indexes for "dm > devices > devices"
14:51:32 10 Oct 1993
Starting Background Processes...
 Notice in the above example, the options chosen allow the bypassing
of all prompts except the first one.
```

:init-network

stops all local network servers and restarts them.

:kill-network

stops all local network servers.

:kill-node

stops all local network servers.

:reclaim-ovf

see "reclaim-ovf"

:reset-async

resets the current port or all ports to default value(s).

If the "port.number" is omitted, all ports are reset. The default values are: "converse", "tandem", and "mirror" are all "off", yet enabled for use.

This verb is usually only executed at boot time.

Syntax

:reset-async {port.number}

Example

```
:reset-async
All ports are reset to default values.
:reset-async 5
Resets port 5 to its default values.
```

:restart-node

stops all local network servers and restarts them.

:scrub-ovf

see "init-ovf".

:shutdown

is invoked by the "shutdown" verb.

This logs all processes off, kills all phantom jobs, and flushes all frames in memory before halting the system.

Example

```
:shutdown
Executing system shutdown procedure...
Do you wish to continue (y/n)? y
Terminating all print jobs...
```

```
Wrapping up process on line 3
Wrapping up process on line 4
Wrapping up process on line 5
Wrapping up process on line 6
Wrapping up current process on line 0
Pause for wrapup processing to complete...
Flushing memory, boot when disk is quiescent...
#
```

:start-network

starts all local network servers.

Note that a local host must have been previously defined using the "network-setup" command.

<<

gets user input or parameters in a paragraph.

The paragraph input function has many forms. All of them get input from one of several sources, and substitute that input in place of the original function on the paragraph line.

The first form of the paragraph input function, which begins with "c" followed by a number, indicates that the input function should be replaced by the nth parameter on the original TCL line executing the paragraph. The first parameter returns the paragraph name.

The next form, using the "i" indicator, is identical to the "C" form except that if the parameter is null, then the user will be prompted for the desired input.

To get input from a file, use the "f" indicator. If the item or file is not found, then the user will be prompted with the specified prompt.

To get input from the keyboard, use the last syntax. This has several portions:

a Normally, if a duplicate prompt appears twice in a paragraph, the user will NOT be prompted more than once for the input. Instead all further uses of the same prompt will remember the original input and return the result without user intervention. If the "a" option is used, then the user is "A"lways prompted even if the data is already present.

r This option causes the input statement to "R"epeat until the input is null. The final result will contain every previous input separated by spaces. This is useful for creating item lists.

After the options, the user may specify a direct cursor address or several pre-defined constants.

Finally, the input statement requires a prompt string followed by an optional mask. The mask has the same syntax as that used by the BASIC matches function.

If the user types "quit" at any paragraph input prompt, then the paragraph will terminate and return to TCL.

Syntax

<<cnumber>>

<<inumber,prompt.text>>

<<f(filename,itemname{,attribute{,value}}),prompt.text>>

<<{a|r}{,@([x,y|bell|clr]),prompt.text{,mask}>>

Example

```
paragraph
display File = <<i2,Enter File>>
```

```
list <<Enter File>> _
<<r,Items in <<Enter File>> (Return to terminate)>> a1 a3 a3

  This paragraph will extract the file name from the second word of
the original TCL statement.  If that parameter is not present, then
the paragraph prompts for the name.  Next the line "File =
{file.name}" is displayed.  After that, the user is prompted for
"Items in {filename} (Return to terminate)>"  Note how the
<<Enter File>> prompt is repeated, but the paragraph
processor substitutes the existing file name since it has already
been entered.  At the "Items..." prompt, the user may type several
item names followed by the (return) key.  After terminating the input
by entering a null entry, the paragraph will display a listing of
those items in the desired file.
```

abs-dump

dumps the abs file to magnetic media, creating a "bootable" tape.

Syntax

abs-dump file.reference {(options)}

Options

a Diskette drive 0 (A:). Not compatible with the "b" option.

b Diskette drive 1 (B:). Not compatible with the "a" option.

h High Density (1.2mb or 1.44mb floppy). Not compatible with the "s" option.

s Standard Density (360k or 720k floppy). Not compatible with the "h" option.

f Force link field to bottom of frame.

i Suppress display of the frame counter.

l Zero link field (Default for floppy).

r Allow re-specification of destination abs frame size.

w Swap header field sizes.

abs.fid

displays "abs" information.

"file.reference" is the name of an ABS file. If not specified, the boot abs "dm,abs," is implicitly used. The abs base fid, release date, implementation, patch level and release title are displayed.

Syntax

abs.fid {file.reference}

Example

```
abs.fid
abs          abs  abs   abs     abs         abs    abs    abs
filename     fid  size usage    date       implem  lvl  revision
---------- ------ ---- ----- ----------- -------  ---  --------
abs           18  1438  1332 23 Apr 1992  RS6000    2  5.2.0.A3
item 000=>    18             23 Apr 1992  RS6000    2  5.2.0.A3
```

absdump

See "abs-dump".

account-maint

invokes a menu to create, modify, or delete user accounts.

The account definition item, located in the "mds" file, may also be updated or deleted using the Update processor. This requires a sys2 privilege level.

Example

```
account-maint
account maintenance workscreen
1 Account name              -Lasalle
2 Synonym account/Modulo    -19
3 Privilege level           -SYS0
4 Password                  -none
5 Accounting update         -L
6 Retrieval codes
7 Update lock codes
enter line number to update
or functions ex-exit fi-file fd-delete
"Account name" is the new account name just entered.
"Synonym account" is the is the name of the original (source) account, if a
"q-pointer" is being created.
"Modulo" is the  modulo used in creating the file for this account ("D"
option). The default is 19.
"Privilege level" is the user privilege level. It may be "sys0", "sys1", or
"sys2".
"Password" is the password to be assigned to this account. The password is
displayed as '********'.  "de" must be entered to delete an existing password.
Passwords are case-sensitive and may be multi-valued.
NOTE: The password must be entered EXACTLY as specified, including upper and
lower case characters, as "B" and "b" are different characters.
"Accounting update" may be one of the following codes:
"l"  Does not update the "dm,acc," file. This is the default value.
"r"  Invokes the user's logon sequence whenever they type in 'end' while in
the debugger. (Another way of stating that it restarts the user's logon macro
or menu).
"u"  Updates the 'dm,acc,' (accounting history) file whenever the user logs on
or off.
"ur"  Combines the effect of the "u" and "r" options.
"Retrieval codes" are the codes granting this user retrieval permission to
specified files.
"Update lock codes" are the codes granting this user update permission for
specified files.
```

account-restore

restores an account from an "account-save", "file-save" or "md-save" magnetic media.

The account/master dictionary will not be restored if it already exists in the "mds" file.

The system prompts for the "ACCOUNT NAME ON TAPE". It only has to be entered if the account name on tape is different than the account name stated in the command line.

Additional changes saved on a transaction log and/or incremental save tape may also be applied, once the account/master dictionary has been restored. The default settings prompt for both of these.

An "account-restore" may be started from any tape of a multi-tape "file-save".

To save time in searching a tape, the "list-file-stats" listing may be consulted to determine the reel on which the data for the master dictionary starts. The listing is only generated if the "s" option is chosen while doing a save.

After the restore is complete, the system creates indices for files having an index specified in the file-defining item.

The appropriate media/device must be attached prior to issuing this command.

Syntax

account-restore account.name {(options)}

Options

frame.size (integer number). Designates the frame size of the source system which created "save" media. This option automatically affects modulos, which are either divided or multiplied by the given number, according to the data frame size on the target machine (see the "what" command for determining data frame size). If the target frame size is bigger than the source frame size, all modulos are divided by the ratio (target / source = ratio). For example, if the target system has 2K data frames and the source system had 500-byte frames, the modulos would all be divided by 4. This file resizing operation occurs after the reallocation parameter has been checked, unless file resizing has been disabled during the initial boot process.

a Indicates that the tape is already positioned at the data section of the correct account. This means that it is positioned PAST the label, right at the beginning of the data. For example, if the "t-fwd" command was executed to position to the right spot on the tape, a "t-rdlbl" command would have to be executed to be positioned at the beginning of the data.

c "compatible" tape. This is used on tapes produced on "licensee" systems (Ultimate, McDonnell Douglas, ADDS, GA, etc.) The restore process runs noticably slower.

d Data-sensitive files. All files except the md will be restored as ds-type files (case sensitive).

f Disables the "fast" restore. This should always be used when restoring non-D3 tapes.

n Disables file reallocation process.

s Suppresses messages on txlog tapes.

z Bypasses prompts for restore from incremental or transaction logs.

Example

```
account-restore production
account name on tape: <return>
This restores the "production" account from tape and calls it "production".
account-restore dm.old
account name on tape: dm
This restores the "dm" account from the tape and calls it "dm.old".
account-restore OLD.R83.PROD (500
account name on tape: PROD
This restores an account created on a system with 500 byte data frames. If the
target were to have a data frame size of 2K, for instance, all new modulos
will be divided by 4 to prevent files from being drastically oversized.
```

account-save

saves individual account(s) to the currently attached peripheral storage device.

"account.name" indicates the name of the account to save. If more than one account is specified, each must be separated by a space. If not specified, the system prompts for the account name. On D3, the accounts to save can be specified in a select list.

"tape.label" is an optional 48-character character string which is written into the tape label area at the beginning of each "reel" of the save.

Unless the "u" option is specified, the system does a complete account-save. However, "account-save" does not save a file with "dx" in attribute 1 of its file-defining item. Although "account-save" does save a file with "dy" in attribute 1 of its file-defining item, none of the items in the file are saved.

In order for the "incremental" account save to work, the system marks any item that is changed. This mark is normally cleared by a complete file-save; however, when using "account-save", the marks indicating changed items are not cleared.

If a file has a "dx" or "dy" code and an item in it is updated, that item is not marked as changed when it is filed; therefore, even if the "x" or "y" is removed before the "account-save" is run, that item is not saved by an "incremental" account save.

The "account-save" processor locks groups as it saves them. While the group is locked, no process may access any item in that group. Locking groups prevents spurious Group Format Error messages that would occur if another process changed an item while it was being saved. Up to four groups may be locked at one time. These groups are the ones containing: the mds dictionary pointer, the md pointer, the file dictionary pointer, and the group for the data currently being saved.

If the "save" is being done to Streaming Cartridge Tape (SCT), an explicit end-of-data sequence must be written since SCT cannot back up. Use either "t-weof" from TCL or the FlashBASIC "weof" statement to write the second eod needed to indicate the end-of-file. Otherwise, "account-restore" or "sel-restore" fails to see the end of the tape.

"account-save" is a FlashBASIC program which issues the "save" command with the "f", "t", "v" and "i" options. Any options available to the "save" command may be added to the command line.

Syntax

account-save {account.name} {account.name...} {(options)}

file-save tape label = {tape.label}

account name = account.name

Example

```
account-save
file-save tape label = Thursday
account name = dm
1   35  > dm
1   36  > dm > newac
1   37  > dm > newac > newac
...
The process displays the files being saved in the following form:
Reel#, File#,  > , account.name,  > , dict.name,  > ,
data.name
In all multi-reel operations, if the save detects end of media before
completing the save, it prompts with the message:
Load volume #n and type 'C'
label 08:00:00 16 Jan 1997 DATA ...   #_
When the next "reel" is inserted or mounted, "c" continues the
process, or "q" will stop it and return control to TCL.
The system does not check to determine if a different volume was
inserted or mounted.
```

acct1

subroutine used by the "create-account" process to set up default accounts.

acct2

subroutine used by the "create-account" process to set up default accounts.

acct3

subroutine used by the "create-account" process to set up default accounts.

add

adds two integer numbers together and displays the result as an integer number.

Syntax

add number number

addd

Invokes "add" command.

addx

adds two hexadecimal numbers and displays the result as a hexadecimal number.

Syntax

addx hex.number hex.number

Example

```
addx 1F 33
52
```

admin

contains systems administration functions and must be invoked from the "dm" account.

Example

```
admin
administrator menu
1)  file save/restore - menu
2)  system status - menu
3)  system control - menu
4)  account/user maint - menu
5)  off -  log off system
Enter number of choice, number? for Help, <RETURN> to exit menu, or verb:
```

admin.control

menu used to set the system date and time, logoff processes and send messages.

Example

```
admin.control
system control menu
1) set-time  ...  set system time-of-day
2) set-date  ...  set system date
3) logoff ......  logoff a specific process
4) msg .........  send message to a user
Enter number of choice, number? for Help, <RETURN> to exit menu, or verb:
```

admin.files

utilities for data backup and restore.

Example

```
admin.files
D3 Systems
1) set-device  ....  specify tape device
```

```
2) file-save  .....  perform a system backup
3) account-save  ..  backup a specific account
4) t-restore  .....  restore a specific account
5) save (fads  ....  verify file integrity (w/out tape i/o)
6) t-verify  ......  verify integrity of backup
7) f-resize  ......  resize files for next restore
8) <menu>  ......    tape configure/control - menu
Enter number of choice, number? for Help, <RETURN> to exit menu, or verb:
```

admin.maint

is used for creating and maintaining users and accounts.

Example

```
admin.maint
accounts/user administration menu
1) sort mds  .........  display account master dicts
2) sort users  .......  display valid users
3) u users  ..........  add/change/delete user
4) create-account  ...  create an account master dictionary
5) delete-account  ...  delete an account master dictionary
Enter number of choice, number? for Help, <RETURN> to exit menu, or verb:
```

admin.status

provides information on the users that are logged on to the system as well as reporting the status of the system lock table.

Example

```
system status menu
1) listu  ..........  list users logged on
2) time  ...........  display current time/date/day
3) list-locks (i  ..  list item locks
4) list-locks (g  ..  list group locks
5) ovf  ............  display available 'overflow' frames
6) buffers  ........  display memory buffer information
7) load.mon2  ......   monitor system 'cpu' load
8) listacc  ........   list system accounting statistics
9) what ............  system status 'at a glance'
Enter number of choice, number? for Help, <RETURN> to exit menu, or verb:
```

admin.tape

contains options to list the devices as well as commands to set up tape devices for usage.

admin.tape.setup

provides utilities to assign the various tape devices for use.

alarm (Unix)

schedules a task at a given time, or after a given delay in seconds, at a given date .

If "date" is not specified, the current date is used. If an asterisk (*) is used as a date, the command will be executed every day, at the specified time.

If "command" is not specified, the options apply to the "alarm" command itself. If command is specified, the options are part of the command.

If "command" starts with an asterisk (*), it is assumed to be simple text, which will be displayed on the user's terminal when the alarm expires.

The alarm signal handler must be set to "useralarm" by the "trap" command.

If no option or command is given, the Update processor is entered to add/change/review alarms.

If an input with time out is done after the alarm has been set, the alarm is suspended for the duration of the wait and restarted for whatever remaining time. Any command which should have been executed during the wait is postponed until the input with time out is done.

The alarm is run by the current process. If an event should have been executed while the process executes a TCL sleep command, it is cancelled. If an alarm occurs while the user is logged off, it is discarded.

The alarm can run as long as the process is not disconnected. To be sure, however, the "alarm (s" command, without any argument, should be included in the user macro to restart the alarm every time the user logs in.

If the alarm handler is not set properly, "alarm" will complain.

The trap handler can be set by the command:

:trap alrm run dm,bp, useralarm

Syntax

alarm {delay{,date} {command}} {(options}

alarm {delay{,*} {command}} {(options}

alarm {hh:mm{:ss}{,date} {command}} {(options}

alarm {hh:mm{:ss}{,*} {command}} {(options}

Options

c Cancels all currently scheduled tasks for the user.

l Lists currently scheduled tasks for the user.

s Sets current alarms. This option simply re-arms the alarm. It should be included in every user macro to start the alarm every time a user logs in.

Example

```
alarm 18:00 * File save is starting
Sets an alarm for 18:00 (6:00 p.m.), today. At this time, if the process is
still logged on, the text 'File save is starting' will be displayed on the
user's terminal.
alarm 120 off
In 120 seconds, the process is logged off. This command could be included in a
PROC, to make sure an input screen will not wait forever for an input. When
the input is complete, "alarm (c" will clear the time out.
alarm 09:00,12/03 display Meeting time.
At 9 o'clock, on December the 3rd, "Meeting time" displays.
alarm 17:00,* msg * File save in 30 minutes
Every day, at 17:00, the message command will be executed.
alarm
This enters the Update processor to edit the user alarm item.
alarm (l
List currently scheduled events.
alarm (s
This displays the next scheduled event, and starts the alarm. This command
should be included in the users' macros, so that the alarm is refreshed every
time a user logs in.
```

b/list

formats and outputs FlashBASIC source code listings.

It may also be used to format a FlashBASIC source program and write it back to the file from which it came.

Syntax

b/list file.reference itemlist* {(options}

Options

b Expands lines containing one "!" to a string of "*"'s. This only works if the "!" is all by itself on the line.

c Does not indent comment lines.

f Writes "formatted" program back to the file in which it was found.

i Uses "internal" defaults for formatting instead of "*blist" defaults in the "messages" file. See the "*blist" item listed under warnings.

l Suppresses display of line (attribute) numbers.

m Double spaces listing.

n No pause; suppresses pause at end of page on terminal display.

o Overwrites existing item (when used with "f" option); otherwise, it creates a new item-id of "id.blist", where 'id' is the original program name, followed by ".blist".

p Outputs to printer; does not update file.

r Forces all indentations to zero.

s Draws vertical structure lines. This "connects" statements that are generally "nested". For example, "for" statements are connected to their corresponding "next" statements with vertical lines composed of "|" characters.

t Outputs to crt, unless an "f" option is included.

u Updates item with unresolved format structure.

basic

invokes the FlashBASIC compiler, which translates the specified source code into executable object code with "case sensitivity" in effect. This causes the compiler to only accept upper case "keywords".

The "compile" verb allows "keywords" to be in upper or lower case.

For example, with case sensitivity in effect, the variables "TOTAL" and "total" are treated as two separate variables.

See "casing on" and "casing off" for handling case-sensitive input from the terminal.

Syntax

basic file.reference itemlist* {(options)}

Example

```
basic bp testprog
This TCL statement compiles the program "testprog" in the "bp" file.
The "object code" is placed in the dictionary of "bp".
```

basic-prot

toggles or displays the status of the FlashBASIC object protection scheme.

When enabled, this feature is global, thus it affects the entire system.

The D3 System shares FlashBASIC object code between all processes running a given program. While this vastly decreases memory requirements, it also opens the possibility of one user compiling a program while another user is concurrently running that same routine. This circumstance tends to produce random, unexplainable aborts that can be difficult to track on large systems. The protection scheme involves insulating running object code from updates caused by recompilation.

When protection is enabled, all previous revisions of FlashBASIC object code are kept in the same dictionary group, but are simply marked as "deleted". These "deleted" items are automatically cleared during the "save" process. (See the discussion of "dirty bits" in the topic on the "save" verb.) This allows compiling programs while they are currently being executed. Users running a given program when that program is compiled will continue to run the old version. If a user drops out of the program, to TCL, for example, and re-executes the program, the system will execute the newest object version.

Syntax

basic-prot {(option}

Options

f Toggles object protection off.

n Toggles object protection on.

basic-prot-off

turns off the FlashBASIC object protection scheme.

See "basic-prot" for an explanation of the protection scheme.

basic-prot-on

toggles on the FlashBASIC object protection mechanism.

See "basic-prot" for an explanation of the protection scheme.

bformat

formats a FlashBASIC source program and update the source file with the formatted item.

The action of bformat is identical to blist, except that the output is not printed, but is filed into the source file overwriting the original source program.

file-name is any FlashBASIC source file.

item-list is a list of program names, or an asterisk (*) for all items, or null if there is a select list active.

The bformat verb is table driven, like the BLIST verb. The table is stored in the messages file. The item-id of the control table is "BF" followed by a 4 character hexadecimal number. The table number is contained in line 4 of the verb definition. The default table number is 0, thus the default table item-id is "BF0000". The structure of the table is identical with the BLIST control table. The only default options are R for renumber, C for comment indent inhibit, and the numeric options for specifying the starting statement label number and increment. It is possible to specify the numeric options in the control table, and leave the R option as a run-time option. In this case, the numeric options need not be specified at run-time.

Syntax

bformat file.name {{item.list}|{*}} {(options}

Options

r Renumber statement labels and all references to the labels (GOTO, GOSUB, RETURN TO).

n1-n2 Used with the R option, n1 specifies the new beginning statement label number, n2 specifies the increment between statement labels. Both n1 and n2 default to 10.

The R option is useful for renumbering a source program to make it easier to follow. If the R option is used, then a line containing only an exclamation point (!) and a statement label may be used to change the current new statement label number to the label specified on that line. This is true as long as the current statement label number is greater than what would be the next number (previous label number plus increment). This is useful when specific labels make the program easier to follow, such as at the beginning of subroutines, etc. Note: Certain forms of the GOTO, GOSUB, and RETURN TO statements will not be renumbered. This occurs when there is no blank between the key word and the destination label, e.g. GOTO20 will not be renumbered, but GOTO 20 will. Also, a mixture of numeric and alpha-numeric statement labels following an ON GOTO or ON GOSUB will fail to renumber any numeric labels past the first alpha-numeric label.

blist

Formats and outputs FlashBASIC source code listings, according to the options chosen. It may also be used to format a FlashBASIC source program and write it back to the file from which it came.

The D3 version of blist can be customized. The blist verb is table driven. The table is stored in the messages file. The item-id of the control table is "BL" followed by a 4 character hexadecimal number. The table number is contained in line 4 of the verb definition. The default table number is 0, thus the default table item-id is "BL0000".

Line one of the control table defines the left margin, number of spaces to indent for each construct, and the default options. Each of these is separated by a comma. The default control table contains "7,3,(BS)" on line 1, designating a left margin of 7 (statement label area), 3 spaces to indent for each construct, and the options B and S are used by default.

Line two of the control table defines the key words used at the beginning of a statement which may begin or end a construct or which may begin a comment. Each value on this line contains the key word followed by a subvalue mark, followed by a type character (* for comment, ; for statement), followed by two digits. The first digit specifies the number of outdents for the statement, the second digit specifies the number of indents for the statement. Note that some statements may contain a value in each digit (such as WHILE, which will outdent the statement containing the WHILE, and then indent the following statements).

Line three of the control table defines the key words used at the end of a statement which may begin or end a construct. The format is the same as line two.

Syntax

blist file.reference itemlist* {(options}

Options

B Expand lines containing only an exclamation point (!) into a line of asterisks.

C Do not indent comment lines.

D Double space the listing.

L Suppress the editor line number.

N No wait at end of page (on CRT listings).

P Output to the spooler.

S Draw vertical structure lines.

If the S option is specified, then vertical lines will be drawn between the beginning of a structured construct and the end of the construct. This option is useful when constructs span pages or are nested many levels deep.

T Output to the terminal.

If a line is too long to fit into the page width (as setup with the TERM command), an attempt is made to fold the line at blanks, commas, or other delimiters. The folded portion of the line is indented to the same level as the original line. Any control characters encountered are displayed as periods.

The T option is intended for use when the P option has been included in the default options (see next section). The T option overrides the P option and should never be included in the default options (else output to the spooler will be impossible).

blkio

displays or sets the IO blocking factor, or number of frames read in one disk access during sequential file accesses. The setting is global for the whole system, and can be changed at any time.

The "blkio" command controls how many frames are read in one disk access by processes doing sequential file accesses, like the save or AQL processors.

Without any argument, the current setting is displayed. If the mechanism is active, the following message is displayed:

IO blocking factor set to n frames.

indicates that sequential access will read "n" frames at each disk access.

If the mechanism is disabled, the following message is displayed:

IO blocking disabled.

indicates that the IO blocking mechanism is not active, and that frames are read one at a time.

The argument "factor" is the number of frames to be read at each disk access. '1' or 'off' disables the grouping of reads. The maximum number of frames that can be read is 64. See the general description of "blocked IO" for a detailed discussion of the blocked IO mechanism.

The blocking factor can also be set by adding the following statement in the virtual machine configuration file:

blkio n

where 'n' is the blocking factor.

The TCL "buffers" command allows monitoring the effect of blocking IO, by the "Frame faults" counter.

Syntax

blkio {[factor|off]} {(option}

Options

q Quiet. Do not display anything while setting the blocking factor.

Example

```
blkio
IO blocking factor set to 2 frames.
   Display the current setting.
blkio 4 (q
   Set the blocking factor to 4 frames, suppressing the message.
blkio off
IO blocking disabled.
   Disable the mechanism.
```

block-print

produces a "banner" by converting characters to a large block format, made up of rows and columns of the character itself.

If the text contains too many characters, the text string is wrapped at a word boundary, if possible; otherwise, the text is wrapped after nine characters.

Text enclosed within quotes attempts to print on the same line without breaking on the space(s) between the words.

The characters in the text string are defined in the "dm,block-convert," file.

A character definition must consist of exactly nine attributes: (example of definition for 'H' character:)

```
id H
001 7      ;* # horiz cells: 1234567
002 C2,3,2 ;* raster line 1: HH   HH
003 C2,3,2 ;* raster line 2: HH   HH
004 C2,3,2 ;* raster line 3: HH   HH
005 C7     ;* raster line 4: HHHHHHH
006 C2,3,2 ;* raster line 5: HH   HH
007 C2,3,2 ;* raster line 6: HH   HH
008 C2,3,2 ;* raster line 7: HH   HH
009 B7     ;* raster line 8:
```

where:

"raster" is either:

Cnn {,nn {,nn...}}, or

Bnn {,nn {,nn...}}

such that:

Cnn indicates the id 'character' to be printed nn times.

Bnn indicates a blank to be printed nn times.

, indicates a switch from 'character' to 'blank', or vice-versa.

New items may be edited into the 'block-convert' file to create new languages, or even typefaces (such as script or italics). However, the height must remain at 9 characters (attributes).

Each word or passage is centered on the output line according to the width of the device to which it is being output. The device width is determined by the most recently executed "term" command.

Syntax

block-print text {(options}

Options

n No pause (nopage); suppresses pause at end of page on terminal display.

p Directs output to system printer, via the Spooler.

u Upper case option. If the banner character is lower case, the block character is made up of the equivalent upper case character.

Example

```
block-print "Eat At Joe's" Bar & Grill (p
Without the double quotes around the first part of the banner, this command
would fail with an "uneven number of delimiters" message. Secondly, the quotes
around "eat at joe's" passage force the passage to appear on the same output
line - side by side. Without the quotes around a passage, each word appears by
itself on the line.

block-print Hi
HH   HH   ii
HH   HH
HH   HH  iii
HHHHHHH   ii
HH   HH   ii
HH   HH   ii
HH   HH  iiii
```

bootstrap

reloads the D3 software from a currently executing system and then invokes the "coldstart" procedure.

This invokes a program which first kills all the Spooler jobs, then invokes the ":bootstrap" verb.

break-key

toggles the availability of the break key, or displays its present status when issued without options.

Syntax

break-key {(option}

Options

f Disables break key.

n Enables break key.

s Suppreses display of status message.

Example

```
break-key (f
[1323] Break key disabled.
break-key (n
[1322] Break key enabled.
```

break-key-off

disables the break key on the current port.

break-key-on

enables the break key on the current port.

Example

```
break-key-on
[1322] Break key enabled.
```

brk-debug

indicates that the break key will invoke the debugger on subsequent uses.

If the current process is a FlashBASIC program, the FlashBASIC debugger is invoked. In all other cases, the system (virtual) debugger is invoked. If the <break> key is set to push a level, the debugger may be entered with the "debug" or "de" command.

On some systems, when the <break> key is set to push a level, it is not possible to push a level while in the debugger. To push a level while in the debugger, enter a colon (:) followed by <return> or <enter>.

brk-level

causes the <break> key to push a level on subsequent uses.

Note: on some systems, when the <break> key is set to push a level, it is not possible to push a level while in the debugger. To push a level while in the debugger, enter a colon (:) followed by a <return> or <enter>.

It is not possible to push a level while at the TCL prompt. At least one character must be entered.

buf-map

displays a visual "map" of the filesystem, indicating which frames are currently being held in the RAM buffer area, and updates the display every few seconds.

The buffer map is laid out as a grid of frame addresses for the entire system (bottom left-hand corner is fid-1, and top right-hand corner is maxfid), and displays which fids are currently active in the buffers.

"minfid" and "maxfid" are the minimum and maximum FID numbers to be included in the result.

The buffer displays a number between 0 - 7, which indicates the buffer's "state". The values are shown in the upper right-hand corner as a reminder.

Press "x" to exit the display.

Syntax

buf-map {minfid-maxfid} {(options)}

Options

s Sample about 10% of the memory, rather than reading all buffers in memory. This option provides a quick snapshot of the memory usage, but is less accurate.

Example

```
buf-map
1-358398. Fid loaded: 8740. 8742 buffers. Type H for help.
- - - - - - - - - - - - - - - - - - - - - - - - - - - -    2265
1 :     5 : R
2 : W   6 : RW
3 : B   7 : RB                                          6
4 : BW  8 : RBW                                         6
 - - - - - - - - - - - - - - - - - - - - - - - - - - - -6- 1700
5
5                                                       5
```

```
5                                                       5
5                                                       5
5- - - - - - - - - - - - - - - - - - - - - - - - - - - -5- 1135
5                                                       5
5                                                       5
5                                                       5
5                                                       5
5- - - - - - - - - - - - - - - - - - - - - - - - - - - -5- 570
5                                                       56
5                              1 5                      56
5                             11 5                      16
1                      5    51111                      15
+---------+---------+---------+---------+---------+---------+
Range 1 to 358398. Hit X to stop.. 0000
States:
The diagram shows the different "states" of the buffers in memory by
a number as shown in the state table below.
With the "s" option, only a sample is taken.
Refer'cd Batch Write-req    State
0        0     0            1  Available
0        0     1            2
0        1     0            3  Available
0        1     1            4
1        0     0            5
1        0     1            6
1        1     0            7
1        1     1            8
During the sampling, type:
h  To display this screen.
s  To stop sampling and display results.
x  To exit the process.
```

buffers

displays a single "snapshot" of the status of memory-resident buffers. The "l" (loop) option repeats the display until voluntarily stopped.

The following elements are monitored by the buffers command:

Name Description

Activ Number of Process activations. Each disk read, keystroke, process wake up after a sleep increments this counter. When the number of frame faults is subtracted from this counter, this gives an idea of the volume of data entry.

Idle Idle time. Not supported on Unix Implementations

Fflt Frame faults. This counts the number of disk reads. If this number approaches the disk I/O bandwidth (as determined by the manufacturer), the system becomes disk-bound. Solutions range from increasing the memory allocated to D^3, to changing disk drives, to reorganizing the D^3 database on separate disk drives to increase parallelism ("disk striping").

Writes Disk Writes. All writes are normally done by the background flush process to update disk from dirty frames in memory. This number should remain at 30% - 50% of the number of frame faults. A higher number indicates a lot of updates, or else there may be insufficient memory allocated for the D^3 virtual machine.

Bfail Buffer Search Failures. This counter counts the number of failures to allocate a buffer in memory for a new frame. When non zero, this indicates that the memory is insufficient. This

counter should never be non zero. If it is, it indicates that the memory allocated to D3 is too small.

RqFull Disk Read Queue Full. Not supported on Unix Implementations

WqFull Disk Write Queue Full. This counter counts the number of instances where the flusher cannot keep up with the dirtying of frames. This is an indication that either the Flusher's write queue is too small for the given configuration (See the 'dwqnum' parameter in the "pick0" file), or that the memory allocated to D3 is too small.

DskErr Disk Errors.

Elapsd Elapsed time. This is the time in seconds between two sampling. For internal use only.

DblSrc Double Search. This counts the number of collisions between two or more processes frame faulting on the same frame at the same instant. A non zero counter should be exceptional.

Breuse Buffer Re-Use. This counts the number of instances where a memory buffer has been allocated by one process to read one FID and another process allocated the same buffer to contain another FID. A non zero counter should be exceptional.

Bcolls Batch Contentions/Collisions. This counts the number of collisions between a 'batch' process (i.e., a process which is disk intensive) and an 'interactive' process (i.e., a process which is keyboard input intensive). By default, D3 insures that interactive processes are given priority over batch processes in accessing certain resources. See the command 'set-batchdly' for more details.

Sem Semaphores Collisions. This counts the number of collisions between two processes trying to access a system wide internal table.

Vlocks Virtual Locks Failures. This counts the number of cases when a D3 process tried to assert a virtual lock and failed to acquire it because another process had it.

Blocks FlashBASIC Locks Failures. This counts the number of cases when a D3 process tried to assert a FlashBASIC lock and failed to acquire it because another process had it.

B0reg Buffers with no Virtual Registers attached. These are the buffers not currently attached for immediate reference. At any given time, very few buffers are actually attached. It is therefore normal that this number be almost equal to the total buffers in memory.

B1reg Buffers used by more than one process, but not used by its owner any more. These should be in very small number.

B2reg Buffers used exclusively by their owner. On RISC implementations, this situation allows better performance, because there is no conflict on these buffers. Normally, these buffers contain private workspace, data which is not shared, etc...

B>3reg Buffers used both by their owner and other processes. This number represent the number of pages actually shared among processes (data files) at any given time.

ww Write Required. This counts the number of buffers currently modified and not yet written to disk. This number should never go above 50% of the whole buffer pool. If it does, then the flusher ('set-flush') is probably not being activated enough.

IObusy Buffers being read from disk. This counts the number of pending disk reads. This counters is usually null, since the reads are too fast to be picked up.

Mlock Number of buffers memory locked. If the ABS section is locked, this number is at least equal to the ABS size. Also included, are the tape buffers when the tape is attached.

Ref Referenced Buffers. This counts the number of buffers which have been recently used.

WQ Write Queued. Number of buffers currently enqueued for write.

Tophsh Top of Hash. This number measures the quality of the hashing algorithm used to find a frame in memory. This number must be high (above 60% of the total buffers).

avail Available buffers. Number of buffers that are candidates for replacement. These are the buffers that nobody has been using recently. When this number drops below 10% of the total buffers, performance decreases significantly.

batch Batch Buffers. This is the Number of buffers used by batch processes. A high level (something approaching 50% of disk buffers) indicates that disk intensive activity is taking place by batch processes.

Syntax

buffers {(options}

Options

number (integer number). Sets delay between "snapshots". Default is 1/5 seconds.

c Clears history log before starting process. If not present, the history file is created as a "dx" file, meaning that it will not be saved by a file-save or account-save. This process appends to the "buffers.log" history file, unless the "c" option is specified.

h Creates a history file called "buffers.log", where statistical data about system performance may be stored.

s Displays system counters.

l Loops continuously. Operator is prompted to enter an "x" to stop or an "r" to redraw the screen. The default delay between activations is five seconds.

buffers.g

produces a graphic histogram of buffer usage for a range of dates and times.

"counter" is the attribute name in the "dm,buffers.log," file to examine. The available attribute names are:

0 time Times.

1 activ Activations.

2 idle Idle time.

3 fflt Frame faults.

4 writes Disk writes.

5 bfail Buffer search fails.

6 fqfull Read queue fulls.

7 wqfull Write queue fulls.

8 dskerr Disk errors.

9 elapsd Elapsed time.

21 ww Write requireds.

22 iobusy I/O busy.

23 mlock Memory locked.

24 ref Referenced.

25 wq Enqueued writes (write queues).

26 tophsh Top-of-hash.

27 avail Number of available buffers.

28 batch Batch.

Additional attributes available on a hosted Unix system are:

10 dblsrc Double-source.

11 breuse buffers re-used.

12 bsleep buffers sleeping.

13 sem semaphores.

"start.day" is the beginning day-of-the-week for the graph results. The available values are: sunday - saturday, or 0 - 6.

"end.day" is the ending day-of-the-week for the graph results. The available values are: sunday - saturday, or 0 - 6.

"*" Spans the entire week.

"start.time" is the beginning time-of-day for the graph results. The valid values are: 00:00:00 - 23:59:59.

"end.time" is the ending time-of-day for the graph results. The valid values are: 00:00:00 - 23:59:59.

The reports are histogram averages of the buffer values sampled over a period of time (from the "buffers" command). These reports can give the System Administrator a better idea of the workload of the D^3 system, and identify possible bottlenecks in the system's performance.

The activity log is stored in the file buffers.log with a data level per weekday (buffer.log,Monday, buffer.log,Tuesday, etc...). The file is created automatically when the buffers (H) command is used for the first time. Each data level is cleared when changing day, so that the file records a whole week of activity automatically. The itemid is the internal time on five digits.

The buffers command also creates automatically the dictionary attributes corresponding to the various counters, as shown in the table above. The attribute TIME displays the sampling time.

The attribute DESCRIPTION in the D pointers Monday, Tuesday etc... contains the date.

The file is created with a DX attribute.

Syntax

buffers.g counter {start.day{-end.day} {step {start.time-{end.time}}} {(option}

buffers.g counter {* {step {start.time- {end.time}}} {(option}

Options

g Displays a graph, rather than a histogram. With this option, the 'step' is automatically calculated. With this option, the results are averaged in an attempt to make the curve smoother.

p Direct output to printer.

Example

```
buffers.g sem  6
0      1      2      3      4      5      6...
+------+------+------+------+------+------+---
16:52:05
16:52:11
16:52:18 **************
16:52:25
16:52:31
16:52:38 *******
16:52:45 **********************
16:52:51
16:52:58
16:53:05 *******
16:53:12 *******
16:53:18
Number of samples   : 13
Total               : 14
Average per period  : 0.0002 / sec.
Max value           : 4
Max value /s        : 0.2857
Peak time           : 16:52:45
buffers.g writes tuesday (g
49.0 *
46.3 |                                          *
43.5 |                    *                    **
40.8 |      ***        *  *    ***
38.1 |     *   *     ** *    **          **
35.4 |    *     *          **     *  ***  *   **
32.7 |*  *       ***               **        *
29.9 |                                    **
27.2 | **
24.5 |
21.8 |
19.0 |
16.3 |
13.6 |
10.9 |
 8.2 |
 5.4 |
 2.7 |
 0/s +------+------+------+------+------+------+------
   09:09:26      09:11:46      09:14:06      09:16:26
buffers.g fflt * 01:00:00
List the number of frames faults (disk reads), for the whole week, by step of
one hour. In the example below, no history was recorded before Wednesday.
No log for Sunday
No log for Monday
No log for Tuesday
20Feb1991; Wednesday; Ctr=fflt, Step=01:00:00,
Range=00:00:00-23:59:59
0      8848   17696  26544  35392  44240  53088  61936
+------+------+------+------+------+------+------+------+----
10:59:28 ************************
11:59:54 ************************************************************
13:00:25 ************************************************************
14:00:52 ***********************************
15:01:18 *************************
16:01:49 ************************************************************
17:02:22 ***********************************
18:02:55 ******
19:03:32 *********************************************
20:04:08 *************************************************
```

```
21:04:43
22:05:21 ************************************************
23:05:55 ************
Number of samples   : 155
Total               : 622070
Average per period  : 7.1999 / sec.
Max value           : 88481
Peak time           : 13:00:25
buffers.g ww monday-friday 00:30 08:00-17:30 (p
List the percentage of write required write required buffers, for the week
days only, during business hours, by steps of 30 minutes.

Interpreting Results
After taking a significant sample, list the results with the buffers.g
command. The most useful parameters to survey are:
Fflt    This measures the number of frame faults. If this number approaches
the disk bandwidth as determined by the manufacturer, the system becomes disk
bound. Solutions range from increasing the memory allocated to D3, to changing
disks, or reorganizing the D3 data base on separate disks to increase
parallelism.
Writes  This number should stay about one third to a half of the number of
frame faults. It is not 'normal' for a system to do more writes than it reads,
under normal operation. If this is not the case, see the section 'Flusher
Adjustment' in this article.
Bfail   This number should never be non zero. If it is not the case, the
memory allocated to D3 is definitely too small.
WqFull  This number should not be non-zero 'too often'. If it is the case, and
if the number of writes is too big also, there is an abnormal rate of writes.
See the section 'Flusher Adjustment' in this article.
Bcolls  If this number becomes too high, this indicates that a lot of batch
jobs (like selects of big files) are done while other processes are doing data
entry. It is also an indicator that indeed interactive jobs are receiving
higher priority than batch processes. See the section 'Interactive - Batch
Processes' below.
ww      This number should never go above 50 % of the whole buffer pool. If
this is the case, the flusher is probably not activated often enough. See the
section 'Flushed Adjustment' below.
avail   This number should never go below 10% of the whole buffer pool. If
this is the case, memory must be increased or the flusher must be adjusted.
```

bulletin.board

prints bulletins from the "bulletin" file located in the "dm" account.

This may be included in the user "logon macro" list to automatically execute each time the user logs on.

The optional "new" parameter retrieves items which have not been previously read.

Syntax

bulletin.board {new}

Example

```
bulletin.board new
PICK SYSTEMS BULLETIN BOARD
Date: 08/28/97
Re  : Company Picnic
From: Pat Davis
The Company Picnic will be held this Saturday in the company warehouse due to
the impending thunderstorms. Unless the weather changes, the volleyball game
will be changed to shuffleboard.
HIT ANY KEY TO CONTINUE
```

caf

subroutine used by "create-account" and "delete-account".

cai

subroutine to establish default account information during "create-account".

cal

displays a calendar by running the Unix "cal" command.

"month" is the month of the year, and must be a number between 1 and 12, inclusive. If a month is specified, only the calendar for the given month is displayed.

"year" is the year for the calendar, in either two or four digits.

Syntax

cal {{month} year}

Example

```
cal
January 97
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

capt

enables or disables the capturing of every TCL command issued. When enabled, TCL commands are placed in the "captcl" file using the item-id "user-id*port.number".

The "captcl" file must be created, or a pointer must be established to it prior to using this verb.

"capt" is the verb invoked by the "capture-on" and "capture-off" commands.

Syntax

capt {(option}

Options

o Turns capturing off.

n Deletes item after disabling and is only valid with the "o" option.

Example

```
capt
Turns on TCL command capturing.
capt (o
Turns off TCL command capturing.
capt (on
Turns off capturing and deletes item from "captcl".
```

capture-off

disables the automatic capturing of TCL commands previously invoked with the "capture-on" command.

Options

s "Silent" operation. Suppresses message output.

x Does NOT delete the item from the "captcl" file.

Example

```
capture-off
Capturing turned off at 15:01:48  01 May 1992 on pib# 22
```

capture-on

enables the capturing of every TCL command issued.

The captured output is placed in the "captcl" file using the item-id "user-id*port.number" as the item-id.

The "captcl" file will be created, if it does not exist to using this verb.

Options

s "Silent" operation. Suppresses message output.

Example

```
This example illustrates how a series of TCL commands could be trapped to a
file:

capture-on
Capturing started on 14:53:11  01 May 1992 on pib# 22
term ,,,,,,79,52
sp-assign hs f11
sselect ap.doc by category
save-list ap.doc.by.category
get-list ap.doc.by.category
list ap.doc category token (p
capture-off (x
Capturing turned off on 14:56:18  01 May 1992 on pib# 22
The above sequence of commands is trapped in the item, "dm*22" in the
"captcl" file:

li captcl
Page   1      captcl                  17:58:21 01 May 1992
dm*22
001 term ,,,,,,79,52
002 sp-assign hs f11
003 sselect ad by category
004 save-list ap.doc.by.category
005 get-list ap.doc.by.category
006 list ap.doc token category (p
007 capture-off (x
[405] 1 items listed out of 1 items.
```

case

toggles case sensitivity on the current port, or displays its present condition when used without options.

For a system to run with the ability to be "case sensitive" and "case insensitive" among different users, the entire operating system must be structured so that the data base remains independent of case. All lookups for commands, item types, codes, etc. are "case insensitive" by default. This basically means that all dictionary lookups will be case insensitive.

Two macro versions of this verb, "case-on" and "case-off", are provided to direct comparisons within data files to be case sensitive or case insensitive.

The type of comparisons performed on data within attributes using AQL will be determined by the status of the case bit.

The default is case insensitive.

To remain case sensitive, add the "case-on" verb to the user logon macro.

Syntax

case {(option}

Options

f Turns off case sensitivity. Case insensitivity is the system default.

n Turns on case sensitivity.

s Suppresses output of status message.

Example

```
case
[1319] Case insensitive.
This form of the command displays the current state of case interpretation.
case (n
[1318] Case sensitive.
This form of the "case" command toggles case sensitivity "on".
case (f
[1319] Case insensitive.
This form of the "case" command toggles case sensitivity "off".
```

case-file

converts the case of items to either lower or upper case.

This invokes the program, "conv-case", and passes it the "e", "q", "u" and "o" options.

If the "file.reference" is omitted, the process prompts for it. If it is provided, the process prompts "DICTIONARY LEVEL?". Answering "y" to this causes the process to affect the dictionary of the given file.

As the items are "cased", the item-ids are displayed on the screen.

The "case-file" program resides in "dm,bp,".

Syntax

case-file {file.reference}

Example

```
case-file test
DICT LEVEL FILE Y,N?n
This converts all the item-ids in the "test" file to upper case.
```

case-off

disables case sensitivity on the current port.

Example

```
case-off
[1319] Case insensitive.
```

case-on

enables case sensitivity on the current port.

Example

```
case-on
[1318] Case sensitive.
```

cat

displays information about the size and location of FlashBASIC object code.

This display also shows the time and date when the program was last compiled.

Syntax

cat file.reference

Example

```
cat bp
Binary Item Listing For File jesbp       17:56:00 11 Mar 1992
bp........................... Ptr Fid   Length
helpgen                          01AF6D     0001
lc.help                          01AF6E     0001
callcp                           01AF70     0001
startest                         01AF72     0001
lockit                           01AF73     0001
matchtest                        01AF74     0001
fix.ad                           01AF75     0001
test.prestore                    01AF79     0001
form.feed                        01AF7A     0001
linefeed                         01AF7C     0001
```

catalog

creates a "verb" entry in the md of the current account, making the specified program executable directly from the TCL prompt.

If the cataloged program is a "mainline" program (not a subroutine or function), it can be executed by issuing its program.name at the TCL prompt.

"file.reference" names the file containing the program to catalog. In D3, if no file is specified, it defaults to "bp".

"itemlist*" names the previously compiled FlashBASIC program(s) to add to the master dictionary. If no itemlist* is specified, or if an asterisk (*) is used, all programs in the file are cataloged.

Once a program has been cataloged, it does not need to be cataloged again, even if it is changed and recompiled. (Issuing the catalog command from TCL is the same as using the <ctrl>+xc on a FlashBASIC program in UP.)

If there is currently a cataloged program with the same name, it is overwritten.

All external subroutines must be cataloged.

Activating a cataloged FlashBASIC program :

program.name {(options}

To activate a cataloged program, enter the program name at the TCL prompt character. All options available to the "run" command are allowed.

Syntax

catalog file.reference itemlist*

Options

o Overwrites any existing item except file-defining items and Q-pointers. A cataloged FlashBASIC program item in the master dictionary has its name specified in attribute 4. If an item already exists in the master dictionary that is not in FlashBASIC verb format and the "o"

option has not been specified, an error message displays indicating that the item is already on file and the program is not cataloged.

* See the "run" verb for additional options.

Example

```
catalog bp testprog
[244] 'testprog' cataloged.
The "TCL" command sentence to catalog the "testprog" object in the dictionary
of file "bp".
```

cd

changes the current working directory to the given "new.directory".

The new directory remains until the D3 process is disconnected (not logged off) or a new "cd" command is issued. If "new.directory" is omitted, the default home directory is selected.

Syntax

cd {new.directory}

Example

```
cd /tmp
```

cf

Invokes "create-file" command.

charge-to

changes the current accounting charges accumulation.

Subsequent usage statistics are accumulated to a new accounting entity composed of the user-id followed by the given "charge to" "text.string" in the form: "user-id*entity".

The charges are accumulated in the "dm,acc," file.

Syntax

charge-to text.string

Example

```
who
37 dm krb
charge-to mastercard
<  connect time= 121 mins.; CPU= 23747 units; lptr pages= 4  >
who
37 dm*mastercard krb
"connect time" is the number of minutes the account has been logged on.
"CPU" is the number of cpu units used by the account. CPU units vary from
system to system.
"lptr pages" is the number of pages sent to the printer (spooler).
```

charges

displays the current account usage statistics, including the total time logged on and cpu activity statistics.

Example

```
charges
< Connect Time=43 Mins.; CPU=356 Units; LPTR Pages=0  >
"connect time" is the number of minutes the account has been logged on.
"CPU" is the number of cpu units used by the account. CPU units vary from
system to system.
"lptr" pages is the number of pages sent to the Spooler.
```

check-account

performs a "dummy" save of an account by issuing the "save" command with the "f", "d" and "i" options.

check-dx

looks through every file on the system to find and identify those which have a "dx", "dy" or "dl" in attribute one of the "d-pointer". "dx" and "dy" files are not saved by the save processor.

After issuing the command, the following prompt appears:

Enter machine type (i.e. Production, Development) :

Any response may be provided at this prompt. The response is placed in the "heading" statement on output.

See "file-defining items" for information on "d-pointer" types and the effects they produce.

Options

l check for DL type files

x skip DX type files

y Check for DY type files

Example

```
check-dx
Enter machine type: Demo
Files on Demo With DX-type File Pointers
DM
file-of-files
file-of-files  <---- file pointer is DX'ed
1 file found with a DX file pointer.
```

check-file

performs a consistency check on a file similar to that done by the "save" processor.

Syntax

check-file file.reference

Options

g skips any gfe's found

check-files

performs a "dummy" save of an account.

check-resizing

completes re-hashing files in the process of being resized.

This command is executed in the system-coldstart macro and must not be removed as it corrects any file consistency problems which may exist after a system crash.

check-ws

validates the integrity of the workspace for a given port.

If the port.number is not specified, the process prompts for it.

If the process works successfully, two numbers display:

0000 00

The first number, "0000", is the workspace status. The second number, "00", is the level.

The following chart illustrates the workspace status tally (note that the bits are grouped in four sets of four bits):

abcd efgh ijkl mnop where:

a child.pcb dtally bad.

b pcb fid bad.

c pcb links bad.

d scb bad.

e dcb bad.

f db table bad.

g sws bad.

h ovf.stk bad.

i ibbeg fid bad.

j FlashBASIC workspace bad.

k fopen table bad.

l local group lock table bad.

m save.itm,list.active,list.active.2 bad.

n available, not used.

o pbuf begfid bad.

p available, unused.

Syntax

check-ws {port.number}

Example

```
check-ws 49
0040 01
The table above may be used to verify the output. In this example, the
workspace status tally is 0040, which means that the basic workspace is bad.
The "01" indicates that this process is at level "01".
```

chksum

calculates a hexadecimal "checksum" for either an entire file or a list of items in the file, and outputs the results to a file.

When an "item.list*" is specified, statistics are calculated only for the selected item-ids. If not specified, the entire file is used.

A new data section called "checksum" is created in the specified file. Each item in the "checksum" data section corresponds to an item in the requested file's data section and the first attribute of each item contains the hexadecimal checksum for the item.

Syntax

chksum file.reference itemlist*

Example

```
chksum temp
[404] 7 items selected out of 7 items.
```

```
runoff.box
i/nosave
basic.debugger.t
filename.sysprog-bp
op.col
op.macro.file
up.x
List the items in the checksum data-level.
li temp,checksum (c
runoff.box
001 3443F07
i/nosave
001 65D7D2B7B21
basic.debugger.t
001 644C5D32EE04
filename.sysprog-bp
001 3A4D745500C
op.col
001 6005F7623714
op.macro.file
001 5BF260B60902
up.x
001 14CC018FF4BE
[405] 7 items listed out of 7 items.
```

choose.term

selects one of a variety of choices of terminal type characteristics as an alternative to using the "term-type" program.

The "choose.term" program may be invoked from the user logon macro item in the "users" file.

Example

```
choose.term
dev name           dev type       term type
........           ........       .........
1) Att 605          att605             z
2) Console          ibm3151            m
3) Esprit           esprit             e
4) IBM 3151         ibm3151            i
5) Viewpoint        v                  v
6) VP-A2            vp-a2              a
7) Wyse 50          wy-50              w
Enter your numeric choice or term-type: w
Terminal definition 'wy-50' selected.
```

cl

Invokes "copy-list" command"

cleanpibs

clears tandem connects for all ports logged on before port 0 (zero). This is used at coldstart time to make sure that any tandem information leftover from a system crash does not get picked up.

clear-basic-locks

displays and clears all 64 FlashBASIC execution locks.

The verb invokes the "clear-locks" command and passes it a "b" option.

clear-file

clears the dictionary or data section of the specified file of all items and retains the primary file space area.

Clears data from the file specified by "file.reference". The file is only cleared, not deleted. The file retains its name, base, and modulo for both dictionary and data sections.

To clear the dictionary section, "dict" must be specified before the "file.reference". If "dict" is specified, file-defining items ("d-pointers") are not cleared.

If no "level" is specified, the default is "data".

Syntax

clear-file {(}{data} file.reference{, file.reference}) {(option}

clear-file {(}dict file.reference) {(option}

Options

c "Scrubs" each frame of the file's primary space, changing every byte in the frame to x'ff'.

n No logging. When this option is active, no log of the clear-file is placed in the file-of-files item. This can increase performance significantly on temporary files which are cleared frequently.

Example

```
clear-file data old.invoices
Removes all items from the data level of the "old.invoices".
clear-file dict invoice.history
Removes all items except "d-pointers" to data sections from the
dictionary of the "invoice.history" file.
clear-file data invoices,archive
Clears the "archive" data section located in the dictionary of the
"invoices" file.
```

clear-index

clears the specified b-tree index expression in the specified file, leaving only a null root.

Indices may be cleared indiviually, or all at once by using an "*" as the index.expression.

Syntax

clear-index file.reference {index.expression}

Example

```
clear-index entity a1
This clears the index on attribute 1 of the entity file.
clear-index invoices *
This clears ALL indices from the "invoices" file.
```

clear-jobs

selects items with the status "logoff" or "completed", and deletes them from the "dm,jobs," file.

Syntax

clear-jobs

clear-locks

clears all locks, or specific locks according to the option(s) provided.

When used with the "i" option, and optional host may be specified to release all item locks residing on a remote server.

Syntax

clear-locks {host} {(options}

Options

b Clears all FlashBASIC locks.

f{fid} Clears all group and item locks for a specific frame-id, specified as a decimal number. Note: The "f" option indicates that the number immediately folowing it is a frame-id. Without the "f" option, the number is interpreted as a port number.

g{f{fid}} Clears all group locks and ignores item locks for a specific frame-id, specified as a decimal number. See notes under "f" option.

i{f{fid}} Clears all item locks, and ignores group locks, for a specific frame-id, specified as a decimal number. See notes under "f" option.

o{f{fid}} Clears all group read-only locks for a specific frame-id, specified as a decimal number. See notes under "f" option.

q Clears all Spooler locks.

r{f{fid}} Clears all group retrieval locks for a specific frame-id, specified as a decimal number. See notes under "f" option.

u{f{fid}} Clears all group update locks for a specific frame-id, specified as a decimal number. See notes under "f" option.

s Clears all system locks.

Example

```
clear-locks (g3
This clears all group locks for port 3.
clear-locks (gf12345
This clears all group locks for decimal frame 12345.
```

cls

clears the screen, leaving the cursor at the colon prompt at the bottom of the screen.

If level-pushing is in effect, the number of colons appropriate to the level are redisplayed.

If the screen does not clear, verify that the "term type" parameter corresponds to the "emulation" on the terminal.

cmdu

produces a report of the last TCL command issued by every active user on the system.

Syntax

cmdu {(options}

Example

```
cmdu
Page   1 dm,pibs,       10:23:40 19 Mar 1992
pib# location....... user..... md.. last.cmd
27   Dick Pick       dp        ba   lu
60   Andy Meyers     am        qa   m
41   Sam Smith       ss        ba   main
33   Lisa Jones      lisa      ba   calls
45   Steve Johnson   sj        ba   termw
38   Fred Flint      ff        qa   to qa
```

coldstart

initiates the sequence of activities that must be performed each time the system is powered on.

This is the last step in the system boot sequence. Coldstart sets (or prompts for) the time and date, starts printer(s), starts the Spooler subsystem and phantom ports, and resets the accounting history file.

Any site-specific "coldstart" processes should be placed into an item called "user-coldstart" in the md of the "dm" account.

coldstart.log

records the type of shutdown taking place into the "errors" file.

comment

performs cursor control and screen display funtions similar to the "t" command in PROC. Its primary funtion is to add screen control capabilities to macro funtions.

Parameters are passed to the "comment" function as discrete units surrounded by parentheses. The following types of text are supported:

(column,row) "x,y" (column,row) positioning

(-n) special cursor control features

(xn) prints the characters whose numeric value is n

(string) prints the string of characters at the current cursor position

Several text segments may be entered on a single "comment" command line, but each must be surrounded by a single pair of parentheses.

The optional "+" at the end of a line suppresses the automatic carriage return and linefeed.

Syntax

comment (text){(text2)}{(...)}{+}

Example

```
Example to clear a screen:
clear
001 n clear-screen macro
002 comment (-1)
A sample "logon" macro:
001 n logon macro
002 comment (-1)(-13)(User Logon)(0,2)(User:)(-14)(20,2)+
003 who
004 comment (0,3)(-13)(Time-Date:)(-14)(20,3)+
005 time
006 comment (0,5)(-13)(Spooler Assignment)(-14)
007 sp-assign ?
008 comment (0,10)(-13)(Terminal Assignment)(-14)
009 term
This example of a "logon" macro clears the screen and displays the users
name/account, the current date and time, and the current terminal and printer
assignment parameters.
```

compare

compares items line-by-line to find which lines have been inserted, deleted or changed.

"compare" produces a comparative listing of each item in the specified itemlist, indicating where differences occur between the source and target item{s}.

Status characters:

"d" Indicates a deleted line.

"i" Indicates an inserted line.

Syntax

compare file.reference itemlist* {(options)}

with:{(file.reference} itemlist

Options

a Compares all elements on each attribute. The default is the first three elements, delimited with blanks (assembler source general format).

b Compares entire first value of attribute. If not specified, only the first three words are compared. This option is used to compare assembly object code.

c Combines output into one-column format, rather than two-column "split-screen" format.

d Does not compare object code addresses. This is intended for compiled assembly language programs.

f Issues a form-feed between each item; Starts display of each item on a new page.

i Suppresses the message, "[409] item not on file".

n Activates nopage function on output to the terminal.

o Compares entire attribute, except for the first value. This is used to compare assembly object code by comparing the entire attribute except for the first value and first subvalue within each value.

p Directs output to system printer, via the Spooler.

s Suppresses display of item-ids; This is available only with the z option.

z Lists only lines with differences and displays item-ids only of identical items.

Example

```
compare entity 12345 (zsa
with: (entity,archive 12345
compare bp count (a)
with: count.2
bp count                bp count.2
1 *Count from 1-10      1 *Count from 1-10
D   2 equ tem to 10
I                       2 ten = 10
3 for i = 1 to ten      3 for i = 1 to ten
4   print i             4   print i
5 next i                5 next i
6 print                 6 print
7 end                   7 end
8                       8
```

compare-list

forms a single list from two lists using intersection, union, or exclusion operators.

"compare-list" performs the operation (&, #, =, +, or -) on "list.a" and "list.b" and creates a new list, "list.c". If "list.c" is not specified, it overwrites "list.a".

Operators and their meanings:

& Represents an "and", or "intersection", which means that the item-ids must exist in both lists.

Represents a "not", or "exclusion", meaning that the item-ids must NOT exist in both lists.

= Represents an "or", or "union", indicating that the item-ids may exist in either list.

+ Concatenates "list.b" to the end of "list.a". (list.a + list.b)

- Removes item-ids in "list.b" from "list.a". (list.a - list.b)

g Performs a "get-list" on "list.c". (Makes "list.c" active after the process.)

s Generates and displays statistics about the items selected during the process.

Syntax

compare-list {file.reference.a} list.a {operator} {file.reference.b} list.b {file.reference.c} {list.c} {(options)}

or, if primary and secondary lists are already active:

compare-list {operator} {list.c} {(options}

Options

? Displays help text on screen.

Example

```
Assume that the two following lists are already created and reside in the
"pointer-file".
list1  list2
001 cat  001 banana
002 dog  002 apple
003 banana  003 orange
compare-list list1 + list2
Since no "list.c" was specified, "list1" is overwritten and contains "banana",
"cat", "dog", "apple", "banana" and "orange".
compare-list list1 + list2 list3
This example is exactly the same as the first example, but the list is saved
as "list3". "list1" and "list2" remain unchanged.
compare-list list1 # list2 list3
This example creates a "list3" which contains "apple", "cat", "dog" and
"orange", since none of these appear in both "list1" and "list2".
"list1" and "list2" remain unchanged.
compare-list list1 = list2 list3 (gs
list2 list1 list3 Sort time - faster than a speeding second.
3     3     5 Duty time - faster than a speeding second.
[404] 5 items selected out of 1 items.
This example creates and activates "list3" which contains "apple", "banana",
"cat", "dog" and "orange", or all unique strings in both "list1" and "list2".
"list1" and "list2" remain unchanged.
compare-list list1 - list2 list3
This example creates a "list3" which contains "cat" and "dog". Those strings
which exist in "list2" ("banana", in this case) are removed from "list1".
"list1" and "list2" remain unchanged.
compare-list list1 & list2 list3
This example creates a "list3" which contains "banana", since it is the only
string which appears in both "list1" and "list2". "list1" and "list2" remain
unchanged.
```

compile

The "compile" verb invokes the Pick/BASIC compiler and translates the specified source code into executable object code with case insensitivity.

Case "insensitivity" means that the variables "TOTAL", "Total" and "total" all represent the same symbol. A case "sensitive" state would treat these as three separate variables.

Each line of the source program is scanned for syntax errors during compilation, even after an error has been detected.

A "." (period) is displayed for every 10 lines of code.

If an error is encountered, the line with the error and the error message are displayed; no object code is produced. The "q" option prevents the messages from scrolling past on the terminal.

If the "o" option is used, FlashBASIC is invoked after the first stage of compilation is complete. When all stages are completed, and if no errors are found, the compiled form is saved and can be executed using the "run" verb. The program may also be "cataloged" with the "catalog" verb.

"itemlist*" consists of one or more source program names, separated by spaces. If no name is specified, or if an asterisk (*) is used, all programs in the file will be compiled.

Programs may be created and modified using UP. They can be compiled when exiting UP by using either the <ctrl>+xc command, which compiles and catalogs the program, or the <ctrl>+xr command, which compiles and runs the program. Any other editor may also be used.

Syntax

compile file.reference itemlist* {(options}

Options

a	Displays object code generated by Pick/BASIC compiler.
b	Turns on array bounds checking when used with the "o" option level 1-9. Without this option in effect, situations such as "array subscripts out of range", which would formerly result in a "fatal" Pick/BASIC debugger abort, will ignore the condition, possibly resulting in a monitor halt. This option is not necessary when using the "o" option level 0 (the default), or when not using the "o" option at all.
c	Compresses the object by suppressing the end-of-line (EOL) opcodes from the object code item. This option is designed to be used with debugged and cataloged programs. Because the EOL opcodes are used to count lines for error messages, any runtime error message in a program compiled with the "c" option will indicate the error is on line 0. Additionally, the "c" option removes the ability to single-step with the Pick/BASIC debugger.
d	Prevents the run-time "d" option from entering the debugger.
e	Lists only error lines encountered during the compilation of the program. The listing indicates line number in the source code item, the source line itself, and a description of the error associated with the line.
f	When used with the "o" option, generates floating point arithmetic.
h	Hides FlashBASIC object code from other users. In other words code is not shared.
i	Lists lines from any included program as part of the listing. If used with the "l" option, the source program listing, the included lines are indicated by a "+" after the line number.
k	Keeps a shared, FlashBASIC module loaded. See the "shpstat" program for more information about shared loading.

l	Generates a line by line listing of the source program during compilation. Error lines with associated error messages are indicated. When the "l" option is used, "*"'s are not displayed. Each line of the listing takes the place of the "*".
m	Generates a program "map" of the descriptor table and correlates source code lines to generated object code frames. Each variable in the program is listed along with its decimal offset.
n	No pause; suppresses pause at end of page on terminal display.
o{level.number}	Produces FlashBASIC optimized code. This option may be followed by a number from 0 to 9, which indicates the FlashBASIC (optimization) "level". If not specified, the level defaults to 0. Level 0 generates optimized cross-platform compatible code. This code is much smaller than level 1 code and can be moved from platform to platform. Level 1 generates platform-specific code which tends to be about 10% faster than level 0 code. Levels above 1 produce platform-dependent code with incrementally smaller performance increases than that obtained by level 1. Using such high optimization settings can make compile time take several times longer and should not be used except where performance is absolutely critical. Note that if any module in a program is compiled with the "o" option, then all modules. See the related subjects on FlashBASIC for more information.
p	Routes all output generated by the compilation, except the cross-reference listing, to the printer, via the Spooler.
s	Suppresses generation of symbol table. The symbol table is used exclusively by the Pick/BASIC debugger for reference; therefore, it needs to be kept only if the user wishes to use the debugger.
x	Creates a cross-reference of all the labels and variables used in the Pick/BASIC program and stores this information in the "bsym" file, which must exist before using. The "x" option first clears data in the "bsym" file, then creates an item for every variable and label used in the program, using the variable or label name as the item-id. After creating the cross-reference items, attribute one contains the line numbers where the variable or label is referenced; each line number is a value. The line number where a label is defined, or where the value of the variable is changed is preceded by an asterisk (*). If the variable names and labels are in both upper and lower case, the "d-pointer" for the "bsym" file-defining item should not have the "s" option.
w	Optimizes without source when used with the "o" option. This allows optimizing programs without source code. This option requires a list of the items in the dictionary level of the file to be active prior to use. Note that the normal dictionary object statistics are not updated when this option is used.
y	Allow multiple FlashBASIC compiles concurrently. Normally, each FlashBASIC compile attempts to set BASIC lock 49 to keep multiple compiles from dramatically slowing down the machine. The Y option avoids this logic.

On D3 Unix releases, default options may be stored in attribute 6 of the "compile" item in your master dictionary. For example, putting an "o" in this attribute makes all compiles in that account produce FlashBASIC code.

Example

```
compile dm,bp, term-type (lp
This compiles the "term-type" program and sends the listing to the printer.
compile bp j (oc
This creates a FlashBASIC native object module which may be run from D3 as if
it were an interpreted Pick/BASIC program. The "c" option is used primarily to
reduce object size and compile time, both of which can be significantly
greater when using FlashBASIC.
select dict bp.old
[404] 114 items selected from 114 items.
compile bp.old (ow
This creates FlashBASIC code for all the Pick/BASIC object pointers in
"bp.old". There is no need to have the source present.
ct md compile
001 VR
002 3]9
003 F
004 dm,bp, :ccompile
005
006 o
Notice that an "o" was placed in attribute 6. This verb, when invoked will
always Flash compile programs. (see o option)
```

compile-catalog

The "compile-catalog" command ("<ctrl>+xc" from within the Update processor) compiles and catalogs a Pick/BASIC program. It may also be used from the TCL prompt, when provided with a file and item-id, or list of item-id's.

All options used with the "compile-catalog" command are passed to both the "compile" and "catalog" programs.

Syntax

compile-catalog file.reference itemlist*

compile-run

compiles and runs a FlashBASIC program.

Syntax

compile-run file.reference item.list*

<ctrl>+xr (from UP)

compile.catalog

invoked by the "compile-catalog" command.

config

invokes a menu for the administraion of the system configuration for the current virtual machine, or for a virtual machine specified by "vmname".

Without any argument, a master menu is displayed. With an argument (specified by "topic"), the configuration screen is entered directly. All changes made to a configuration file are not taken

into account immediately. It is necessary to shut down the D3 virtual machine and reboot it for the changes to take effect.

Most configuration operations are done using the Update processor. See the Update processor commands section. All fields have an on-line help facility activated by typing '?'<return> in the first position of the field. To commit a change, type <ctrl>+x followed by 'f'. To abandon a change, type <ctrl>+x followed by 'e' and 'y' to confirm.

Changing a configuration requires that the D3 process is run while logged on as root. To activate a D3 process as root, type:

su

>password: (enter root password)

ap -l

Each topic is further discussed in separate entries in the on-line documentation. The following topics can be addressed:

"startup"

Initial system configuration. This option should be run during a virgin installation, the very first time D3 is loaded on the system. It is not allowed to run the startup option for another virtual machine. This option must be run while logged on as root.

"ports"

Configuration of the ports settings. The number of physical ports (or pibs) and phantoms can be adjusted. This is not the number of licensed users.

"core"

Core (D3 memory) configuration, virtual machine key and size of the optional memory set aside to catalog FlashBASIC locked object code.

"flush"

Flush process adjustment and size of the write queue.

"security"

Name of a valid Unix user, owner of the virtual machine, and list of the Unix groups allowed access to the virtual machine.

"disk

List of the logical disks allocated to D3. It is not advised to modify this field. This should be for reference only.

"tape"

List of the devices used as a tape by the virtual machine. This list can be edited to add or remove devices added or removed after D3 has been installed.

"options"

Edit various options.

"all"

All options. A complete configuration file is edited.

Syntax

config {topic} {vmname}

Example

```
config
Enters the main configuration menu.
config core
Changes the memory size.
config all dev2
Edit the whole configuration file for the the virtual machine 'dev2'.
```

config core

command enters the Update processor to show memory usage of the specified virtual machine.

The following elements are edited:

"Core Size (K)"

This is the size in Kilobytes allocated to the D3 shared memory segment. The size of this segment is critical for performance. Roughly, the higher, the better. This number has to be balanced, however, with the requirements of Unix itself, any other Unix applications running on the system and the physical memory available. The objective is to avoid paging, both from D3 and from Unix. The D3 "buffers" command will show low 'available' buffers when the core size is too small, and the Unix command "sar" will show high paging activity when this number is too high. If it is impossible to find a compromise, the physical memory (RAM) is probably insufficient. After changing this argument, it is necessary to shut down D3 and bring Unix to single user mode.

"Key"

This number, in hexadecimal, must be unique on the system. Its actual value does not matter. It is an identifier of the virtual machine, as seen by other D3 virtual machines or Unix applications. Use the Unix command "ipcs" to determine keys currently used.

"Shared BASIC size (K)"

This optional field is the size in kilobytes allocated to contain the most frequently used FlashBASIC programs. The TCL command "shpstat" shows the usage of this area. Locked and/or sharable, FlashBASIC object code is created by the TCL "compile" verb.

Syntax

config core {vmname}

Example

```
Core size (K)         16300
Key (hex)             10
Shared BASIC size (K) 1024
```

config disk

enters the Update processor to show the disk configuration.

Up to 16 logical disks can be used as the D3 file system. When more than one volume is used, they are logically concatenated and appear as one single logical volume.

This option should be used only to examine the disk configuration. Changing any of the parameters will require a full file restore the next time the virtual machine is rebooted.

If "vmname" is not specified, the configuration file of the current virtual machine is edited.

"Disk device name"

This is the Unix device name of the logical disk used as the D3 disk. Permissions to this device should be restricted to 'root', for security reasons. The exact nature of the device is highly implementation dependent. It can be a partition, a Unix file, local or remote, or an unmounted Unix file system. In fact, any direct access device can be used.

"Start blk"

This is the logical start of the D3 data space, expressed as an offset in kilobytes from the start of the logical disk. This number should normally be set to 0.

"Size (K)"

This is the size in kilobytes of the logical disk. The size should not be less than 128 kilobytes, and no more than 2 Gigabytes (2,097,152 K). If the D3 virtual machine requires more than 2 gigabytes of space, several logical disks have to be used.

"Comment, options"

Free text. On D3/SCO, however, if the logical disk is a partition located outside of the Unix file system (external), the comment field must start with track testing replacement. The installation procedure adds these automatically.

Syntax

config disk {vmname}

Example

```
Disk device name   Start blk Size (K) Comment, options
/dev/rhd01         0         32010    t 37 Partition
/dev/rpick         0         20000    Division
Two logical disk are defined (in the case of D3/SCO):
- One partition of 32010 K, with a pseudo track of 37 kilobytes, for
bad track detection.
-  One division (i.e., an un-mounted Unix file system) of 20000 K.
The total of the D3 virtual space is 52010 K.
```

config flush

enters the Update processor to show the flush parameters.

The flusher is a background process ("daemon") which is in charge of copying modified memory back to disk. This process is normally sleeping, until either the number of pages requiring to be copied back to disk reaches a predetermined level, or after a fixed time.

Activity of the flusher can be monitored by the "buffers" TCL command.

"Flush period (s)"

This is the period in seconds after which the flush process wakes up to see if any pages in memory needs to be written back to disk. If the system is busy at that time, the flush process goes back to sleep almost immediately, since writing data to disk would load the disk when the application might need to do some useful reads. If the system is quiescent, the flush process starts writing data back to disk until a user process needs the disk again. The lower the flush period, the more often the memory will be copied back to disk. An absolute minimum should be a few seconds and the maximum around 30 to 60 seconds. This number can be changed dynamically by the TCL command "set-flush".

"Size of write queue"

This number determines the threshold at which the flusher wakes up to copy modified pages back to disk. For most systems, a value of 256 to 512 is suitable. If the memory is really low

(less than a few megabytes), this number can be reduced to around 32, which should be a minimum.

Syntax

config flush {vmname}

config options

enters the Update processor to show various options of the current virtual machine or of the specified virtual machine "vmname".

"Machine name"

This text is returned by the 'which' (TCL) command and the system(100) FlashBASIC intrinsic function, after the name of the D3 configuration file. It can be any string. The installation procedure always changes this field to a default.

"Boot mode"

This one-letter option defines whether the system comes up in single ('s') or multi-user ('m') mode. When in Multi-user mode, users are allowed to log on as soon as the system starts (right after the message 'diagnostics ... successfully completed'), during the execution of the system-coldstart procedure. When in single user mode, user logon is denied until the line 0 executes the 'maxusers (mlq' command. This command should be included at the end of the 'user-coldstart' macro, after all application-specific commands are done (starting printers, setting terminals, communications, etc...). The default is to boot in single user mode.

"Auto boot sleep"

This number specifies in seconds the grace delay allocated when doing an auto boot ('ap -a x') to stop the boot and return to a manual boot. The default is 3 seconds.

"ABS lock"

This flag, "on" or "off", controls whether the entire D3 kernel is brought in memory at boot time ('Locking ABS ...'). The installation procedure turns this flag on automatically if there is enough memory allocated to D3. Severe performance degradation will occur if this flag is turned off. Disabling the ABS locking should be reserved for very small configurations (less than a few megabytes).

"Break char (hex)"

This is the default value of the alternate break character, coded in hexadecimal. This value can be overridden on a per-port basis by the 'set-break' TCL command.

"Escape char (hex)"

This is the default value of the escape level pushing character, coded in hexadecimal. This value can be overridden on a per port basis by the 'set-esc' TCL command.

"Priority"

This number specifies the relative priority of the D3 processes running on the virtual machine, compared to other Unix processes, by affecting the 'nice' value. Legal values are from -20 (highest priority) to +19 (lowest priority). '0' is a 'neutral' value, giving equal priority to D3 processes and non D3 processes. The flush process always run at a nice value of -10 (higher priority) than the base line.

Syntax

config options {vmname}

Example

```
Machine name      development
Boot mode         s
Auto boot sleep   5
ABS lock          on
Break char (hex)  1c
Escape char (hex) 1b
Priority          10
These options have the following effect:
- system(100) in FlashBASIC will return the string:
    pick0:development
- The system is booted in single user mode.
- The grace delay for booting automatically is set at 5 seconds.
- The alternate break char is set to x'1c'. This key normally
associated with <ctrl>+\ (control + backslash).
- The escape level pushing is associated to x'1b', which is the
normal escape character.
```

config ports

enters the Update processor to show the port and phantom setting or the current virtual machine or of the specified virtual machine "vmname".

By default, the installation procedure reserves a fixed number of ports (or pibs) and phantoms. Depending on the hardware platform, these numbers could be, for example, 256 ports and 32 phantoms.

"Number of ports"

This is the expected number of devices (serial, parallel, network, etc...) that the system will be able to support. This number should be at the very least equal to the number of licensed users plus the number of printers, serial and parallel. It is advised to put some margin on this number to allow for extensions. Putting more that the required number costs only a few kilobytes on disk, which is negligible.

"Number of phantoms"

This is the number of phantom processes the system will be able to use. This count does not include the scheduler itself, which is always added to this number. This number should be at least 2.

Syntax

config ports {vmname}

Example

```
Consider a system with 256 ports, 32 phantoms, licensed for 64 users.
If the system is equipped with 128 serial devices, it would be
possible to have (for instance):
98 terminals,
30 printers.
All the 98 terminals would receive the D3 logon message, but no
more than the licensed number of users (64) would be allowed to log
on to D3 at any given time. This configuration would leave room to
expand to 256-30=226 licensed users.
```

config security

enters the Update processor to show the 'user' and 'groups' definition of the virtual machine:
"Unix User name"

This optional field contains the name of a valid Unix user, called the 'owner' of the virtual machine. It is strongly recommended to not leave this field empty, otherwise all users connected to the virtual machine effectively have root access to all the resources of the Unix system, which is a security risk. When the user is defined, it becomes the 'owner' of the virtual machine. All users allowed to be connected to the virtual machine become this user, as far as Unix is concerned. The Unix administrator should give proper authority to this special user so that all D3 users can use the required Unix resources (communications, access to Unix files, etc...).

"Authorized Unix groups"

This optional field contains the list of the Unix group(s) to which a Unix user must belong to be granted access to the virtual machine. To enter more than one group, type <ctrl>+v at the end of each entry to open a new line. If the group list is empty, then all users have access to the virtual machine.

Syntax

config security {vmname}

Example

```
Unix user name            pick
Authorized Unix groups
All users are allowed to get to the virtual machine, and they all
become the user 'pick'.
Unix user name            pick
Authorized Unix groups    staff
                        development
Only the users belonging to one of the groups 'staff' or
'development' are allowed to connect to the virtual machine, and they
all become the user 'pick'.
Unix user name
Authorized Unix groups    system
Only the users belonging to the group 'system' are allowed to connect
to the virtual machine, and they all become 'root'.
```

config tape

enters the Update processor to show the list of tapes currently defined in the specified virtual machine.

Up to 16 devices are supported. The installation procedure senses the real tape devices and add them to the configuration file, along with pre-defined pseudo floppies. The System Administrator can add devices freely.

"Tape device name"

This is the Unix device name of the peripheral. The System Administrator must ensure that proper access rights are set so that the virtual machine can do all operations on it. If the file is a special file (a device), then use the raw (character) rather than the block device.

"blksz"

This is the default block size of the device. This field is used only at boot time, in case a file restore is attempted on this device. It has no influence on the default block size defined by the "t-att" command.

"t"

This is the type of the device. The following types are supported:

f Floppy, real or pseudo

h 1/2" tape

q 1/4" tape

v 8mm tape

c serial device or network or Unix pipe

d 4mm tape

"Opt"

This string defines the various options. Depending on the device, the following options are available:

On All devices:

l Link field present for ABS restore. Required for floppy and network.

On Floppies only:

o 3 1/2", 2.88 M

q 3 1/2", 1.44 M

d 5 1/4", 1.2 M

l 3 1/2" or 5 1/4", 720K

s 5 1/4", 360 K

x pseudo floppy (no size)

On Quarter-inch /4" tape (SCT) only:

h highest density

s standard density

On Half-inch tapes (9 track) only:

h 6250 bpi

m 3200 bpi

s 1600 bpi

"Comments"

Free text. On D3/SCO, however, if the device is a quarter-inch (SCT), the comment must start with the auxiliary device name. For instance:

Tape device name blksz t Opt Comments

/dev/nrct0 16384 q lh /dev/xct0 #1

Syntax

config tape {vmname}

control-chars

enables or disables the entering of control characters at input, depending on the option specified.

Under default conditions, non-editing control characters are accepted by standard input commands. By doing "control-chars (f", these characters are ignored.

The settings of this option are sent up and down levels for the current port.

The "control-chars" setting may be temporarily overridden in a BASIC program with the "INPUT CTRL" statement.

Note that this setting does not affect the standard TCL stacker since it utilizes all control keys for editing.

Syntax

control-chars {(options}

Options

f Disables control character input.

n Enables control character input. This is the default

s Suppresses output of status message.

conv-case

converts the characters within specified items to upper or lower case, according to the options provided.

If no option is provided, all characters are converted to upper-case.

Syntax

conv-case file.reference itemlist* {(options)}

Options

? Provides "help" on syntax and options (below).

e Used with "q" option to convert case of items within quotes due to "chain", "tcl", and "execute" statements. If the "e" option is used by itself, all characters are changed to upper-case. If "e" is used with the "q" option, it changes everything except strings in quotes that are not preceded by "chain", "tcl" or "execute".

i Suppresses display of item-id(s).

l Converts upper case text to lower case.

o Outputs the item-ids on the same line.

q Leaves text within quotes unchanged.

u Converts lower case text to upper case (default).

Example

```
conv-case bp * (equ
Converts all the items in the "bp" file to upper case characters,
except literals inside quotes or backslashes. Each item-id displays
on a separate line.
```

converse

links the current process to a device. The "target" or "slave" device is put into "pass-through" mode, allowing data in and out in "raw" mode. The target device (the one being "linked" to the current device) must be available on the given "port.number". It may not be already attached or linked to another process.

A "hotkey", or "termination", sequence disconnects the link. The default "hotkey" sequence is <escape>x. The default can be modified by using the "d" option.

If the termination sequence contains an escape character, as in the default, an "esc-data" command is issued automatically, so the process will not get locked out from being able to suspend "converse".

When the device is successfully attached, "LINE ATTACHED" is displayed on the master terminal, unless the (S) option is used.

Hitting the <BREAK> key on the master terminal, sends a <BREAK> to the slave port.

On all Unix implementations, if the D3 process is not connected to the virtual machine, then "converse" is not allowed, even if the "where" verb with a "z" option shows the port as being available.

Syntax

converse port.number {(options}

Options

c Captures all output on the (slave) screen and writes it to an item in the "dm,cap-file," using the item-id "*n", where "n" is the current port.number.

d/hexnum{,hexnum...} Designates the sequence of characters to terminate "converse" mode. Each character is provided in its hexadecimal equivalent of its ASCII value. Each character is separated by a comma. The "/" is required after the "d" option and before the first hex number. For example, "d/41,42,43,44", indicates that an "ABCD" will disconnect the link.

i Same as the "c" option, except that output from the master process is captured.

n Suppresses capturing on converse mode. This is the default.

s Suppresses the message when the process terminates.

x Terminates converse mode on target device number and returns target device to logon.

z Used in conjunction with (I) or (C), translate the non printable characters into a dot, the carriage returns into an attribute mark and suppresses the line feeds. Without this option, if a character x'ff' is captured, the item is truncated.

Example

```
converse 16
Attaches the current process to port 16.
converse 33 (c
Attaches the current process to port 33, capturing screen output from
the slave process.
converse 3 (d/2a,2b,2b
Attaches to port 3 and sets the hotkey sequence to "*++".
converse 42 (iz
Attaches to port 42 and captures the screen output from the master
process, making sure all data is displayable.
converse 42 (x
Terminates converse on port 42 and returns port to logon.
```

copy

copies items to an output device, to another item-id, or to another file.

If an output device is not specified in the options (either to the terminal or printer), The prompt, "to: " displays on the next line.

If copying to another file, a left parenthesis must precede the file name. A closing right parenthesis is not needed. If the "file.reference" is not specified at the "to:" prompt, the items are copied to the source file.

If no "item.list*" is included, the items in the destination file are given the same item-ids as the items in the source file.

The "file.reference" may refer to a dictionary level by preceding the file.reference with the word, "dict", followed by a space. No "itemlist*" is required when the copy follows a "select", "sselect", "get-list" or "qselect" (or any other "list-generating" process).

File-defining items (FDI's) are "protected" and may not be copied with the "copy" command. See "move-file", "rename-file" and "steal-file".

Syntax

copy file.reference itemlist* {(options)}

to: {(file.reference} {itemlist}

Options

n (integer number) Specifies the number of items to copy. Typically used for copying data for test files.

a Activates assembly "mlist" format.

d Deletes source (original) item after copying. Not valid with "t" or "p" options.

f Outputs a form-feed between each item; only valid with "p" and "t" options.

i Suppresses display of item-ids when copying between files.

m Activates macro (assembly) format on terminal or printer output only.

n Activates nopage function on output to the terminal; only available with "t" option.

n When used with "o" option, this inhibits new items on copies between files. This means that the item-id must already exist in order for the item to be copied.

o Overwrites existing items when duplicate item-ids exist; has no effect if used with "t" or "p" options.

p Directs output to system printer, via the Spooler. See also the "cp" command.

s Suppresses display of line (attribute) numbers when used with "t" or "p" options.

s Suppresses error message display when copying between files.

t Directs output to the terminal. See also the "ct" command.

u If the item-id is in the destination file, the item is copied to a new item-id by creating a new item-id composed of the original item-id concatenated with a letter, starting with "a" (lowercase "a").

x Outputs in hexadecimal; only available with "t" and "p" options.

Example

```
copy dict invoices customer.name (p
copy invoices * (od
to: (invoices,archive
copy dict invoices size
to: (dict sales
```

copy-list

copies a saved list to either a new item-id, a new file.reference, or to the specified output device.

If a file.reference is not specified, the items are copied from the "pointer-file".

If no "itemlist*" is specified, the default item-id, "%user.name" is used.

In D3, a "file.reference" may be specified because any file can hold lists. Conversely, the "copy" command may also be used, but it does not default to the "pointer-file".

"cl" is a shorthand for "copy-list".

Syntax

copy-list {{file.reference} {itemlist*} {(options}

to:{(file.reference} {itemlist}

Options

n (integer number) Copies only the first "n" items.

d Deletes the source item(s) from the source file after copying.

f Issues a form-feed between items; each item starts on a new page. Only valid with "t" and "p" options.

i Suppresses display of item-ids.

n Activates nopage function on output to the terminal.

o Overwrites duplicate item-id{s}.

p Copies list to printer, via the Spooler.

s Suppresses display of line numbers when used with "t" or "p" options.

s Suppresses error message output on a file copy.

t Copies list to terminal.

u If item-id is in destination file, copies item, but renames it by concatenating current item-id with a character, starting with lower case "a".

x Lists output in hexadecimal.

Example

```
copy-list dm,pointer-file, temp.list (d
to:(prod,pointer-file, cust.list
This copies a list called "temp.list" from the "pointer-file" in the "dm"
account to the "pointer-file" in the "prod" account. After copying
successfully, the list is deleted from the "dm,pointer-file," with the "d"
option. Filepaths are provided in the source and destination files. This
command is valid from any account.
```

cp

copies the specified item(s) to the printer, via the Spooler.

Syntax

cp file.reference itemlist* {(options}

Options

n integer number). Specifies the number of items to copy. Typically used for copying data for test files.

a Activates assembly "mlist" format.

f Outputs a form-feed between each item.

s Suppresses display of line numbers.

x Outputs in hexadecimal.

Example

```
cp dm,bp, term-type
```

create

is a simplified method of creating a file with a set of attribute-defining items from the TCL command line.

"create" automatically generates the dictionary items and then immediately invokes the Update processor to allow data entry into the newly-created file.

The "create" program is "smart" enough to watch for certain keywords in the command line and to insert the appropriate processing codes in the "conversion" attributes. For instance, if it detects the string "date" as one of the "attr.names", it inserts a "d2/" as the output-conversion of the corresponding attribute-defining item.

It also searches for keywords such as "zip", "zc" and "csz" and inserts the "zc" processing code; "phone" inserts a generic telephone mask. All fields are automatically indexed.

Syntax

create file.reference attr.name {attr.name...}

Example

```
create players name phone birthdate specialty
players NEW ITEM
name
phone
birthdate
specialty
```

create-abs

sets aside a contiguous set of frames as the destination for loading the abs code (the D^3 Virtual Code). This space is called the "abs" area.

This command provides the ability to create a whole new executable abs to avoid the risk of corrupting the existing abs.

The item, "%abs%", in the dictionary of the "abs", file is a pointer to the abs area.

The first frame in the abs area contains the date and implementation of the abs.

The data section is used as a cross-reference file by the incremental loader.

"create-abs" executes a "clear-file" on the data section of the "abs" file, then creates one item in the data section for each frame in the abs area. These items contain an "F" in attribute 1. The item-ids are sequential numbers, starting with 000.

"file.reference" indicates the file where the modes are to be loaded.

If a "DBSYM" data file for both source and destination files exists, the destination will be cleared and the source copied over.

Note that the options are NOT optional.

Syntax

create-abs file.reference (options

Options

item-size An integer number parameter specifying the number of frames, in decimal, to set aside for the machine code (new abs area). The frames must be contiguous. If there are not enough contiguous frames, an error message is displayed.

l Prompts user for source file and copies the system modes from the source abs used in booting the system (boot abs) to the destination abs area, initializing the file for future loads. If no numeric parameter is specified, it uses the abs size of the source file to create the abs area. This option overrides the "z" option.

zframe.count ("z", followed by a number) Zeroes out the frames in the abs area. This option requires a numeric parameter to specify the size of the abs area, and is incompatible with the "l" option. The following prompt displays: Enter source data-abs file name (RTN for ABS): To run the boot abs, enter "boot.abs". To run a different abs, enter the name of the abs.

Example

```
create-file abs.test 3 395
create-file data abs.test,dbsym 395
create-abs abs.test (l
Enter source abs file name (return for boot abs): <return>
[1005] abs file creation successfully completed.
```

create-account

creates a new master dictionary, copies all the necessary items into it, and updates the "mds" file with the name and address.

The "newac" file in the "dm" account contains the items that are placed into a new account md. Additional, application-specific items may be added to newac. For security purposes, items can be removed from newac as well. For example, dangerous verbs such as "delete-file" and "clear-file" could be either removed or renamed to prevent accidental use.

See "master dictionary" for an explanation of the attributes requested in "create-account".

If the account.name is omitted from the command line, the process prompts for it.

Syntax

create-account {account.name}

Example

```
create-account fred
mds,, 'fred' NEW ITEM size = 31
mds,,          fred
type           D
modulo         37
ret-lock
upd-lock
password
syspriv
justification  L
width          12
reallocation
"mds,," is the item-id of the account to create
"type" describes the type of item.  D indicates this is a
file-defining item.
"Modulo" is the modulo used in creating the file for this account.
"Retrieval codes" and "Update lock codes" are unique locks for
limiting access to files within the account. See "retrieval locks /
update locks" for more information.
"Password" is the optional logon password for this account. The
password will be displayed as a formula conversion in hexadecimal.
"Syspriv" is the user privilege level.  May be "sys0", "sys1", or
"sys2". See "system privileges" for more information.
"Justification" and "width" describe the output specifications for
the MD.
```

```
"Reallocation" describes the reallocation modulo for the MD, if any.
This value must be enclosed in parentheses.
```

create-file

creates a new dictionary and data file, a dictionary-only file, or a new data section on an existing dictionary file; space is allocated and reserved if available.

Path-names to other accounts may not be used. The syntax varies according to the section of the file being created.

The first form creates a new file with both data and dictionary sections.

The second form creates a new file with only a dictionary section.

The third form creates a new data file section in an existing dictionary.

"file.name" is the name of the file to be created in the current account.

"dict.name" is the name of a dictionary to be created in the current account.

"data.name" is the name of a data file to be created in the specified dictionary.

"dict.modulo" indicates the number of frames to reserve in the primary file space for dictionary items. See "primary file space".

"data.modulo" indicates the number of frames to reserve in the primary file space for data items.

The internal modulos actually used may be slightly higher to give a more optimal hashing scheme.

When a dictionary-only file is created, a "q-pointer" (synonym-defining item) with the file.reference as its item-id, is automatically created in the dictionary. This allows the file to be accessed without specifying dict.

"cf" is a short hand fro "create-file".

Syntax

create-file file.reference dict.modulo data.modulo {(options)}

create-file dict file.reference dict.modulo {(options)}

create-file data dictionary.reference,data.reference data.modulo {(options)}

Options

l Logs any updates to this file to the transaction logger. This option is not compatible with the "x" or "y" options.

n Negates all update protection for the given file irrespective of the global update protection setting. This option is not compatible with the "u" option.

p Primary file space is to contain only "pointer" (indirect) items regardless of item length.

s Item-ids are upper/lower case-sensitive. This option is compatible with all other options.

u Enables update protection for the given file irrespective of the global update protection setting. This option is not compatible with the "l" option.

x Does not save this file on "save-type" (file-save, account-save, etc.) backups. The file will not exist after a file restore.

y Does not save the data in this file on "save-class" (file-save, account-save, etc.) backups. On a file restore, the file is recreated, but will be empty.

The options are included as part of the "d/code" in the file identification items in the master dictionary for dictionary files and in the file dictionary for data files.

The "l", "x", and "y" options can also be added or deleted from an existing d/code by using "update" or "ud". The "s" code cannot be added or deleted.

Example

```
create-file customers 7 101
This creates both a new dictionary and data section.
create-file dict pointer-file 31
This creates a dictionary-only file.
create-file data customers,archive 101
This creates a new data section called "archive" and places a pointer
to it in the dictionary of customers. The new data section may now
"share" all the attribute-defining items in the customers dictionary.
```

create-index

creates a new "b-tree" index in the specified file using the "a" processing code expression provided.

An "a" processing code or an "*" (asterisk) must be specified.

"a.code" specifies the "a (algebraic)" processing code to be used in forming the keys to the index. The processing code must include an attribute number immediately after the "a." The first attribute number in the processing code is the "master" attribute. This attribute determines the numer of values that an index key will generate.

The "*" (asterisk) designates all indices. This may be used as an alternate to a series of "create-index" commands by placing the "i" processing codes directly into the file-defining item ("d-pointer"), then using the "*" with the "create-index".

Once an index has been created, any changes made to the file through any process which updates the file are automatically reflected in the index.

An existing index may be overwritten using the "o" option.

The pointer to the index is placed in the "correlative" attribute of the file-defining item (attribute 8) as an "i" processing code. The number following the "i" indicates the "root" fid of the index. This should never be manually altered.

Syntax

create-index file.reference a.code {(options)}

create-index file.reference * {(options)}

Options

s Suppress the display of the running count of items.

o Overwrites existing indices.

l On releases 6.1.0 and higher, create-index will lock only the index nodes on which it is currently working to facilitate parallel access by other users. The "L" option will force create-index to lock the entire file during index creation (This was the default on pre-6.1.0 releases).

Example

```
create-index entity a1
This creates an index on the contents of attribute 1.
create-index invoices a3(tcustomer.file;x;;1)
This creates an index on the results returned from attribute 3 of
invoices, translated to customer.file, and returning attribute 3 from
```

```
the item, if found.
create-index sales a1:3
This creates an index on attribute 1 concatenated with attribute 3.
This example illustrates how a series of indexes could be built:
First, the index references are placed in the correlative attribute
(8) of the (entity) files d-pointer, so it looks something like this:
correlative      Ia1
Ia23
Ia27:6
Next, the command is issued:
create-index entity *
Afterwards, the correlative attribute contains the "root fid" for
each of the indexes.
```

create-macro

creates an executable "macro" from the last TCL command entered and places the macro in the md of the current account.

The macro is stored in the master dictionary and is executed by entering the macro name at the TCL prompt.

If attribute 1 of the macro is an "m", the macro.name must be enclosed in quotes when referenced from TCL. An "m" type is the default when using "create-macro".

If the macro.name already exists, a proper message will be given. The (O) option allows overwriting it.

During activation of the macro, any additional parameters following the macro name at the TCL prompt are passed into the first executable command in the macro, but do not affect any other commands in the macro.

Any number of valid TCL commands may be placed in a macro. The first value of each attribute must be a valid TCL command. Subsequent values are treated as data to the command.

D3 only allows macro's to be placed in the md of an account.

Syntax

create-macro macro.name {(option}

Options

m (default if no option is given). Creates a "pause and display" type macro. When activated, the TCL command contained in the macro displays at the command prompt and can be modified before executing. Attribute 1 becomes an "m".

n Creates a "non-stop" macro which executes immediately upon activation, without displaying the TCL command being issued by the macro. Attribute 1 becomes an "n".

o Overwite the macro if it already exists.

Example

```
If the last command on the stack was the Access sentence:
list entity name phone
and the next command is:
create-macro show.entity (m
Assuming that the item is NOT already present, the message:
show.entity created
appears on the screen.
If the "show.entity" macro had been present, the message:
[415] 'show.entity' exists on file.
Executing macros:
Assuming that the macro "show.entity" is stored in the md, and is
```

```
defined as shown in this example, following are various methods by
which it may be executed:
First, simply executing the macro by itself:
show.entity<return>
list entity name address
The "list entity name address" appears on the next line at TCL, with
the cursor under the "l" in "list". Any aspect of the sentence may be
modified prior to execution. The entire Update processor command set
is available for modifying the command.
Pressing <return> issues the sentence as presented.
Secondly, passing input from the command line:
show.entity hdr-supp leg-supp<return>
list entity name address hdr-supp leg-supp
Again, because this a "pause"-type macro, the TCL command is
displayed prior to execution, with the "hdr-supp" and "leg-supp"
modifiers appended to the command string.
An alternate example of passing input to the command:
show.entity with phone "714]" hdr-supp leg-supp<return>
list entity name address with phone "714]" hdr-supp leg-supp
Again, any element that is valid with the first TCL command in the
macro may be provided from the TCL command line. The input is only
passed to the first valid TCL command within the macro.
Modifying macros:
Since "show.entity" is an "m"-type macro, to modifiy it, the item-id
must be referenced within quotes:
u md "show.entity"<return>
md 'show.entity'  size = 46
01 M
02 list entity name address
At this point, the full complement of Update processor commands are
available. Additional TCL commands may be placed on subsequent
attributes. The "M" may be changed to an "n" to make it a "non-stop"
macro, which means that it executes the first TCL command immediately
upon activation. Since macro's are executable, if the UP command
<ctrl>+xr is used, the macro will be filed and automatically
executed.
```

create-nqptrs

builds a "q-pointer" for each "d-pointer" file in the "dm" account and adds them to the "dm,newac," file.

These "q-pointers" are provided to each new account subsequently created.

create.account

invokes the "create.account" program.

crt-delimiters

enables/disables the display of attribute and value marks.

When enabled, attribute and value marks are displayed as '^' and ']', respectively.

When disabled, attribute and value marks are replaced with a CR/LF, instead.

At boot time, 'crt-delimiters' defaults to on.

If the "crt-delimiters" statement is executed without options, then it displays the current status.

Syntax

crt-delimiters {(options}

Options

n Displays attribute mark as '^' and value mark as ']'.

f Not to display attribute and value marks.

ct

invokes the "copy" command and copies the specified item(s) to the terminal.

Syntax

ct file.reference itemlist* {(options)}

Options

n (integer number). Specifies the number of items to copy. Typically used for copying data for test files.

a Activates assembly "mlist" format.

f Outputs a form-feed between each item.

i Suppresses display of item-id(s).

n Activates nopage function on output.

s Suppresses display of line numbers.

x Outputs in hexadecimal.

Example

```
ct dm,bp, term-type
```

cvtcpy

is a utility to copy a tape to another tape, doing format and block conversions.

This utility reads D3 or Ultimate tape formats, with a block size up to 64K, and creates a D3 tape.

Arguments are expressed in any order, as "keyword=value" and are all optional, except 'if':

if Input device Unix name. This field is required.

of Output device Unix name. If omitted, the data is dumped on the terminal or printer, in decimal, or hexadecimal if the (X) option is used. The output device or file must exist.

ib Physical input block size. This size is not necessarily the attached size. Some devices have header information in each block. If not specified, the input block size is obtained from the label, if present. The block size is expressed in decimal, in bytes or in kilobytes if followed by a 'k'.

tib Physical tape input block size. This size is the block size on the input tape. If not specified, 'tib' is set equal to 'ib'. Some devices have a physical block size which is dictated by the controller. For example, on HP-UX, reading a 16K block requires reading 32 512 byte blocks. IN this example, we would have ib=16k and tib=512. The block size is expressed in decimal, in bytes or in kilobytes if followed by a 'k'.

ob Physical output block size. This size is not necessarily the attached size. Some devices have header information in each block. The actual attached size is in fact ob-oo (see 'oo' below). If not specified, the output block size is set equal to the input block size after it has been determined. The block size is expressed in decimal, in bytes or in kilobytes if followed by a 'k'.

io Input offset. This offset specifies at which byte offset the data actually starts in the physical block determined by ib. If not specified, the offset is assumed to be null.

oo Output offset. This offset specifies at which byte offset the data actually starts in the physical block determined by ob. If not specified, the offset is set equal to io.

il Input label block size. This value specifies the size of the block in which the label is found. If not specified, the label block size is assumed to be equal to the input block size. On devices which support variable block length (1/2"), this argument can be considered as the maximum block size of a label block. On devices where the device writes data in fixed blocks (SCT), it is important to specify the label block size exactly.

ol Output label block size. This value specifies the size of the block in which the label is written. If not specified, the label block size is assumed to be equal to the output block size. The logical label is written at the beginning of the label block, at the offset 'oo', if specified.

reel Output reel number. If not specified, the reel number is extracted from the input label.

fwd Number of files to skip before reading.

skip Number of blocks to skip before reading. The label is not skipped, unless a (U) option is used. If both 'fwd' and 'skip' are specified, the 'fwd' operation is done first, and then the blocks are skipped.

seek Number of blocks to skip on the output device before writing. This option is supported only if the device is a direct access device (floppy or file).

count Maximum number of input blocks to tansfer. If not specified, the data is copied until an error occurs or two consecutive file marks are encountered. When the maximum number of blocks has been reached two file marks are written on the output device, thus truncating the data.

files Maximum number of input files to tansfer. If not specified, the data is copied until an error occurs or two consecutive file marks are encountered. When the maximum number of files has been reached a second file mark is written on the output device to terminate the tape properly.

itype Input device type. This option allows setting usual defaults for the device block size, label size and offset. These values can be explicitly specified to override the default. The valid types are: 'floppy', 'sct', '8mm', 'dat' or '4mm', 'half' or '1/2'. If the device is a pseudo floppy (Unix file), the device type is determined automatically and can be omitted.

otype Output device type. This option allows setting usual defaults for the device block size, label size and offset. These values can be explicitly specified to override the default. The valid types are identical to 'itype'.

el Embedded label length. This option allows reading labels that are embedded in a block of data instead of being located in its on block.

When possible, the label logical format is determined automatically.

Syntax

cvtcpy {keyword=value} {(options}

Options

F Do not write any filemark.

I Ignore block size on tape label, use values set by tcl parameters.

N Do NOT discard the first block on the second source reel. By default, cvtcpy assumes that the last block on reel was duplicated on the following reel, and discards it when reading. This option

keeps the first block. If a CIE end or reel label (_R) is recognized, this option is set automatically.

P Outout to printer.

Q Quiet. Suppress all messages, except the final message and the error messages.

S Single reel. Disable the mechanism by which 'cvtcpy' attempts to cross reels on the source tape. Without this option, three consecutive file marks are interpreted as an end of tape.

U Unlabelled tape. No label is expected and none is written. With this option, the block sizes must be explicitly specified.

V Verbose. Displays more information about the process.

X Dump data in hexadecimal when there is no output device (dump to terminal or printer).

Example

```
cvtcpy if=/dev/rmt0.1 itype=half of=/dev/rmt1.1 otype=8mm
  Copy a half inch tape to an 8mm tape. All defaults apply (output
block size 16k, no offset, label block size=512).
cvtcpy if=/tmp/floppy (x
  Dump a pseudo tape (Unix file) on the terminal, in hexadecimal.
cvtcpy if=/dev/rmt1.1 itype=8mm of=/dev/rmt0.1 ol=512 ob=12000 oo=0
  Copy an 8mm tape to /dev/rmt0.1 (whatever it is), specifying a
label block size of 512 (the actual label data is still 80 bytes), a
physical data block size of 12000, and no offset (which means the
attached block size is 12000-0=12000).
cvtcpy if=/dev/rmt0.1 itype=half of=/dev/rmt1.1 otype=8mm fwd=1
files=3
  Copy 3 files, skipping the first file, from a half inch tape to an
8mm tape.
```

data

stacks the data "string" from within a paragraph.

Syntax

data string

date

returns an internal, external, or julian date derived from a date provided, or displays a calendar for the month and year of the date provided.

Displays the date given in "date.parameter" in external, internal, and Julian formats.

"date.parameter" is the date to convert and display. It may be in external, internal, or Julian date format. If it is not provided, the current system date is the default date.

The (C) option displays a calendar. The operator is then given the choices:

D = Previous Month (same year)

F = Next Month (same year)

X = Exit

Y = Previous Year (same month)

U = Next Year (same month)

Syntax

date {date.parameter} {(options}

Options

c Produces a calendar of the month containing the given date.

j Indicates that the "date.parameter" is a Julian number.

n When used with the "c" option, displays calendar only and does not prompt for additional options.

Example

```
date
February 22, 1992; Saturday  internal: 8819  julian: 53
date 1/1/2000
January 1, 2000; Saturday  internal: 11689  julian: 1
date 10000
May 18, 1995; Thursday  internal: 10000  julian: 138
date 1/16/97 (c)
January 16, 1997; Thursday  internal: 10609  julian: 16
+---------------------------------+
|  January         1997           |
|SUN |MON |TUE |WED |THU |FRI |SAT |
|    |    |    |  1 |  2 |  3 |  4 |
|  5 |  6 |  7 |  8 |  9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 |    |
|    |    |    |    |    |    |    |
+---------------------------------+
(D=Prev Month, F=Next Month, X=Exit)
(Y=Prev Year,  U=Next Year         )
```

date.iconv

is a FlashBASIC subroutine which converts an external date to its internal format.

The external subroutine only allows one argument to be passed. This argument is put through the "iconv" function with a "d" processing code. The internally converted response replaces the value of "date.argument", if "date.argument" is a variable.

Syntax

call date.iconv(date.argument)

Example

```
print "Enter an external date:  ":
input date
call date.iconv(date)
print "The internal date is ":date
```

db

Invokes "debug" command.

dcd

toggles or displays the status of data carrier detect on the specified port.

If the port number is not specified (as an option), dcd is toggled on the current port.

When activated, this protocol allows the system to sense changes in the dcd signal. When the system senses a loss of the dcd signal on a port, it logs the port off.

Without any option, the status of the specified port is displayed.

On D3 Unix implementations, options and parameters are interchangeable.

This command affects the argument "clocal" of the terminal driver, when the "f" option is used.

When the dcd is lost, the process is interrupted by a Unix HANGUP signal which forces it to execute the TCL command specified by the "trap" command. (see "trap, TCL, Verb").

An appropriate cable must be used to connect the system to the modem for the dcd signal to be detected. When the "n" option is used, the terminal driver is changed to "-clocal".

Syntax

dcd {(options}

Options

n (integer number) Displays or changes settings for port n.

f Toggles dcd off. Data carrier detect is inactive.

n Toggles dcd on. Data carrier detect is active.

s Suppresses output of status message.

x Displays the current carrier detect status (x is any character other than "f", "n", or "s")

Example

```
dcd
[1311] Data carrier detect is off.
Tests the current state of dcd.
dcd ?
[1311] Data carrier detect is off.
dcd n
[1310] Data carrier detect is on.
dcd 16 fs
(No output appears. This turns off dcd on port 16 and suppresses the
output).
```

dcd-off

toggles off data carrier detect on the specified port, or the current port if none is specified.

Syntax

dcd-off {port.number}

dcd-on

toggles on data carrier detect on the specified port, or the current port if none is specified.

Syntax

dcd-on {port.number}

debug

Enters the system debugger (if at TCL) or enters the FlashBASIC debugger (if running a FlashBASIC program).

Even if the verb "debug" is not available, the system will still enter the debugger with the proper privileges.

Example

```
debug
I ut.go.debug:nnn
!
```

decatalog

removes the compiled FlashBASIC object code of the specified item(s) from the DICT of the specified file, and removes the corresponding verb entry (or entries) from the MD of the current account.

A program does not have to be previously cataloged to be decataloged. If the program verb does not exist in the master dictionary, "decatalog" still deletes the object code.

Once a program item has compiled successfully, an "object-code pointer-item" is placed in the dictionary level of the file. This "pointer" item defines the virtual location where the object code is actually stored. The name of the object pointer item is the same as that of the source program itself. It has the following format:

id program.name

attr 1 first frame of contiguous block of frames reserved for object item.

attr2 number of contiguous frames used to store the object.

Syntax

decatalog file.reference itemlist*

Example

```
decatalog bp enter.supplier
[242] 'enter.supplier' decataloged!
```

define-terminal

displays the available terminal (driver) definitions and allows changing existing definitions or adding new ones.

Each terminal definition is an item in the "devices" file with the terminal name used as the item-id.

The definitions that are marked with an asterisk (*) are also stored as items using the term type codes as the item-id.

These item-ids are used by the "term" verb to set up terminal characteristics.

In menu options 1 and 2, the name of the terminal is requested. If the definition for the specified terminal is on file, the definition is displayed. If it is not on file, the option to create a new terminal definition is offered.

In menu 3, a prompt requesting confirmation of the deletion displays.

In menu 4, The terminal name is requested, the corresponding item in the devices file is flagged as the selected term type, and the menu redisplays.

In menu 5, the terminal name is requested, the flag marking the corresponding item is deleted, and the menu redisplays.

Example

```
define-terminal
System Cursor Definition Utility
The following terminals are defined in the devices file.
Terminals marked with an asterisk (*) are selected to be
```

```
included in your system cursor definition.
*A ADDS580     G GTC          M MIME         *W wy-50-link
*A ADM-11      *G IBM3161     *N WY-100      *W wy-50132
A ADM3A        *G IBM3164     *P PERTEC      *W WY-60
A CIE-ANT      *G mn3161      *Q QVT102      *W wy50-132
*B AMPEX210    *G mn3161-132  *R R25         *W WY60
B BEEHIVE      H HONEYWELL    S ansi.c       *W wy60.1
*B IBM3151     I AT.MON       *S SOROC       X DATAGRAPHIX
*B ibm3151-132 I IBM          T TEC          *x xterm
*B mark        I IBM3010      T TV920        x xterm-dim
*B mn3151      *I MM-MON      *T TV950       Z ANSI
C DTC          *I port-mon    *V VP          Z att605
*C VT-52       *J VT100       V VP-60        Z att705
*D DATAMEDIA   J vt100-dim    *V VP-A2       Z HFT
D dg           J vt320        W link         Z rt.console
E EMULOG200    *K prism       *W link125     Z RT.MON
E EX-34        *L LSI         *W w           Z Z
*F TV910       *M AMPEX-D80   *W WY-50
*G g           *M AMPEX.D80   *W wy-50-132
1)   Create Terminal Definition
2)   Modify Terminal Definition
3)   Delete Terminal Definition
4)   Add Terminal To Selected Definitions
5)   Delete Terminal From Selected Definitions
EX   Exit DEFINE-TERMINAL To TCL
Enter Selection (1-5) or EX :
Modify Terminal
Enter Existing Terminal Name To Be Modified ?prism
Terminal: prism
1.  Term Type.............................. K
2.  Terminal Description................... Prism-IV
3.  Backspace and Screen Size.............. 8,80,24
4.  Cursor Address Code.................... A
5.  @(X,Y) Cursor Addressing............... CHAR(11) Y CHAR(16) X
6.  @(-1)  Clear Screen & Home............. FF SOH
7.  @(-2)  Cursor Home..................... SOH
8.  @(-3)  Clear to End of Page.......... ESC "J"
9.  @(-4)  Clear to End of Line.......... ESC "K"
10.  @(-5)  Start Blink................... ESC "B"
11.  @(-6)  Stop Blink.................... ETX HEX(20)
12.  @(-7)  Start Protected Field.........
13.  @(-8)  Stop Protected Field.........
14.  @(-9)  Cursor Back.................... NAK
15.  @(-10) Cursor Up...................... SUB
16.  @(-11) Enable Protect Mode..........
17.  @(-12) Disable Protect Mode.........
18.  @(-13) Start Reverse Video........... ETX "D"
19.  @(-14) Stop Reverse Video............ ETX " "
20.  @(-15) Start Underline............... ETX "0"
21.  @(-16) Stop Underline................ ETX "" ""
22.  @(-17) Slave On....................... ESC "R"
23.  @(-18) Slave Off...................... ESC "T"
24.  @(-19) Cursor Forward................. ACK
25.  @(-20) Cursor Down.................... LF
26.  @(-21) Graphics Character Set On.....
27.  @(-22) Graphics Character Set Off....
28.  @(-23) Keyboard Lock.................
29.  @(-24) Keyboard Unlock...............
30.  @(-25) Control Character Enable......
31.  @(-26) Control Character Disable.....
32.  @(-27) Write Status Line.............. HEX(E3)
33.  @(-28) Erase Status Line.............. HEX(E5)
```

```
34.  @(-29) Initialize Terminal Mode......
35.  @(-30) Download Function Keys........
36.  @(-31) Non-embedded Stand-out On.....
37.  @(-32) Non-embedded Stand-out Off....
38.  @(-99) Embedded Visual Attributes?...
39. @(-100) Half Intensity................
40. @(-101) Full Intensity................
Is table for terminal prism correct? y
"TYPE" is a single letter which identifies the terminal to the
system. The terminal-type field in the terminal table corresponds to
the single-character value typed in at the 'term' command.
"SCREEN SIZE" is the size of the display screen (in columns by rows).
It is entered as cols,rows. If a FlashBASIC "print @()" command
exceeds these values, the maximum allowable value will be used.
"CURSOR ADDRESS CODE" is usually a series of characters used to
denote cursor addressing, followed by the row and column of the
position. Sometimes the row and column are in binary, sometimes not.
Sometimes it requires column and row. Others use an ASCII code in
place of binary row/column addresses. Choose the method your terminal
manufacturer uses (NOTE: 'X' is the X-axis, 'Y' is the Y-axis):
"A{+}"  ADDS-type addressing:
row = char(Y+64)
col = char((int(X/10)*6)+X)
"L{+}"  Lear-Seigler addressing:
row = char(Y+32)
col = char(X+32)
"T{+}"  TEC addressing:
row = char(-(1+Y))
col = char(1(1+X))
"H{+}"  Hazeltine addressing:
row = char( X )
col = char( Y )
"D{cr}{+}"  Decimal addressing. It outputs a 1-3 digit binary code
(each) for row and column. "c" specifies the number of digits to
output for the column (1-3); "r" specifies the number of digits to
output for the row (1-3). If used, the code will force leading
zeroes. A "D" alone generates "floating" decimal numbers from 1 to 3
digits.
row = char(x)  as 1-3 digits
col = char(Y)  as 1-3 digits
All cursor addressing codes may be followed by a '+', which adds one
the row and column addresses before generating the address codes, to
take care of the terminals whose addresses starts at "0,0" instead of
"1,1".
"@(X,Y) CURSOR ADDRESSING" is the string of control and escape codes
needed to tell the terminal that a row and column address is coming
up next. For example:
ESC "=" Y X
is a cursor-positioning code for a Lear-Siegler terminal. The Y  and
X will be replaced by the proper ASCII codes (and is determined by
the cursor address code above).
"@(-1) .. @(-300)" are explained in the section describing "@
functions".
Entering values for the codes:
ASCII codes may be entered in one of several ways:
ASCII strings
ASCII characters are enclosed in double-quotes i.e. "T02".
ASCII control codes
ASCII control codes may be entered using the 2 or 3-letter word which
represents the unprintable ASCII control code:
"NUL" = 00   "BS" = 08  "DLE" = 16  "CAN" = 24  "SP" = 32
"SOH" = 01   "HT" = 09  "DC1" = 17   "EM" = 25
```

```
"STX" = 02   "LF" = 10  "DC2" = 18  "SUB" = 26  "DEL" = 127
"ETX" = 03   "VT" = 11  "DC3" = 19  "ESC" = 27
"EOT" = 04   "FF" = 12  "DC4" = 20   "FS" = 28
"ENQ" = 05   "CR" = 13  "NAK" = 21   "GS" = 29
"ACK" = 06   "SO" = 14  "SYN" = 22   "RS" = 30
"BEL" = 07   "SI" = 15  "ETB" = 23   "US" = 31
ASCII character value in decimal
The decimal ASCII value may be entered by typing the word "CHAR(",
the decimal ASCII value (1-255), and a closing parenthesis ")", i.e.
CHAR(27)
ASCII character value in Hex
The hexadecimal ASCII value may be entered by typing the word "HEX(",
the 1-byte hex value (00-FF), and a closing parenthesis ")", i.e.
HEX(1B)
String function
Strings may be entered as a FlashBASIC string function, i.e.
STR("*",5) or STR(CHAR(12),X)
Cursor Address variable
There are three cursor address variables available: X, Y, and Z. They
cause the specified address (byte or decimal string) to be inserted
into the control string at the specified position.
X Contains the Column.
Y Contains the Row.
Z Contains the previous Row referenced in an @(X,Y) code, or zero if
the last reference was an @(-1) or @(-2).
Any combination of the above may be used on a line. Different codes
are separated with a space (not a comma), i.e.
HEX(1B) SOH "1TS04" or
CHAR(27) SOH "1TS04" or
ESC SOH "1TS04"
There are other codes which have been defined for use, but do not
show up in the maintenance screens. The "dm,devices,devices
<item-id>" item will have to be edited directly to insert these
codes if they are needed.
There may be other features available only on specific terminals, and
there are no existing "@ function" codes for them, such as "uppercase
on/off". To take advantage of those features:
*  ASCII sequences must be hard-coded into every program (forcing the
use of that terminal forever), or
*  A separate file of control codes is created (like 'terminals
perkin-elmer-bantam'), and referenced instead of the "dm,devices,"
file, or
*  The custom codes are added to the "dm,devices," file at, such as,
@(-1001) and beyond.
```

define-up

provides a front-end for defining a keyboard for the Update processor. It builds a keyboard item that is be used by the set-imap program to translate the keyboard keys entered into Update processor commands. It automatically detects whether the keyboard needs the extended character set enabled. After filing the item the "define-up" program then executes a "set-imap terminal-name (c" command to put the new translations into affect.

Syntax

define-up {terminal-name} {(options}

Options

q Quick update of keyboard. Requests definitions of only the most commonly used commands.

Example

define-up marks.terminal (q
This will define or allow modifications to the keyboard item called marks.terminal.

delete

deletes one or more items from the given file.

If "dict" is specified, the dictionary items are deleted. If "dict" is not specified, the data items are deleted.

This command does not delete the file itself.

Syntax

delete file.reference {itemlist*} {(option}

Options

i Suppresses the "item deleted" message. (ap)

Example

```
delete names smith
[435] item 'SMITH' deleted
```

delete-account

deletes an entire account, and all the associated files, returning the space used to the overflow table and removing the account name from the "mds" file.

The "account.name" is NOT optional, and must be provided with the command.

The prompt, "DISPLAY FILES BEFORE DELETING (<Y>,N)? ", appears after invoking "delete-account". A "y" response displays each file in the account,

This should only be executed from the "dm" account with all other users logged off. After all of the files to be deleted have optionally been listed, the operator is asked if the account should still be deleted. This requires a "y" or "n" response prior to deletion.

Syntax

delete-account account.name

Example

```
delete-account old.production
```

delete-file

releases all frames used by the specified file to the overflow table, and removes the file.reference from the master dictionary of the current account.

Synonym-defining items ("q-pointers") may not be deleted with this command. The account must possess the actual "d-pointer" to a file to be able to delete the file.

Syntax

delete-file file.reference {(option}

delete-file data file.reference{, file.reference} {(option}

Options

u Unconditionally deletes the file, even if it is open at the time of the delete.

Example

```
delete-file temp
This deletes the dictionary and all data sections from the "temp"
```

```
file.
delete-file data temp
This deletes the "temp" data section from the "temp" dictionary,
leaving the dictionary and any other data sections connected to the
"temp" dictionary intact.
delete-file customers,temp
This deletes the "temp" data section from the "customers" dictionary.
```

delete-index

removes either a specific b-tree index from a given file, or all indices if an asterisk is specified.

"a.code" specifies the "a (algebraic)" processing code of the b-tree index to delete. The processing code must include an attribute number.

Syntax

delete-index file.reference [a.code | *]

Example

```
delete-index entity a1
delete-index invoices a1(tcustomer.file;x;;1)
delete-index sales a1:3
delete-index entity *
```

delete-list

removes a list previously saved with a "save-list" (or "sl") command.

Deletes list from specified filename, otherwise defaults to "pointer-file".

If no listname is provided, the default list name of "%user.name" is deleted.

"dl" is a short hand for "delete-list".

Syntax

delete-list {{file.reference} list.name}

Options

i Suppresses the "item deleted" message. (ap)

Example

```
delete-list invoices
[435] item 'invoices' deleted.
Deletes the specific item "invoices".
delete-list *
Deletes all the lists in the pointer file.
delete-list<return>
[435] Item '%tom' deleted.
This deletes the default list for the current user.
```

delete.account

removes an account.

Syntax

delete.account account.name

dev-att

attaches an unattached port to the current process. This command only changes ownership and is required prior to using the "get" or "send" statements in FlashBASIC.

The user gets exclusive ownership of the attached port. No other processes are allowed to attach the port until the owner releases the port.

The port is detached by using the TCL verb "dev-det", or by logging off.

If the target port is not linked to the current process, the target port must be unlinked by the "unlink-pibdev" command, or use the (U) option to force the unlink.

Attaching to a device the current process is linked to does not remove that link.

Syntax

dev-att port.number

Options

U Unlink the target device if required. Attachment fails on a linked device, unless the current process is linked to it.

Example

```
dev-att 3
Device # 3: Attached to Pib # 0
This attaches port 3 to process 0 (zero).
```

dev-det

detaches the port currently attached from the current process.

The user gives up exclusive ownership of the port previously attached via the "dev-att" verb and makes the port available to other users.

Syntax

dev-det port.number

Example

```
dev-det 3
Device #3: Detached from Pib # 11.
Detaches the port 3 from process 11.
```

dev-make

adds an entity to the system.

This command is normally used to add devices to the system. The most common usage is to create a serial device to be used with the FlashBASIC GET and SEND statements, even if there is no D3 process connected to the device.

type Entity type: 'pib', 'serial', 'tcp', 'udp', 'telnet', 'queue'.

number Entity number. If not specified, the system allocates a number.

arg Optional arguments. There may be more than one '-a' option, or several arguments can be specified within one '-a' clause, separated by commas. The arguments depend on the type (see below).

Arguments:

serial: Unix device name, between quotes.

When the entity is successfully created, an entry is added to the 'devs' file, but NOT to the 'devs,init' file. See the description of the file 'devs' for more details.

Syntax

dev-make -t type {-n number} {-a arg{,arg} {-a arg{,arg} ... }}

Example

```
 dev-make -t serial -n 100 -a '/dev/tty32'
  Create the serial device 's100' associated to the Unix device
'/dev/tty32'. The device is reprogrammed 'D3 way', i.e., in raw
mode, no echo. The hardware characteristics (baud rate, etc...) are
unchanged. When the device is successfully created, the newly created
device can be attached and used with FlashBASIC SEND and GET
statements. The hardware characteristics can be changed with the D3
'set-port' command, using the full entity ID 's100', only once the
device has been attached, because the device is not actually opened
when the D3 entity is created, but only at attach time.
```

dev-remov

removes an entity from the system.

eid is the full entity id (eg 's100') to be removed. If the entity has an output link, or is the destination of any link, or is attached, the removal fails.

It is not allowed to remove special entities, like the 'Byte Bucket' (BB) or the Console (CO).

Removing an entity deletes the entry in the file 'devs'.

Syntax

dev-remov eid

Example

```
dev-remov s100
 The serial device 's100' is removed from the system.
```

df

Invokes "delete-file" command.

diag

invokes the internal system diagnostics utility.

dialer

controls the dialer subsystem which allows transferring data to remote systems over the phone in a batch mode. The section 'dialer, General' explains the principle of the dialer subsystem and the main notions used here.

Using the menus :

Without any argument, a menu is displayed. The following are the valid keys. When indicated, arrow keys can also be used:

RETURN Executes the highlighted option.

'Q' Or ESC. Exit menu and go back to previous menu.

'X' Exit menu and go back to TCL.

n Number from 0 to 9. Select the option number 'n'. '0' is option 10.

letter Select the next menu option starting with the specified lumn. letter (except 'x' or 'q').

Ctrl-N Or down arrow. Move cursor down.

Ctrl-B Or up arrow. Move cursor up.

Ctrl-K Or right arrow. Move cursor right to next column.

Ctrl-J Or left arrow. Move cursor left to the previous column.

Ctrl-X Cancel, in a input field.

Displays :

The screen is divided in an upper section for menu display and a lower section for messages, help screen and user prompts.

Installation :

The first time the system is installed, the user is required to enter a local system name and the time zone the system is located into:

Setup Local Host Name

Local host name :

Confirm (y/n/q) :

Enter any name, up to 8 characters and hit return. This name is also shown by the TCL command 'node'.

Setup Time Zone

1 UK

2 Azores

...

Confirm (y/n/q) :

Select the appropriate time zone and hit <RETURN>.

The remote sites and serial devices should then be set using the 'Setup Menu' described later.

Main Menu :

The main menu operation is used for normal operations. The options are:

1 Queue status

Display the status of the queues to all systems (system name, number of queued jobs and description of the next job) or of a specific system (detailed description of each job).

2 Device status

Display the last messages logged by a specified IO daemon. The messages are displayed most recent first.

3 Start dialers

Start the dialer IO daemons. This command can also be executed from TCL to be used in macros or PROCs. One IO daemon is started for each defined device.

4 Stop dialers

Stop the dialers IO daemons. This command can also be executed from TCL to be used in macros or PROCs. When a IO daemon is in the middle of a call, it will not respond to a stop request.

5 Setup

System setup and test. See the section 'Setup Menu' below.

6 Check conflicts

Display any pending conflicts between changes made to the local data base and changes received from remote systems. See the section 'Resolving conflicts' below.

7 List permanent log

The permanent log contains all messages logged by any IO daemon. This log is not cleared automatically. The messages are displayed the most recent first. Use the functions keys shown in the title to navigate though the log. Hitting a colon (':') displays a command menu which allows searching for special text and start the display at a specified time and date.

8 Messages

Displays the messages sent by other systems, using the command 'msg' of the dialer TCL verb (see the section 'TCL Interface' below), or by the dialer subsystem itself. The list of all the pending messages, sorted by user and by time/date is displayed, with the destination user and the subject. The help section shows more information about each message. When selected, the first few lines of the messages are displayed and the user can P)rint it to the currently assigned form queue, D)elete it, or Q)uit to leave the message. The dialer subsystem generates 'Notice mail' messages in case of serious problems. The messages can also be read from TCL with the 'mail' command. See the section 'TCL Interface' below.

Setup Menu :

The setup menu allows defining the remote systems and devices used by the dialer subsystem, and perform accessory functions, like testing the communication, purging the queue, etc...

1 Local Host name

Define the name of the local system. The local system name must also be declared in all the other systems which can accept messages from this local system. This mechanism is required for security reason, to ensure that messages are properly authenticated.

2 Remote system

Define a remote system. Remote systems must be defined so that they can be called and to be able to accept messages from them. A submenu displays the list of the currently defined systems, or 'New System' to create a new entry. The following elements must be provided:

System name

Any string of up to 8 characters. If the system has been defined already, changing the name creates another system (it does not rename the specified system). This can be used to duplicate a system definition.

Calling Schedule

Define for each day of the week, if and when the system can be called. The System Administrator should choose a window time to get the cheapest rates, when applicable, and, when many systems are calling one central site, try to spread the calling window to reduce the risk of collisions. Within an allowed schedule, the call is placed within a 'few minutes' of the specified time. In case of problems, the system will retry no more than three times to establish the communication. The syntax for the schedule is as follows:

never

Do not call the system that day. If all days are set to 'never', the system will never be called. However, data CAN be transmitted to this system, but only when IT calls in.

any (or empty string)

Call at any time, as soon as data is queued for transmission. The actual transmission time will be within the next minute. This should be reserved to leased lines only, since it would be very inefficient to dial for each update.

HHMM-HHMM

The first four digits specify the starting time of the calling window, in 24 Hr format. The second set specifies the ending time. For example '2000-0100' means from 8:00 p.m. to 1:00 a.m.; '0100-0530' means from 1:00 a.m. to 5:30 a.m.

*HHMM

Define a periodic calling schedule. A call can be placed every HH:MM. For example, '*0100' means that a call will take place at 1:00 a.m., 2:00 a.m., etc.. during the whole day. It is not advised to use too small a period. One hour should be a minimum, except in some very exceptional cases.

Phone number

Up to 4 phone numbers can be specified. In case of failure, they are tried in the given order. The syntax of the phone number must be compatible with the modem and, possibly, should include any prefix, wait, extension, etc...

Device

Specifies which devices can be used to establish communication. If none is specified, the system will select the first available device. See the section below about creating serial devices on Unix implementations. The syntax of the device number is:

S nnn

Dedicated serial device number 'nnn'. The system assumes it has exclusive access to this device.

nnn

Use the serial device normally associated to the D3 process number 'nnn'. When the serial device is needed, the D3 process will be 'unlinked' to take control of the device, and linked back on to it when the transfer is complete.

Permissions

Defines the operations the specified system can or cannot do when calling IN. Enter 'y' or 'n' to each field:

Call

Defines if the system is allowed to call in at all. If set to 'n', the local host will hang up if receiving a call from this remote site.

Exec

Defines whether the remote system is allowed to execute commands on the system. Commands are run by the IO daemon itself and should be short.

Upd

Defines whether the remote system is allowed to update the local system's data base.

Adm

Defines whether the remote system is allowed to perform remote maintenance operations.

3 Devices

Define a serial device. A submenu displays the list of currently defined devices or 'New device' to create a new entry. The following elements must be provided.

Device id

Device name. For a dedicated device, the id must be prefixed by a 's'. For a shared device, enter the D3 process number which is normally linked to the device to use. Do not use device 0. If shared, also specify the take over time below.

State

Enable or Disable. If disabled, no IO daemon is started and this device will not be used.

Type

Direct or Modem. A direct device is assumed to be constantly connected to the target system through a leased line, for example. Only one system can be reached through this device. Make sure the system definition is consistent with this setting.

Dialer name

Defines the name of the dialer if the device is a 'modem' type. By default, the dialer is 'hayes'. This name is the item-id of the BASIC subroutine in the 'dm,dialers' file which handles the dialog with the modem.

Mode

Defines whether the device is Input only (it can only receive calls), Output only (it does not accept any incoming calls), or both.

Settings

Defines the serial port setting. Note the device MUST support 8 bit characters.

Take over time

Defines the time window during which the dialer will take over a serial device normally shared with a regular D3 process (i.e., a device which does NOT start with an 's'). The syntax of the time is identical to the one described for the calling schedule on a remote system. If a time interval (eg '2200-0700') is specified, the device will keep the device attached for the specified duration. This should be the normal setting, for example to specify a time out of normal opening hours when nobody would dial on the system to logon to D3. If a periodic schedule (eg '*0030') is specified, the device will attach to the device for a period, then detach for the same period, then re-attach, for a new period, etc... Specifying 'any' or leaving this field blank, effectively makes it dedicated since the IO daemon will attach permanently to the device.

IMPORTANT: Changing the take over time schedule requires to stop and restart the dialer subsystem for the change to become effective.

User reset

Optional command string to send to the modem to reset it to a user defined state when the dialer daemon either terminates or relinquishes the shared device. A carriage return is appended to the string. If left empty, a modem dependent string is used ("ATZ" in case of the Hayes dialer). If this command fails (eg does not respond 'OK'), a factory reset is done.

4 Delete remote system

Delete the definition of a remote system.

5 Delete device

Delete the definition of a device.

6 Test access to system

Attempts to establish a call to the specified system. This option can be used to test a new installation, or to make sure a remote host is reachable. This option dials out, establishes the communication and sends 10 test messages to the remote system which must be powered up and have an appropriate IO daemon started. The IO daemons MUST not be started on the local system else 'no device available' message will be issued.

7 Set local time zone

Defines the time zone the local system is in.

8 Set system map

Defines where the updates to a system are to be sent. This option shows a sorted list of the accounts on the system (up to 256 accounts can be shown). Select the account to be set up, or 'ALL ACCOUNTS'. Note the 'dm' account is never selected. Then select to update either 'All files' or 'Select files or a select list of files. When the 'Select Files' option is chosen, the list of files in the account is displayed; Hit the <space> bar to select the file(s) to be changed and hit <return> when all the files have been selected; a '*' is displayed in front of the selected files. The list is saved in a list which can be used as input to the 'Select list of files' option. The user can elect to post updates to both the data level(s) and the dictionary, to the dictionary only, or to the data level only. An option 'REMOVE CALLX' allows suppressing the posting of the updates on the entire selected account. Finally, the list or remote system(s) where the data is to be sent must be entered, as well as the optional account name on the remote system. If an account name is not specified, the remote account is assumed to be the same as the local account name. Up to four remote systems can be entered. An remote system cannot be specified twice in the list. Confirmation is required. This option updates the D pointers of the dictionaries and/or data levels of the selected account(s) and files to insert or remove, as appropriate, the CALLX correlative to perform the update posting.

This menu option can be used as long as there are less than 6 target system names. Otherwise, it is more convenient to use the TCL form (see the section 'TCL interface' below).

9 Purge queued updates

Purge the queue of data to be transmitted to either all systems or a select system.

10 Clear permanent log

Clear selected messages or all messages from the permanent log. A sub-menu allows selecting messages older than one week, one month, or all messages.

11 Connect to device

Connect directly to the device to send it some simple commands, like resetting the modem. The device must not be attached. Note that only simple commands can be sent to the device (like a reset). Dialing to a system may be difficult.

12 Submit remote command

Connects to a remote system and execute a command on it. The local site must have been declared with 'sysadmin' privilege n the remote system. This option prompts for the remote system to dial on and a command menu: 'Submit TCL command' to execute any TCL command on the remote; 'Terminate remote' to force the IO daemon on the remote system to terminate. The submit command prompts for a command, dials to the remote system, submits the command and disconnects without waiting for the end of the command. The terminate option should be used ONLY if the device on the remote system is a shared device. This allows a user to free the device to be able to do a remote logon.

Resolving Conflicts :

This option from the main menu examines the conflict file and shows any conflict still not resolved. The first menu selects all the conflicts and shows the account, file and item-id. The help message in the message section gives more information about the conflict. The 'Check conflict' option requires that the System Administrator has good knowledge of the data base

structure. Only raw data is presented. A more advanced conflict resolution requires understanding the data base organization and cannot be accomplished by a general purpose tool. The following section 'Conflict Data Format' details the stucture of the data stored in case of a conflict so that an application dependent tool can be developed.

When a conflict is selected, the following information is displayed:

From: sys Date: 12/14/94-09:18 Cause: Conflict change

Acc : dev File: tmp Item : x

sys!dev,tmp, x LOCAL!dev,tmp, x

---------- ----------

In this example, 'sys' is the name of the remote system the conflicting changes is coming from, the time and date are the local time and date, the cause is a short description of the conflict, 'dev' is the local account, 'tmp' the file and 'x' the item. The left hand side of the display shows the remote file information and the right hand side the local one.

The screen is then divided into two columns showing the attributes which are different on the local system before and after the changes from the remote system have been applied.

The user is then prompted to take an action:

Use data from system 'xxx ' 1

Use local data and update remote 2

Quit (leave conflict unresolved) ..... Q

Cancel conflict. Leave data alone C

Show Next attributes in item ..... N

Show top of item ..... T

Select:

1 Use the data received from the remote system to overwrite the local data.

2 Keep the local data and copy the local data to the remote to force it to use the local copy. Note that the copy is just enqueued. The system will wait until the next calling time to actually copy the data. More changes can be made to the same item before the copy is performed. The item which will be transmitted will the item at the time the transfer takes place, not the item the way it is at the conflict resolution time.

Q Quit. Leave the conflict unresolved.

C Cancel the conflict. The data is left as is on each system, CREATING INCONSISTENCIES ON THE DATA BASES. This option should be used with extreme caution.

N Show the next attributes.

T Go back to the top of the item.

Conflict Data Format :

The conflicting data is stored in the file 'dialer.log,conflict'. Each conflict is represented by 3 items:

- Conflict definition item. The item-id is a unique time date stamp. It contains the following attributes (defined in 'dm,bp,includes dialer.inc'):

CFLCT$DATE : Local conflict date.

CFLCT$TIME : Local conflict time.

CFLCT$ACCOUNT : Account on LOCAL system.

CFLCT$DICT : 'dict' if a dict, or ''.

CFLCT$FILE : File name.

CFLCT$ITEMID : Item-id.

CFLCT$SYSTEM : Remote system name.

CFLCT$CODE : Reason for the conflict:

CFLCT$NOFILE : Missing file

CFLCT$BADDIFF : Bad difference string

>0 : Line# in the diff string (see below)

CFLCT$REMACC : Account on REMOTE (source) system.

- 'Old' item. The item-id is the concatenation of '*OLD*' and of the conflict id it depends on. This item contains the item coming from the local system at the time the conflict occurred.

- 'New' item. The item-id is the concatenation of '*NEW*' and of the conflict id it depends on. This item is not really the 'new' item: it contains the 'difference string' describing the changes applied to the remote host. The attribute CFLCT$CODE contains the attribute number in this 'new' item on which the conflict was detected.

The 'difference string' is composed of a series of commands which describes the change to be applied. Each command is composed of one or more attributes, and start by a one letter code. The valid commands are:

Ln Original number of attributes in the item 'n'.

Cn{]m} Change attribute command starting at attribute 'n'. Change 'm' (default 1) attributes. This command is followed by 'm' pairs of attributes 'oldvalue/newvalue'.

Vn{]m} Change value command starting at attribute 'n'. Change 'm' (default 1) attributes. This command is followed by 'm' triplets of multi-valued attributes built as follows:

valnum] valnum] ...

oldval] oldval] ...

newval] newval] ...

'valnum' is the value number which is modified. 'oldval' is the old value. 'newval' is the new value. If the number of old values is less than the number of new values, it indicates that the values were added. If the number of old values is greater than the number of new values, it indicates that the values were removed. In the example shown in the section 'dialer, General', the value change command would be:

V2

2]3

bb]bbb

bbb

An{]m} Add attribute command starting at attribute 'n'. Change 'm' (default 1) attributes. This command is followed by 'm' attributes which contain the added attributes. If the attribute 'n' is not empty, the new attributes are inserted before it.

Dn{]m} Delete attribute command starting (including) at attribute 'n'. Delete 'm' (default 1) attributes.

Merging the changes described by a difference string into an item is accomplished by the FlashBASIC program in 'dm,bp,includes merge.sub'. For example:

```
* Get the old item
read m$olditem from conflict,'*OLD*':id
* Get the difference string
read m$diff from conflict,'*NEW*':id
* Merge the changes in
* m$nocheck = 1 ;* Set to force merging
include dm,bp,includes merge.sub
if m$code=0 then
* OK
end else
* Error
* Changes cannot be merged because of a
* conflict. (use m$nocheck=1 to force merge)
end
```

Note the include is not a subroutine. It is a fragment of code which can be included in-line. See the include itself for a description of the interface.

Other includes of interest are 'dm,bp,includes sdiff.sub' which builds a difference string between two items and 'dm,bp,includes sdiff2.sub' which merges (combines) two difference strings into one, cumulating the changes.

Creating Serial Devices on D3 Unix :

The D3 Unix implementations have the possibility to create serial devices dynamically which are not linked to any D3 process. These devices are accessible only through FlashBASIC GET and SEND statements. It is convenient to create devices this way by inserting in the user coldstart macro the "dev-make" statements required to create these devices (see the documentation 'dev-make, TCL'). For example:

```
dev-make -t serial -n 120 -a '/dev/tty9'
```

creates a device 's120' associated to the Unix device '/dev/tty9'.

TCL interface :

Some functions are accessible from TCL, to be inserted in macros:

start Start the IO daemons.

stop Stop the IO daemons. When a IO daemon is in the middle of a call, it will not respond to a stop request.

queue Display the queue(s). The (V) option shows detailed information about the queues instead of just the number of queue entries.

purge Purge a queue. The system must be specified by a clause system=name . NO confirmation is required.

status Display a device last message. The device must be specified by a clause device=name .

mail Read the messages. This option has the following arguments: user=user.name . If 'user=' is omitted, the current user name is used. If the (Q) option is used, only the number of messages is displayed. A prompt '*' is displayed. The commands are:

n : Show the message number 'n'

L : List the messages numbers and subject.

Pn : Print the message number 'n' on printer.

D[{n}|*] : Delete the message 'n', or the last message message displayed or printed, or all messages ('*') for the selected user.

Q : Return to TCL.

msg Send a message to a user on another system. The messages are stored in the dialer subsystem mail file, and can be examined with the option 7 on the main mail menu. This option has the following arguments: user=user.name text=message subject=subject.text . When there is a space in a field, the data must be enclosed in quotes. If 'user=' is omitted, the current user name is used. If 'subject=' is omitted, the user is prompted for an optional one line subject. If 'text=' is omitted, the user is prompted for several lines of text, terminated by one line containing only a period ('.'). If the text of the message is null, no message is queued.

sysmap Set the system map. The option account=accname defines the account to be changed. The option system=[*list|name{,name,...}] defines to which system(s) the updates must be transmitted. '*list' specifies that the list of system(s) is in the select list 'list'. The optional option remacc=accname{,accname,...} specifies the account name(s) on the remote system(s). If this last option is not specified, of if an account name in the list is missing, the account name on the remote system is identical to the local account name. There must be as many remote account name s specified as there are remote system names. The optional option files={-}{dict|data}[*|-|*list|name{,name,..}] specifies the files to be changed. If this last option is not specified, the file sin the account are not changed, only the system map is affected. The '-' sign specifies that the CALLX correlative must be removed from the specified files. If the key word 'dict' or 'data' is specified, then only this element of the file is affected. Note a space must be present, therefore the whole argument must be surrounded by quotes. The form 'files=*' specifies all files in the account. The form 'files=*list' specifies that the list of files is contained in the select list 'list'. The form 'files=name{,name...}' gives an explicit list of file names. See the examples below for practical examples.

nopost Suppresses the posting of updates on the current port. This option remains in effect on the current port until the 'dialer post' command is issued or the process logged off. This option can be used to manually update a local file without the changes being transmitted to the remote sites.

post Enable the posting of updates on the current port. Thjis option is on bny defasult when the process logs on.

Files

The file 'dialer.log' used by the dialer subsystem is created automatically in the 'dm' account. Its data levels are:

billboard

System wide file keeping track of the posted updates.

conflict

Contains the conflict information

devices

Valid devices list.

dialer.log

Status of the IO daemons.

log

Permanent log.

mail

File containing messages for the System Administrator, like error notice, acknowledgments, submit results, etc...

map

System map.

queue* system

Queue to the system system .

spool

Spooled data.

systems

Remote system definitions.

In case of incidents, files 'dialer.dump.mm/dd/yy' are created, with the same data levels as the 'dialer.log' file (except the permanent log). These dump files contain images of the dialer.log files at the time of the incident. An item 'Reason' in the dictionary explains the cause of the dump.

Syntax

dialer {cmd} {device=name} {system=[*list|name{,name...}} {user=name} {subject=descr} {text=message} {account=accname} {files={-}[*|*list|name{,name,..}} {remacc=name,{name,,}} {(options}

Options

Q Quiet. Suppress all messages. On the 'mail' command, this option just shows the number of messages, if any.

V Verbose. Display more information.

Example

```
Examples :
dialer start
  Start the dialer subsystem from TCL
dialer queue (v
  Display a detailed status of the queues
dialer status device=s120
  Display the last messages produced by the IO daemon asociated to
the device 's120'.
```

```
dialer stop (q
  Stop the dialer subsystem, suppressing all messages.
dialer purge system=seattle
  Purge the queue to the remote system 'seattle'.
dialer msg system=dev user=bob subject="Down time" text="We will shut
down tomorrow at 12:00"
  Queue a message for the user 'bob' on the system 'dev'.
dialer sysmap account=bob system=dev,prod,back remacc=bob2,,bob3
files=*
Defines the system map for the account 'bob'. Updates will be transmitted to
the systems 'dev', 'prod' and 'back'. The remote accounts on these systems are
respectively, 'bob2', 'bob' (same as on the source machine) and 'bob3'. All
files, dict and data, are affected.
dialer sysmap account=bob files=-*
Defines the system map for the account 'bob'. The CALLX correlative is removed
from all files. Note it is not necessary to specify the remote system when
removing the CALLX correlative.
dialer sysmap account=bob system=dev files='data *'
Defines the system map for the account 'bob'. Updates will be transmitted to
the system 'dev'. The remote account is the same as on the source machine. All
files are affected. Only the data sections are changed, not the dictionary.
Note the required quotes.
dialer sysmap account=bob system=dev files='data names, address,zip'
Defines the system map for the account 'bob'. Updates will be transmitted to
the system 'dev'. The remote account is the same as on the source machine.
Only the data sections of the files 'names', 'address' and 'zip' are affected.
Note the required quotes.
dialer sysmap account=bob system=dev files='dict *myfiles'
Defines the system map for the account 'bob'. Updates will be transmitted to
the system 'dev'. The remote account is the same as on the source machine.
Only the dict section of the files specified in the select list 'pointer-file
myfiles' are affected.
dialer sysmap account=bob system=dev remacc=bob2
Defines the system map for the account 'bob'. Updates will be transmitted to
the system 'dev'. The remote account is bob2. Only the system map is changed.
No file is affected.
```

dialer-copy

queues one or more items for copy to a remote system through the dialer subsystem.

The data is copied on the remote system into a file which has the same name as the source file. The item-ids are identical. The copy is unconditional.

"sysname" is the remote system name to which the data will be copied. The remote system must be known to the dialer subsystem. See the section 'dialer, Definition', and 'dialer, TCL' for more information on the dialer subsystem.

"account" is the optional account name on the remote system. If not specified, the account where the file is located on the local machine is used.

The copy does not actually take place at the time this command is issued. Requests are queued to the dialer for a batch copy at the time specified by the System Administrator in the remote system description.

Without the (C) option, the item is left in the original file. Therefore, the item must not be deleted after this command has dev'. been issued, else the actual transfer operation will not find the item and will fail. The (C) option copies the item to a spool file to allow deleting or modifying the item.

Syntax

dialer-copy {dict} file.reference itemlist {(options}

to : sysname{!account}

Options

B Suppresses the message 'n items copied'

C Copy items to a spool file.

I Supresses item-id displays during the copy.

Example

```
dialer-copy bp demo
to system:dev
  Queue the item 'bp demo' for copy to the remote system 'dev'.
select bp = 'test]'
dialer-copy bp (c
to system:dev!dm.new
  Queue a selected list of item(s) for copy to the remote system 'dev', first
copying the item to a spool file, so that they can be deleted or modified. The
data is copied into the file 'bp' in the account 'dm.new' on the remote
system.
```

disc

disconnects the current process from the D3 virtual machine.

If the process was started from shell (i.e., by typing "ap"), control is returned to shell.

If the process was started automatically, in the case of a turnkey system, the User process is restarted immediately. In this case, the "disc" command has no action. However, a different PID is allocated to the process.

If the xon/xoff protocol was changed with the "xonoff" verb, the new setting might be lost. (see "xonoff").

When reconnecting by typing "ap" at shell, the process is reconnected to the virtual machine.

The "exit" macro executes a "disc (f". Trying to shutdown a system where there is one or more processes disconnected with the "n" option hangs. In fact, it times out after a few minutes. Before doing a shutdown, reconnect these processes and disconnect them with the "f" option.

This command closes all Unix files. Therefore, it should not be used in a FlashBASIC EXECUTE statement in a FlashBASIC program which has Unix files opened.

Syntax

disc {(option}

Options

f Logoff before disconnecting. This option is the default.

n Stays logged on to the D3 account before disconnecting. Processes disconnected with the "n" option remain logged on to the D3 account and therefore show in the (D3) "listu" command. Processes at lower levels are NOT wrapped up.

display

outputs text on the terminal.

May be used in macros and PROCs to display messages to the screen.

If "message.text" is terminated by a plus sign (+), the trailing carriage return is suppressed.

"message.text" is either regular text, the following special characters, or any combination of them:

@(x{,y})

Positions cursor at the specified "x,y" coordinates where "x" indicates the row position and "y" indicates the column position.

@(-x)

Generates the corresponding video attributes. The values of 'x' are the exactly the same as the values of the FlashBASIC "@ function".

@({x}N)

Generates 'x' carriage returns. If 'x' is omitted, only one carriage return is generated.

@({x}B)

Generates 'x' bells. If 'x' is omitted, only one bell is generated.

@(O[n|.x])

Generates one character whose decimal representation is 'n', or hexadecimal representation is 'x', preceded by a period. Valid values are from 0 to 254 decimal (x'00' to x'FE').

`command`

The result of the execution of "command", between back quotes, is substituted in place of "command".

Syntax

display message.text {+}

Example

```
display I am `who` !
  Displays:
    I am 0 joe pa !
display @(-1)@(-13)MAIN MENU@(-14)+
  Clears the screen, displays the highlighted text, and suppresses
the trailing form feed.
display Sending DEL: @(O.7F)
  Send the the DEL character.
```

div

divides the first integer number provided by the second integer number provided and displays the result as an integer quotient and remainder.

Syntax

div number number

divd

invokes the "div" verb. It divides the first integer number provided by the second integer number provided and displays the result as an integer quotient and remainder.

Syntax

divd number number

divx

divides the first hexadecimal number by the second hexadecimal number and displays the result as a hexadecimal quotient and remainder.

Syntax
divx hex.number hex.number

dl

Invokes "delete-list" command.

dm

logon macro for the Data Manager account, issues an "exec abs" command each time someone logs to the dm account.

download

downloads font files to a LASER printer.

Syntax
download fonts,descr {itemlist*} {(options}

Options
(integer number). Specifies font number.

? Displays command usage.

c Displays output to crt.

h Displays some help.

p Makes font permanent.

s Suppresses any system messages.

t Displays a sample printout of printable ASCII characters.

x Displays output in hexadecimal.

z Prevents output of font prefix and suffix, just provides binary output. For debugging purposes only.

The "dm,fonts,desc" file contains the font description items.

dtr-off

toggles off data terminal ready.

If the port is not specified, the current port is used.

Syntax
dtr-off {port.number}

dtr-on

toggles on data terminal ready on another port, or the current port if none is specified.

Syntax
dtr-on {port.number}

dtx

converts a given decimal (integer) number to its corresponding hexadecimal equivalent.

Syntax
dtx number

dump

displays or prints the contents of one or more frames in ASCII ("character") or hexadecimal format.

Frame-id's (FID's) preceded by a "." are considered hexadecimal references. A range of fids may also be specified.

Note: the "dump" command uses the same options interface as the Spooler. This means that the options and arguments are NOT preceded by a "(" (left parenthesis).

Syntax

dump decfid{-decfid} {options}

dump .hexfid{-.hexfid} {options}

Options

fid Requests frame by its decimal (integer) address. Preceding the fid reference with a "." (period) designates a hexadecimal address.

a Displays output in EBCDIC format.

c Dumps entire frame, beginning with byte zero.

g Dumps entire group. Frames are traced logically "forward". The dump terminates when the last frame in the logical chain has been found.

l Outputs links only; no data is displayed.

n Activates nopage function on output to the terminal.

p Directs output to system printer, via the Spooler.

u Upward trace; frames are traced logically "backward". The dump terminates when the first frame in the logical chain has been found.

s Dumps just the data bytes in a stream without side baggage such as links, offset, ':', etc. -- just a long byte stream. Works with "x", "g" and "u" options. This is mainly for programs wanting to examine memory in straight byte sequence without having to strip out the extraneous information.

x The frames are dumped in hexadecimal with corresponding ASCII representation along the right side of the output.

Example

```
dump 12345 g
dump l 12345
dump 12345 (lp
dump 12345-12330
dump xp 12345-12330
dump .3039 g        (x'3039'=d'12345')
dump lu .3039
```

echo

toggles terminal character echo on or off.

When terminal echo is off, characters entered from the keyboard are not displayed on the terminal screen.

Example

```
echo<return>    <- enter the command.
who<return>     <- type "who".
```

```
46 rlm krb      <- the output appears.
echo<return>    <- re-enable echo.
who<return>     <- this displays.
```

ed

invokes the line editor for entry or update of any item in the system (i.e. FlashBASIC programs, PROCs, data items, etc.).

Syntax

ed file.reference itemlist* {(options)}

Options

a Activates the assembly formatter. Equivalent to the "as" editor command.

d Allows editing a "d-pointer".

m Activates the macro expansion function. Equivalent to the "m" editor command.

p Directs the output to the system printer, via the Spooler.

s Suppresses the display of line numbers, in normal edit mode, or suppresses object code display when the assembly formatter is "on". Equivalent to the "s" editor command.

z Suppresses the "top" and eoi" messages.

Example

```
ed dm,bp, term-type
```

edit

Invokes line editor.

edit-list

retrieves a previously saved list and enters the line editor (same as "ed" or "edit").

Each element in the list is treated as a line by the line editor; all of the line editor edit commands are valid.

If no file.reference is given, the list is retrieved from "pointer-file".

If itemlist* is not specified, the default list item will be updated: %user, where user is the user-id.

See the "el" command for using the Update processor to update saved lists.

Syntax

edit-list {file.reference} {itemlist*} {(options)}

Options

a Activates the assembly formatter. Equivalent to the "as" editor command.

d Allows editing a "d-pointer".

m Activates the macro expansion function. Equivalent to the "m" editor command.

p Directs the output to the system printer, via the spooler.

s Suppresses the display of line numbers, in normal edit mode, or suppresses object code display when the assembly formatter is "on". Equivalent to the "s" editor command.

z Suppresses the "top" and eoi" messages.

Example

```
edit-list invoices.past.due
edit-list dict customers cust.list
edit-list
```

end

terminates a process on another port, or the current port if a port.number is not specified.

"end" also stops processing at the previous level and returns to the preceding level TCL prompt. However, if there is an active list at the current level, then list list will be cleared, and control will not return to the previous level.

The "end" command can also be used to stop a TCL command that was sent to another line by the "tcl" command, or to terminate a "phantom" job. (See the "z", "zh", "zhs" and "zs" commands).

This command requires a sys2 privilege level.

If the "port.number" and user-id are not specified, "end" stops the process on the current line.

"end" also clears an active list.

Syntax

end {port.number user-id}

Example

```
end
This ends the process at the current level.
end 16 dm
This stops the process executing on line 16, under the user-id, "dm".
select dict entity
[404] 276 items selected out of 276 items.
end
This "kills" an active list.
```

env

see env (Unix)

environ

see environ (Unix)

esc-data

specifies that the code generated by the <escape> key is to be passed through as data.

On D3 Unix implementations, this command does an implicit 'set-esc off' to disable the special Unix related signal handling.

esc-level

specifies that the <escape> key is to push a level when pressed.

On D3 Unix, this command uses "set-esc" to enable the special escape key. Without any argument, the usual ESCAPE value (x'1B' or 27) is assigned to the level pushing function. The optional argument defines an alternate value (see the command "set-esc").

The following arguments are valid on D3 Unix:

".hexnum" Indicates the key to trigger a level push, designated by its corresponding hexadecimal value.

"decnum" Indicates the key to trigger a level push, designated by its corresponding decimal value.

"esc" Defines escape as x"1B", the normal value for escape.

"?" Displays command usage information without changing any existing settings.

Syntax

esc-level [decnum | .hexnum | esc | ?]

Example

```
esc-level
Makes the <escape> key push a level.
esc-level .1a
This causes a <ctrl>+z to push a level. Valid on D3 Unix only.
esc-level 26
This also causes a <ctrl>+z to push a level, but is indicated
by its decimal value. Valid on D3 Unix only.
esc-level ?
Usage: esc-level {decimal|.hexadecimal|off|esc|?}
This example shows the "on-line" help text produced by "?".
esc-level esc
This resets the <escape> key to its "normal" definition.
```

esc-toggle

toggles the function of the <escape> key to either be treated as data ("esc-data") or as a level-pushing mechanism ('esc-level").

estimate-count

Displays a fast estimate of the number of items in a file.

Syntax

estimate-count {file}

Options

displays a fast estimate of the number of items in a file.

If no file-save or simple ACCESS report has ever been run against the file, this command may take a significant amount of time to complete. Otherwise, the result is virtually instantaneous.

exchange

switches item-ids of the two specified items following the file.reference.

Syntax

exchange file.reference item-id1 item-id2

Example

```
exchange bp program1 program2
Item 'program1' exchanged with 'program2' in file 'bp'
Swaps the item-id's for "program1" and "program2". After "exchanging"
the "body" of "program2" is associated with the item-id "program1"
and vice-versa.
```

exec

runs the current process on the specified "abs" area. This has the effect of changing the environment of the D^3 virtual system for this process, until logoff time. At logoff, the "abs" file is automatically reset to the "dm,abs," file.

Syntax

exec file.reference

exit

logs the D3 process off and disconnects it from the D3 virtual machine.

Even if the verb "exit" does not exist, the system will still exit with the proper privileges.

f-resize

Examines existing file modulos for every file found in the "file-of-files" file, and recalculates a new "recommended" modulo for each file.

Files are given a new recommended size according to their statistical information and are given a new modulo based on a file 75% full. The files are not immediately resized. The recommended modulo is stored in the file's file-defining item (d-pointer) in attribute 13.

If a dot "." exists in attribute 13 of the file-pointer, this data file will not be resized.

The complete file resizing process is to:

a) Do a full "save" so all the filenames are in the "file-of-files" file.

b) Perform the f-resize.

c) Do a full "save" of the system (which will save the recommended modulos).

d) Do a restore of the desired account (or of the entire system), and the system will refer to the recommended modulo in re-creating the files, unless "file reallocation" is disabled during the restore.

Syntax

f-resize {(options}

Options

a Resize one account

e Force resize of empty files. As a default, empty files are ignored by the f-resize program.

p Directs output to the system printer, via the Spooler.

r{frame.size} ("r", followed by a frame size) Resizes files according to the frame size specified. If frame.size is not specified, the frame size of the current system is used.

u Specifies that no downsizing will take place.

f.modulo

calculates a file modulo, based on estimated file characteristics.

The f.modulo program prompts for number of items, item-id size, and item size.

Returns a recommended modulo, taking into concern the frame size of the machine and in-line/pointer items.

Syntax

f.modulo

fdisk

invokes a menu to display or alter the fixed disk partition parameters. This is also used to change the active partition prior to rebooting.

Displays the current fixed disk partition parameters and current disk configuration, or changes the active partition.

This information is kept on the Master Fixed Disk Boot Record ("MFDBR"). When the system is booted, the operating system on the partition marked active in the "MFDBR" assumes control of the computer. If no partition is marked active, the system cannot be booted from hard disk.

When fdisk is invoked, a menu with the following options is displayed:

0 Returns user to TCL without executing fdisk.

1 Creates a D3 partition. It displays the current partitions defined in the MFDBR, but does not do anything else. It is provided for compatibility with the PC DOS FDISK command.

2 Changes active partition. This is used to select the partition that contains the desired operating system. The partition that is marked active assumes control the next time the system is booted.

3 Deletes D3 partition. This is used to delete from the MFDBR the reference to the partition that currently has control of the computer (the currently active partition). If there is more than one partition, before the currently active partition can be deleted, another partition must be marked active using option 2. The only partition that can be deleted is the currently active partition.

4 Displays partition data. This is used to display the status of all partitions.

5 Switches and displays parameters. This displays the statistics for the current drive, and if the system supports two drives, gives the option to select one drive or the other.

9 Undoes changes to "MFDBR". This returns the partition settings in the MFDBR to those that were in effect when "fdisk" was invoked.

Example

```
:fdisk
IBM Personal Computer and true compatibles
Fixed DISK setup program version 2.1 PC-AT
(C)Copyright Pick Systems  1985
08:00:00  16 Jan 1997
FDISK Options
Current fixed disk drive: 1
Choose one of the following:
0.   QUIT AND RETURN TO TCL
1.   Create D3 partition
2.   Change active partition
3.   Delete D3 partition
4.   Display partition data
5.   Switch and display parameters
9.   Undo changes to MFDBR
Enter choice: [ ]
IBM Personal Computer and true compatibles
Fixed DISK setup program version 2.1 PC-AT
(C)Copyright Pick Systems  1985
08:00:00  16 Jan 1997
Display partition information for disk 1
# STAT TYPE     START END  CYLN SIZE MB
1   A  GM-AT D3 0001  0612 0612 20.32
2
3
4
Fixed disk space is 614 Cylinders.
                    20.39 Mbytes.
Press ESC to return to FDISK option [  ]
```

fid

displays information about a memory-resident frame.

Syntax

fid frame-id {(options)}

Options

n Outputs report without pausing at the bottom of each page.

p Directs output to system printer, via the Spooler.

Example

```
fid 122
      i i t n b r
w m   o o o o a e
t l f w b p t t u
r o r r u h i c s
e c e i s s a h e
q k f t y h q b d
-----------------
0 1 1 0 0 1 0 0 0
A fwdhq  fwdaq  bckaq  buffid btwa
- ------ ------ ------ ------ -------
0 000000 000000 000000 00007A 400CB800
In the "vertical columns" above each of the bit settings for the
given are descriptions of each bit. Any column with a "0" underneath
indicates that the corresponding value is "off". Non zero indicates
that the value is "on".
"wtreq" indicates a "write-required" state.
"mlock" indicates a "memory-locked" (core-locked) state.
"fref" indicates that this frame has been referenced recently.
"iowrit" indicates that the frame is being written, or is enqueued
for writing.
"iobusy" indicates that this frame is being read from disk.
"tophsh" indicates that this buffer is at the top of the internal
memory hash list.
"notiaq" indicates that this buffer will soon be gone.
"batchb" indicates that this fid was brought into memory by a batch
process.
"reused" indicates that the buffer is eligible for immediate re-use.
"Attr" is the number of virtual registers attached to this buffer.
"fwdhq", "bckaq" and "fwdaq"
   are all pointers to an internal management list.
"buffid" is the fid of the buffer.
"btwa" indicates the real address of the buffer.
```

field

will print out the Nth word in a list of words. Useful when combined with the tcl shell variables.

Syntax

field N list.of.words

Example

```
:field 3 one two three four
three
:field 2 @`count md`
1635
```

file-save

invokes the procedure to perform a full backup of each file in the D^3 file system. This verb is usually invoked from "dm" account.

The operator is prompted for a series of decisions, including:

Is this an Incremental Save (y or <n>)?

Console listing to printer (y or <n>)?

Send statistics report to printer (y or <n>)?

Verify save with 't-verify (e)' (y or <n>)?

Bypass groups with GFE's (y or <n>)?

Do you want to sleep (y or <n>)?

The "file-save" is typically followed by a "t-dump" of the "file-of-files" and generation of the file statistics report.

Is this an Incremental Save (y or <n>)?

An "incremental save" only saves files which have changed since the last full save was done. Incremental saves perform much faster than a "full" save, since there is less to save. The default answer is 'no'. See the "save" command for more information on incremental saves.

Console listing to printer (y or <n>)?

As the save progresses, it shows the current account name and file name. The system then shows the files being saved in the following form:

Tape# file# mds > account > dictionary > data

The display may be directed to the printer by responding with "y", or use the default of 'n' to have it display to the console.

Send Statistics Report to printer (y or <n>)?

As the file-save progresses, it builds a statistics report about the files it saves.

Verify save with 't-verify (e)' (y or <n>)?

After the file-save is complete, the media may be checked for reliability by re-reading it using the 't-verify' command. It is strongly recommended that file-save tapes be verified. If "t-verify" is selected, errors are logged to the "tv.log" file. This prompt also allows additional "t-verify" options to be supplied in the following format:

Verify save with 't-verify' (y or <n>)?y (eilp

The options must be delimited from the "y" or "yes" with a space, a comma, or a left parenthesis. Note that when supplying additional options, the "e" must be specified again if it is desired. Note that by default, the tape is NOT verified against the data on the disk.

Bypass groups with GFE's

Instead of stopping and prompting the user when a GFE is encountered, this option allows the file-save procedure to continue. This is especially useful when no operator is around monitoring the file-save.

Do you want to sleep (y or <n>)?

The file-save does not need to be started immediately. An entry of "y" will cause the process to ask for the time to begin (in 24-hour military time):

Enter sleep time (default = 23:00):

Syntax

file-save {(options}

Options

Any options available to the "save" command may be appended to the "file-save" command at the TCL prompt.

Example

```
:to dm
:set-sct
Tape attached block size: 16384
[1082] Tape device is attached to quarter inch Streaming Cartridge
Tape
:file-save
Is this an Incremental Save       (y or <n>)? n
Console listing to printer        (y or <n>)? n
Send statistics report to printer (y or <n>)? n
Verify save with 't-verify (e)'   (y or <n>)? n
Do you want to sleep             (y or <n>)? n
File Save Beginning at 08:00:00  16 Jan 1997
Block size: 16384
Block size: 16384
[96]  bot
Block size: 16384
1    2 mds
1    3 mds > qa
1    4 mds > qa > lst
1    5 mds > qa > bp.qa
1    6 mds > qa > bp.qa > admin
1    7 mds > qa > bp.qa > bp.qa
1    8 mds > qa > model
In the standard form of this command, the account/dict/file names are
displayed as they are saved:
Tape#  file#  mds  >  account  >  dictionary  >  data
In all multi-reel operations, if the save detects end of media before
completing the save, it prompts with the message:
Load volume #2 and type 'C'
label  08:00:00  16 Jan 1997  DATA "account.name"         # -
When the next "reel" is inserted or mounted, "c" continues the
process, or "q" will stop it and return control to TCL.
At the end of the save process, the following message is displayed:
clearing dirty bits ...
2082
file-of-files being dumped to tape
[802] nnn items dumped
After the (full) file-save completes, the process scans through the
files and resets the "dirty bits", which indicate if the file was to
be saved during an incremental-save or not.  When finished clearing
the dirty bits, the file-of-files file is t-dumped to tape, if the
"I", account save, option was not specified.  If a file statistics
report was requested, it prints at this time, and a spooler queue
entry number is displayed.  See the entry for list-file-stats for
more information.
Entry #1
Block size: 16384
[96] bot
After the file stats are finished, a "t-rew" is issued to rewind the
tape. If "t-verify" was requested, it begins at this time, See the
entry on "t-verify" for more information. After the file-save (and
t-verify) is completed, it automatically logs the process off.
```

```
file save finished at 08:00:00  16 Jan 1997  Wednesday
< Connect time= 91 Mins.; CPU= 4240 Units; LPTR pages= 0     >
< logged off at 08:00:00     on 16 Jan 1997                  >
```

file.usage

see list-file-access

file.usage.clear

see list-file-access

find

searches a file for the existence of one or more strings of characters in any attribute and optionally creates and saves a list.

On releases 6.1.0 and above, the file.reference may be specified on the command line. Otherwise, the program requests it. Next, the search strings are requested by the program. Enter the string(s) without any enclosing delimiters. The system repeatedly asks for additional "search strings". If there are no additional strings to scan for, then entering a <return> at the "search string" prompt begins the search.

The program requests a list name. If provided, the list will be saved in the "pointer-file".

The item-id's of the items containing the search string(s) appear on the screen.

Note that the "find" command actually invokes the "search-file" command, and is simply a synonym.

Syntax

find {file.reference}

Example

```
find<return>
source file to search = bp<return>
string to search for  = customer.file<return>
string to search for  = invoice.file<return>
string to search for  = <return>
enter list item name  = find.list<return>
After responding to the last prompt in this example, the "find"
process begins searching each item in the "bp" file. If an item
contains either of the two strings provided, the item-id is added to
the list. After searching the entire file, the list "find.list" is
saved in the pointer-file.
find
source file to search =errmsg<return>
string to search for  =bad<return>
string to search for  =<return>
enter list item name  =errmsg.with.bad<return>
Item will be saved in the "DM,POINTER-FILE" file
1539 : E^H Bad descriptor type found.
B14  : E Line ^A^H:  Bad stack descriptor.
69   : E^H BAD GROUP
30   :  verb definition is bad
1514 : E^H Bad MDS 'D' type item.
B102 : E Line ^A^H:  Bad statement.
1521 : E^H Bad group found.
1522 : E^H Item with bad count field found.
```

find2

invokes the "search-system" command.

fl

forms a single list from two lists using intersection, union, or exclusion operators.

If file.reference2 is not specified, file.reference2 is assumed to be the same as file.reference1. If list2 is not specified, it is assumed to be the same as list1. If neither file2 nor list2 are specified, the effect is the same as a get-list. The created list overlays list1 in file1.

The "?" displays help text on the screen.

operators:

= Indicates an intersection. Forms list of matching items.

+ Indicates a union. Forms list of nonduplicate items.

- Indicates an exclusion. Forms list of items not in list2.

Syntax

fl file.reference1 list1 {operators} {file.reference2} {list2}

fl ?

Example

```
fl pointer-file football.fans + baseball.fans
This creates a list of all sports fans from both lists.
fl pointer-file football.fans = baseball.fans
This creates a list of those who enjoy both sports.
fl pointer-file football.fans - baseball.fans
This creates a list of those who like football only.
```

flush

"flushes" to disk all memory-resident buffers that are tagged as "write-required".

All write-required buffers are periodically flushed to disk in the normal sequence of events, but this command is provided to ensure data integrity at a given moment. In case of an emergency reboot or problems with the system, "flush" is a safety measure.

The flushing algorithm varies between implementations of D3. On D3 Unix implementations, the flush interval is set by the "set-flush" command. Roughly ten percent of the write-required frames are flushed at each activation.

font-parms

used to upgrade fonts and is provided for conversion of fonts created before May 25, 1990. Before this date, arguments were ordered differently.

Syntax

font-parms file.reference item.list* {(options}

Options

n Replaces all occurences.

s Suppresses display of messages to crt.

format

formats a floppy diskette for use only with the D3 System.

The density and drive specified by the last "set-floppy" or "set-device" verb are used.

If any bad sectors are detected by the format process, the diskette is not usable under D3.

Syntax

format {(options}}

Options

d Formats for use with DOS.

v Verifies diskette only.

Example

```
format (d
This formats the diskette most recently set with the "set-device" or
"set-floppy" command. For use by DOS.
format (v
This verifies the diskette.
```

format.pick

invoked by the "format" command.

frame-fault

a diagnostic tool to randomly read and rewrite frames in the data file space.

"number.frames" indicates the number of frames to test. If omitted, the process runs continuously.

A specific range of frame-ids may be specified.

Syntax

frame-fault {number.frames} | {start.fid-end.fid)}

Example

```
frame-fault 100000-100100
This tests frame faulting on fids 100,000 through 100,100.
```

free

displays information on the used and unused frames on a D3 system.

In addition to the information on the used and unused frames, it displays the total number of frames, the number of frames used by the operating system, the number in use for data, and the number of frames currently available or free.

Example

```
free
Disk Usage for D3's D3 GM-AT ver 1.5    16 Jan 1997
0....1....2....3....4....5....6....7....8....9....100%
*****************************
                                MB          Bytes     Frames
Maximum disk space     100%      19     19,352,000     19,352
System usage            11%       2      2,077,000      2,077
Data usage              29%       5      5,575,000      5,575
Available disk space    60%      12     11,700,000     11,700
```

fuser

displays ownership information about a device.

"fuser" is useful to determine which tape device is currently used, especially in an environment where several virtual machines are active, or where a Unix application shares devices with a D3 virtual machine.

"device.number" is the tape/device number, obtained from the "list-device" command.

"device.name" is the device name. This form can be used with any device name, even if it is not part of the tape devices normally used by the D3 virtual machine. The usual Unix wild cards are allowed (eg. 'fuser /dev/*fd0*', which examines all devices connected to the floppy drive 0).

"*" selects all Tape devices. This is the default option.

Syntax

fuser {device.number} {(option}

fuser {device.name} {(option}

fuser {*} {(option}

Options

p Directs output to system printer, via the Spooler.

Example

```
fuser *
DEVICE       PID  TTY   PIB USER ACCOUNT MODE/COMMAND
/dev/rfd0h:  1265 tty2  001 dm   dm      tp.chk:000
/dev/rfd0l:
/dev/rfd1.15:
```

```
/dev/rfd1.9:
/dev/rmt0.1:
/dev/rmt1.1:
/dev/rmt1.5:
/dev/px.gate:
```

gen.step.addr.data

used by the "MLOAD" program.

generic-restore

restores a specially formatted account-save created by certain utility programs.

The generic-restore program functions identically to the "account-restore" verb, except that it expects the tape to be formatted as a series of T-dumps written by a utility run on a non-D3 machine.

Current utilities for dumping from non-D3 boxes include: UVDUMP for dumping from uniVerse systems, and R83DUMP for dumping accounts from R83-based licencee systems.

Syntax

generic-restore {account.name}

get-fof

returns the item-id in the file-of-files associated with the specified file.

Syntax

get-fof file.reference

Options

l Generate a select list containing the file-of-files id.

s Suppress the output of the item-id.

Example

```
This command is useful for getting statistical data contained about a
file in the file-of-files.  For example:
get-fof bp (ls
3749
[404] 1 items selected out of 1 items.
list fof fname sug.mod modulo
Page 1    fof           09:42:39 09 May 1994
fof..... fname..... sug.mod. modulo..
3749      bp bp            23        17
[405] 1 items listed out of 1 items.
 This example shows that the optimal modulo for the file (in attribute
sug.mod) is significantly larger than the real modulo.  Based on these
results, this file would be a good candidate for resizing.
```

get-list

activates an item-list from a file. The default file is the "pointer-file".

Multiple list.names may be specified to create longer lists. If the optional "file.reference" is used the specified file is used to retrieve the item. "list.name" is the list that contains attributes to place in the active list. If more than one "list.name" is specified, the lists are concatenated into a single list.

If no "list.name" is specified, the default name, "%user.name", is used.

"attr.ref" indicates an attribute name or number. This creates a list from the specified attribute. Note that temporary attribute-defining items may also be used.

Syntax

gl {{file.reference} list.name {list.name...} } {(attr.ref} {(options}

get-list {{file.reference} list.name {list.name...} } {(attr.ref} {(options}

Options

s Creates a "secondary" list.

u Inhibits selection of duplicated items within multiple lists.

Example

```
gl xx yy
[404] 358 items selected out of 2 items.
>
get-list duplicates
[404] 67 items selected out of 1 items.
>
get-list old.invoices new.invoices (u
[404] 405 items selected out of 2 items.
This combines the two lists, "old.invoices" and "new.invoices" into one list.
The "u" option eliminates redundant references within the list.
```

get.pick

returns the name of the (D^3) program currently executing and is used by such utilities as "pick", "psr", "start.ss", "config.rs6000" and "free".

gl

Invokes "get-list" command.

go

see if

group

displays summary information about each group in a file.

The items in the group are summarized showing:

a) fid of group in decimal,

b) offset to beginning of item in the group in hexadecimal,

c) size of item in hexadecimal,

d) item-id.

Syntax

group file.reference {(options)}

Options

i Displays only those groups that have not had any items changed since the last file restore.

n No pause; suppresses pause at the end of the page on terminal display.

p Directs output to system printer, via the Spooler.

s Suppresses listing of details; displays summary group statistics only.

help

provides an interface to the on-line copy of Pick Systems Reference Manual, printed version.

To fully appreciate and exploit the help system, a working knowledge of the Update processor is required.

Quick command summary:

<ctrl>+xe Exit item.

<ctrl>+n Move cursor down one line.

<ctrl>+m Move to next paragraph.

<ctrl>+b Move cursor up one line.

<ctrl>+p Move to next page.

<ctrl>+ze Move to end of document.

<ctrl>+zy Move to previous page.

<ctrl>+t Move to top of item.

<ctrl>+g Move to end of attribute (paragraph).

<ctrl>+g<ctrl>+g Zoom to related subject.

<ctrl>+f Cruise forward on indexed attribute.

<ctrl>+f Move to next sentence in a text attribute.

<ctrl>+d Cruise backward on indexed attribute.

<ctrl>+d Move to previous sentence in a text attribute.

<ctrl>+e Erase to end of line.

<ctrl>+a Search for phrase or repeat last search.

Hotkeys:

<ctrl>+x4 Preview mode.

<ctrl>+x5 Print hard copy of subject. Use this on this item.

"help" invokes the TCL command, "help.display", and displays a read-only copy of the document/topic closest to the specified token. If a token is not specified, the "help" token is displayed.

Once inside the help system, the "token" (subject), "category", "type", "description", "syntax", "options", "compatibility", "see.also", "example", and "warnings" attributes are displayed.

"Cruising" through help topics:

You can "cruise" forward with <ctrl>+f or backward with <ctrl>+d on the "token", "category" and "type" attributes. All other attributes are text attributes. In an text attribute, <ctrl>+f moves to the beginning of the next sentence and <ctrl>+d moves to the previous sentence. Cruising on the "token" field is particularly useful. For instance, there are multiple "delete" tokens. If you were to type "help delete", you would arrive on the "first" reference to delete, which happens to be the FlashBASIC delete function. Issuing a <ctrl>+f while positioned on the token field takes you to the next "delete", which is the FlashBASIC delete statement. Another <ctrl>+f takes you to the TCL delete verb.

Changing topics:

While on the "token" field, you can change topics easily by typing a new topic over top of the existing word or words, then <ctrl>+e to erase the rest of the line, followed by <ctrl>+f to cruise to the topic. For example, suppose you are in the topic, "attribute-defining items", and you want to see the "list" verb. The screen would look like:

topic attribute-defining items

The cursor is positioned under the "a" in the word "attribute". This is where you type "list", so that the screen looks like:

topic listibute defining items

With the cursor under the second "i" in "listibute. A <ctrl>+e wipes out the rest of the "junk" on the line, so it looks like:

topic list

BEFORE you hit <return>, type <ctrl>+f. This will cruise forward to the first "list" subject found. If you hit <return> BEFORE hitting <ctrl>+f, you can still recover, but you have to re-position the cursor to the end of the token attribute (<ctrl>+b to move up, then <ctrl>+g).

"Zooming" to related files and subjects:

The "category", "type", and "see.also" attributes allow "zooming" to supporting files and subjects. "Zooming" is accomplished with two <ctrl>+g's. The first <ctrl>+g moves to the end of the attribute. The second <ctrl>+g "zooms" to the related file or subject. This is most useful on the "see.also" attribute, which is like a hypertext jump to a related subject in the same file.

To practice "zooming", see the topic "cursor movement" in the "see.also" field. To get to the topic, press <return> until the cursor is on the "cursor movement" topic, then "zoom" to the related topic. When you are done with the topic, <ctrl>+xe returns to the previous topic.

Exiting help:

<ctrl>+xe returns control to TCL, unless you have "zoomed" to one or more related help items through the "see.also" field, in which case, each <ctrl>+xe returns to the previous subject.

"help" invokes the "look-only" mode of the Update processor, which prevents the ability to file any items. This means that you can't hurt anything, in case you accidentally erase or type over part of a help item.

Printing a help item:

Once inside a help item, you can print the item using "hotkey" 5. This is accomplished by pressing <ctrl>+x5. The "hard copy" output is exactly the same as the "preview" mode output.

"Previewing" a help item:

To help distinguish the elements of a help item, we have built a special processing language that affects the syntax. The "preview" mode is activated by "hotkey" 4, (<ctrl>+x4).

The required elements in the syntax are shown in boldface characters. All "fill-in-the-blank" (parameters) of the syntax are underlined in "display" mode and italicized on hard copy. Every element listed in the syntax is either discussed in the "description" attribute, or is listed in the "see.also" attribute. See the topic "documentation conventions" for more explanation on the use of special characters in the syntax.

The "preview" mode uses AQL with a number of FlashBASIC subroutine calls. While in preview mode, pressing any output-producing key on the keyboard advances to the next page.

<ctrl>+x returns to the Update processor mode. You can not cruise or zoom while in "preview" mode.

Finding an item when you're not sure what you're looking for:

The help system is especially helpful if you know the topic you are searching for, such as the "locate" instruction in FlashBASIC. Our intent is to help you find your topic within a few hits. For instance, "locate" is one of the "see.also" fields listed under the topic of "statements and functions" in FlashBASIC. From this topic, you can "zoom" to "locate".

As a last resort, you can use the "search" or "find" commands to search through every help subject for a particular word or phrase. For instance, if you knew that there was a command that produced output in a four-column format, you could use the "search" command on the "ap.doc" file, looking for the string "four-column". (As an exercise, see if you can find at least one subject with the string, "four-column"). For more information on "search" and "find", "zoom" to them through the "see.also" field.

Searching for a word or phrase in a particular item:

In some of the documentation items, the "see.also" field can span many screens. To get to a specific point in the item, the UP search command can be used. For example, in the token "attribute-defining items", a "see.also" is provided to "hotkey9". To get to it in a hurry, use <ctrl>+a. When prompted for the search string, enter "hotkey9" (without the quotes), then press <return>. This searches for the NEXT occurrence of "hotkey9", which it finds in the text. Press <ctrl>+a again and it finds the next occurrence, which is the "see.also" to "hotkey9".

Syntax

help {token}

Example

```
help
This brings up this help item.
help locate
This looks for the token, "locate", and starts from that point in the
token's b-tree index.
help fred
This looks for the subject closest to "fred". Since "fred" does not
exist this should arrive in the topic, "free".
```

help.display

is called by the "help" command and invokes the Update processor to display the specified D3 document item-id.

If the item-id does not exist, the first item-id in the document is shown. Different views of the on-line documentation are provided by modifying the "help.display" command.

The "help" and "help.access" commands are used as front-ends to determine which item to look up. The difference in "help.display" is that the exact item-id of the item must be known.

Syntax

help.display {itemlist}

hot-backup

allows the setup and control of a hot backup configuration. Without any arguments a menu is displayed. This is the normal form of operation. See the section "hot backup, General", for a discussion of the important notions of the hot backup configuration.

Using the menus :

All operations are controlled through menus. If the terminal allows it, arrow keys can be used where indicated:

ENTER Validate the highlighted choice.

number From 0 to 9. Select the corresponding choice. '0' selects the option 10.

CTRL-N Move cursor down. (down arrow)

CTRL-B Move cursor up. (up arrow)

CTRL-X Cancel. Applicable only when input is requested.

ESC Quit. Go back to previous menu, or back to TCL. This key can be used to terminate all menus.

Q Quit. Go back to previous menu.

X Exit. Go back to TCL from any menu.

When the cursor is moved to a new field, a short help is displayed in the message area.

Screen layout :

The screen is divided in two sections:

- The menu section, where menus are displayed.

- The message section, where results, messages or help are displayed.

Definitions :

server

A background process. Servers work by pairs: one on each system.

master

The 'master' system is the main system, where users are normally working. By extension, the server running on the master system is the 'master server'.

slave

The 'slave' system is the backup of a master system. By extension, the server running on the slave system is the 'slave server'.

Transaction log queue

All updates to the master system are not transmitted directly to the slave system. Instead, they are put into a 'transaction log queue' on the master system, which is emptied regularly by the master server, in a FIFO manner. In normal operation, this queue should always be very small. When a transaction is extracted from the queue, a transaction number is allocated for it. A transaction is physically removed from the queue only when the slave server has acknowledged it, so that, if anything goes wrong before the item, for example, is stored in the remote file, the transaction can be sent again to the slave.

Log slave updates

Normally, on a slave system, updates made by the slave server itself should not be logged in the transaction log queue. However, it is possible to set the system so that a slave system acts as a master system to another slave, to cascade updates. This is defined by the 'Log slave update' option.

Apply Updates

It is possible to instruct a slave server to NOT modify the database on the slave system. Updates received from the master system are NOT applied to the slave database. Instead they are stored in a temporary file. The purpose is to have a quiescent database on the slave system without having to stop the master system, to be able to do a full save of the data base, for example. Updates can then be re-applied to the slave database at a later time.

File and account auto-create

The slave database is supposed to be an exact image of the master database. However, optionally, the slave server will attempt to create any account or file missing on the slave system.

Helper

On the slave system, long operations, like clearing large files, are not done by the slave server itself, since it would not be able to process any incoming data. The slave server creates phantom processes (helpers) to help it perform these lengthy operations.

Large file creation

Files with a modulo larger than 100 are created in a special way. A smaller file (101) is created by the slave server itself, and a helper process is created to resize the file to its actual size. This has the advantage that the file is created almost instantly, thus becoming available on the slave system, without stopping the slave server. The slave server checks periodically on the helper to make sure it completes.

Temporary hold files

The slave server sometimes creates temporary files on the slave system to hold data it cannot store in their final destination immediately. For example, when a clear file command is received, a helper does the actual clearing, and a temporary hold file is used so that updates to the file being cleared received AFTER the clear command are not also delted.

D Pointer Controls :

In order to be able to write data in files, the slave server does some controls, and changes, on the file-defining D pointers on the slave system:

- Update protections are removed. This is to ensure that the slave server will be able to write into the file.

- CALLX correlatives are changed to use explicit path names to the subroutines. The account and file name where the subroutine resides is found in the cataloged object in the MD of the account where the file is located. If the subroutine is not found, the CALLX correlative is removed.

- Bridges to other files are controlled to use an explicit path to the target file. If the file cannot be found, the bridge is removed.

However, the amount of controls is necessary limited. A-correlatives are not controlled, and, obviously, references to files in BASIC subroutines are not controlled either. Therefore, if precautions are not taken, it is possible that the slave server will abort trying to update such a file. Note that NO data loss would occur, though, because the offending transaction will not be acknowledged and will be retransmitted. Starting the server with the traces ON, will identify which item and file causes the problem.

Installation :

Log on to the 'dm' account and type:

hot-backup

The first time this command is invoked, a server must be defined. The following menu is displayed:

Setup Server

1 New Server

Select and press <RETURN>

Type <RETURN> and fill in the appropriate information as explained in the section 'Setup Server' below.

Setup Server :

The 'Setup Server' menu is brought up automatically the first time 'hot-backup' is installed, or by selecting an option in the main menu.

Server name :

Any alphabetic string of up to 8 characters. This name will reference the server in all the other menus. If there is only one server defined on the D^3 virtual machine, this server will always be implied in all commands.

Server type :

Master or Slave server. Depending on the server type, some fields of the remaining of the menu will become unapplicable.

Host name :

Master only. Name of the Unix host where the associated slave server is located.

TCP port number :

For a Slave Server, TCP port number, from 1024 to 32767, on which it is listening. For a master server, TCP port number of the associated slave server.

Protocol : inet

Cannot be modified.

Log slave update :

Slave only. If 'ON', all updates received by the slave are logged in the transaction log queue, to be sent to another system. If 'OFF', the updates are not enqueued. The default is 'OFF'.

Check period :

Master only. Period, in seconds, with which the master server will perform a variety of system checks. 0 disable all checks. A period of 300 seconds (5 minutes) should be suitable for most systems.

TxLog timer :

Master only. Period, in seconds, with which the master server will empty the transaction log queue. If the queue is not empty, then the server empties it continuously. If it becomes empty at some point, the master server goes to sleep for this period. A small value (2 or 3 seconds) is suitable.

Notify list :

List of D3 users or D3 port numbers to notify in case of error, separated by a comma. The port numbers are expressed in a syntax similar to the TCL command 'msg'. For example, to notify the users 'dm' and 'bob', and the line 0, whether it is logged on or not:

dm,bob,!0

Apply updates :

Slave only. 'ON' or 'OFF'. Instruct the slave server to apply the updates it receives from the associated master server immediately (ON) or delay them (OFF) until it receives an explicit command. Normally, this option should be set to 'ON'. Note that, when later disabled, the server definition will be modified automatically. When the slave server starts, it will report the status of this option. Applying updates should not be disabled too long, since the slave data base is not an exact image of the master data base while updates are not applied.

Auto create :

Slave only. 'ON' or 'OFF'. Create automatically any missing account and file on the slave system. It is a good precaution to leave this option ON.

Comments :

Any text.

Confirm (y/n/q):

'y' to confirm the server creation. 'n' to go back to any of the previous fields. 'q' to quit and abandon.

Current Server Selection :

Most menu commands refer to a 'current' server, identified by its name. If there is only one server on the D3 virtual machine, then this server is always implied. If there is more than one server, the following menu is displayed when first entering 'hot-backup':

Operation title

1 server.name

2 server.name

...

Select and press <RETURN>

Type the selected number and <RETURN> to select a server. The 'current' server name is displayed under the hot backup header. In the following, 'selected server' will designate either the only server on the machine, or the current server selected by this menu. To select another 'current' server, hit 'ESC' or 'Q' back to the server selection menu.

Main Menu

1 Status

Display the last messages displayed by the selected server. The title of the message section on screen indicates the server name and additional information. The messages are displayed in reverse chronological order (most recent first). The server does not need to be running for this command.

2 Query Server

Query the selected server. The server must be running. See the section 'Query Server' below for the detail of the information returned.

3 Start Server

Start the selected server. This process displays various information about the starting of the server.

4 Stop Server

Stop the selected server. Confirmation is requested.

5 Show Server

Show the setup of the selected server. Type 'y' return to the main menu. The description of the displayed information is identical to the 'Setup Server' menu above.

6 Show statistics

Transaction statistics. If run on the slave server, the result is displayed immediately, if the server is running. If run on the master server, a request is sent to the slave server and displayed later. Both servers must be running. The response time depends on how busy the slave server is. This command is a quick way of determining whether the communication is established. The statistics are accumulated by the slave server and cleared every time it restarts.

7 Display Queue

Display the status of the transaction log queue. The following information is shown:

n frames in queue

DL or ALL files logged

File updates enqueued for all processes

or

File updates no longer enqueued

Current transaction: nnn

Last ACK'ed transaction: nnn

Current transaction: nnn

Upd itm 'item.id' in 'file.reference'

The number of frames is a rough indication of the amount of data not yet transmitted. The current transaction number is the transaction number being transmitted, if non zero. The last ACK'ed transaction number is the last transaction which was received and successfully processed by the associated slave server. If the last ACK'ed number is not reported, this means the query was done at a moment when all transactions have been acknowledged. The last line is an indication of the first transaction not yet acknowledged by the associated server. This can be an update item, a delete item, etc... In case of problem, this is the first transaction which would be retransmitted.

8 Special operations

Perform less frequently used operations. See the section "Special Operation Menu" below.

9 Transaction log menu

Control the transaction log queue. See the section "Transaction Log Menu" below.

Special Operations

1 Status

Display the last messages displayed by the selected server. Same as on the main menu.

2 Turn traces ON

Turn traces ON on the selected server. Traces are slowing down the server considerably and should not be left ON in normal operations. This is a debug tool only.

3 Turn traces OFF

Disable the traces on the selected server.

4 Setup Server

Setup a new server or modify the selected server. This the section 'Setup Server' above.

5 Delete server

Delete the selected server definition. Confirmation is requested.

6 List permanent log

List the permanent log of all messages logged by all servers. The log is shown in the message area on screen. Use the CTRL-N to see the next logs, and CTRL-B to go back in the log. Type ':' (colon) to enter a special command line which allows setting searching the first log entry after a given date or time, and search for a string. Type 'q' or ESC to return to the special operation menu.

7 Clear permanent log

Clear the permanent log file. A sub menu offers the choice of clearing the whole file, entries older than one week, or older than one month. Confirmation is requested.

8 Stop applying updates

Slave only. Instruct the selected server to stop applying updates coming from its associated master server. The updates are stored in a temporary hold file 'hb.log,apply'. The number of pending updates is shown in response to a 'Query Server' operation from the main menu. Updates can be applied by selecting the next option.

9 (Re)start applying updates

Slave only. Apply the pending updates to the database, if any, and restart the normal state of applying new updates as they arrive.

10 Close opened files

Slave only. The slave server maintains a list of all the files it has opened since it started. If it becomes necessary to delete a file on the slave system, or after recovering from a missing file-of-files item on the master system (see the discussion below about the error message 'Cannot read FOF item'), the slave server must be instructed to close all the files it has opened. This option allows this. If the files are accessed again, the slave server will re-open them.

Transaction Log Menu

The transaction log menu should be used on a system which is setup as a master system. It controls what and how updates are transmitted to the associated slave system.

1 Display queue

Display the status of the transaction log queue. Same as option 7 on the main menu.

2 Log DL files only

Log only the files which have a DL attribute in the D pointer. This option affects the entire system. This mode of operation allows fine control over what is being sent to the other system.

3 Log ALL files

Log updates to all files on the system. Note the slave server will NOT apply any modification to selected files in the 'dm' account on the slave server: abs, accounts, devs, errors, file-of-files, hb.log (hot backup control files), jobs, pibs resizing. This option should be used with caution, since the slave server may create accounts and files to receive data.

4 Stop enqueuing

Stop enqueuing updates. Confirmation is requested. This option should be used with extreme care, since, once selected, all updates to the master data base will NEVER be transmitted to the slave system. The only legitimate use of this option is to stop enqueuing updates following a major problem on the main system which required switching users to the slave system. This event is also logged in the 'errors' file.

5 Start enqueueing

(re)Start enqueuing updates. This event is also logged in the 'errors' file.

6 Clear log queue

Clear the transaction log queue. Confirmation is requested. All queued updates are lost. This event is also logged in the 'errors' file.

Query Server Results :

On a Slave server, the following information is displayed:

Server <servername>. PIB <pib>. PID <pid>. Server type [Slave|Master]. Traces [ON|OFF].

Total [recvd|sent] 289K. {Cur trans 1041.} ACK'ed trans 1039. {updates {NOT} applied.}

{n file(s) failed to open successfully}

{n pending update(s)}

Status <status>

Where:

servername

Name of the server.

pib

D3 port number of the server.

pid

Unix process id of the server.

Cur trans

Slave only. Current transaction number received by the slave server. This number should increase as data is exchanged between the servers.

ACK'ed trans

Current transaction number successfully acknowledge by the slave server. Unless the queue is empty, the ACK'ed transaction number should be slightly less than the transaction number received by the slave.

{updates {NOT} applied}

Slave only. If present, indicates whether updates received from the master system are currently applied to the slave database.

Total [sent|recvd]

Total size, in kilobytes, received for the current day. Note the master reports a slightly higher number than than the slave, due to additional protocol information.

{n file(s) failed to open successfully}

Slave only. If present, indicates that the slave server received requests to open files and was not able to do so. At any given time, there may be a few pending open files, so repeat the command to make sure there are some problems. Examine the permanent log file to get more information.

{n pending updates}

Slave only. If present, indicates that the updates to the slave data base are currently disabled, and that there are 'n' pending updates.

Status

Server status:

Reading network

This is the normal state of the slave server.

Wait response to call

Master only. The master server is attempting to call its associated slave.

Wait incoming call

Slave only. The slave server is waiting for its associated master server to establish a communication.

Stopped

The server has stopped. The previous message in the status or permanent log file would indicate the reason of the termination.

Connected

Intermediate state right after a connection is established.

Writing to network

Sending data to the associated server. If this state persists and no data transfer occurs, this indicates the receiver is not reading data. There is a 30 second time out after which the connection will be shutdown and restarted.

Waiting for ACK

Master only. Wait for an acknowledgment from the associated slave server.

Idle

Master only. The queue is empty and there is no traffic.

Reading queue

Master only. Intermediate state while the master server is extracting messages from the transaction log queue.

Non Menu Operation :

It is possible to perform some operations from TCL by specifying a 'command' on the TCL line. If there is more than one server, the server must be specified by 'server=server.name' argument. This for is useful to perform some automatic commands in macros.

'command':

start

Start the specified server.

stop

stop the specified server.

status

Display the status information (same as option 1 on the main menu) of the specified server.

queue

Display the transaction log queue information (same as option 7 in the main menu).

debug

Force the specified server to enter the BASIC debugger. This option is for testing only. The server can then be debugged using tandem.

Main Error Messages :

This section lists the main error or warning messages that can either be displayed in the message area or logged in the permanent log file.

ERROR: This command applies only to Slave servers

This command can only be used when selecting a slave servers. Some function cannot be performed by master servers.

Not a phantom. Running on PIB xxx. PID yyy

This message can appear when interrogating a server which is either started in foreground (for debugging purpose) or, most likely, which has aborted, or was killed, and left an inconsistent status in the jobs file.

Terminated abormally

The server phantom process has terminated, but without finishing properly. The process was probably logged off, or encountered an fatal error.

Notice from server 'servername'

The message in the message area is coming from the specified server.

Waiting for server 'servername' to start

Wait for the phantom process to start.

Waiting for server 'servername' to initialize

The phantom process has started and now wait for the server to initialize itself.

FAILED: Cannot start phantom process

The phantom process failed to start. Check the scheduler, make sure there are not too many queued jobs. Use 'list-jobs' to determine the cause of the abort.

FAILED: Server 'servername' encountered an error and stopped

The phantom process started normally, but the server could not complete its initialization. Use the 'status' menu option to see the cause of the failure. The most common cause is a 'BIND' error, indicating that the TCP port is already in use. Often, when stopping a server, the TCP connection is not purged immediately and the TCP port number remains in use. This usually cleans up by itself after the various TCP protocol time outs have expired. Stop the master server. Retry a few minutes later. If the BIND error persists, change the port number using the 'setup' option, on BOTH the master and the slave sides.

WARNING: Server 'servername' is already running on PIB xxx

A start server failed because the server appears to be already running. This has no action on a running server. If the server is NOT running, and this message still appears, this might indicate a corruption of the hot backup control file. Make sure the server is not running (using where) and do:

delete hb.log servername

WARNING: Transaction are NOT enqueued

This warning is issued by the master server to indicate that updates to the data base are not being enqueued, and therefore not being transferred to the slave system. The databases are now different.

WARNING: Transaction queue has grown to xxx frames

Periodically, the master server checks if the queue has grown to more than 500 frames above the previous value. This message is an indication that the slave server is not able to keep up. Make sure the input server is not stopped, or has not aborted. This message MAY be normal in exceptional situations where massive updates are being done.

malloc error. err=xxx

The system could allocate memory. This is an indication of a serious Unix problem. Check the swap space.

socket creation failed

The socket library is not linked properly to the Monitor being run by the phantom. Refer to the installation guide to link the proper socket libraries fro implementation for which this library is an optional package.

Cannot call host 'hostname'. Errno=xxx

The master server cannot establish a communication. 'xxx' is an indication of the error. The error code is very implementation dependent. Make sure the host is reachable by using the Unix command 'ping hostname'. If so, the most likely reason is that the slave server is not started or has encountered an error. Use the status menu option on the slave.

Cannot read FOF item 'xxx'

Due to a corruption in the file-of-files file on the master system, the system is unable to identify the file being updated. 'xxx' is the file number (as known on the master system). Data is normally transferred to the slave system but stored in a temporary hold file named "hb.missing,mastername*xxx", where 'mastername' is the name of the master server, and 'xxx' the file number. To correct this situation, the file-of-files file on the master system must be rebuilt by doing a full file save (possibly a dummy save), with the (S) option. The missing fof item can also be rebuilt by hand if the file name can be identified from the content of the file

being stored in the temporary hold file on the backup system. The data can then be copied to the real file. Instruct the slave server to close its opened files, using the special operation menu, to force it to open the new file.

Lost+Found File :

When a slave server is unable to process a transaction (for example, the account does not exist and the automatic creation of missing accounts is disabled), the transaction is stored into the trash file 'hb.log,lost+found'. The item-ids of this file are unique timedate stamps. This file can be used to recover the salvaged data. The format of an item is:

1 Item type:

1: Unknown transaction code.

2: Slave server was reset.

3: Internal error.

4: Time out on transaction. These errors can occur when stopping the master server.

5: Bad network message.

6: Cannot perform the operation. The error message usually indicates the cause of the problem.

7: D pointer can not be updated. There is a D pointer like element in the master dictionary or file dictionary preventing the creation of the file. This item was deleted to allow the file creation and its content written starting at the attribute 10.

8: The MD of the account in which the slave server is running (normally 'dm') contains an item which collides with the name of an account. Though this is normally authorized, this is not a good practice. The server removes this item and dump the content of this item.

9: The slave server removed a correlative from a D pointer because it could not resolve a reference to a CALLX subroutine or to a bridge. The lost+found item contains the removed correlative.

10: Transaction could not be processed when applying delayed updates.

11: Incorrect transaction length.

2 Transaction opcode, if applicable, or '?'.

3 Server name.

4 Log internal time.

5 Log internal date..

6 Transaction number.

7 Last ACK'ed transaction number.

8 Message.

9 Time date in external format.

10 Body of the transaction which could not be processed or any data the server could not handle. More than one attribute can be used. This data may be incomplete.

Syntax

hot-backup {{command} {server=name}}

Options

Q Quiet. Valid only for the non menu operation. Supresses all messages.

V Verbose. Turn traces on when starting the server from the menu.

Example

```
Examples :
hot-backup
  Enter the main menu.
hot-backup start
  Start the (only) server on the virtual machine.
hot-backup start server=todev
hot-backup start server=fromprod
hot-backup status server=todev
hot-backup status server=fromprod
  On a system which is both slave and master, start a master server
'todev' and a slave server 'fromprod'. Obtain status from both of
these servers.
```

if

conditional branch statement in a paragraph

The if statement compares the two string operators. If they are both numeric, then they are compared numerically. If either or both cannot be converted to a number, then they are compared alphabetically.

Between the two strings is a single-character conditional operator which may be one of the following:

<,l Less Than

=,e Equal To

>,g Greater than

If the condition evaluates to true, then the paragraph processor starts searching from the beginning of the paragraph for a label which matches the one after the "go" statement, but which must be followed by a colon and a space. When this label is found, execution of the paragraph continues on the line containing the label.

One of the string parameters in an if statement is usually an input substitution.

Syntax

if {"|"}string1{"|"} [<|l|=|e|>|g] {"|"}string2{"|"} then go label

Example

```
if <<Input a number between 1 and 10:>> > 5 then go
greater
display Less than or equal to 5
if 1 = 1 then go done
greater: display Greater than 5
done: * There must be a space after the colon
 This example asks the user for a number, and then prints a result
based on the value of this number.  Note how the "if" statement is
used on the third line to implement an unconditional branch.
```

import (Unix)

imports Unix files to D3 items.

<New line> characters are converted into attribute marks. The source Unix files are either specified explicitly or are specified by the directory on which they can be found.

"file.reference"

Destination D3 file name.

"item.list"

List of the destination items. If not specified, a list must be active.

"dir"

Directory where the source files are to be found. The filenames are the same as the item-id(s) from the source list. The "l" option is used to convert Unix filenames to lower case.

"unix.file"

Explicit list of the source Unix files. Complete pathnames must be used if the files are not on the current directory. The current directory may be changed by the "cd" command. If no source list is provided, the destination list is used as a source list also. Wild cards are legal in unix.file, but in this case, all the files are concatenated before the copy.

Syntax

import file.reference item.list {(options}

from :[dir |{unix.file {unix.file} ... }]

Options

en Expands Unix tabulations into "n" spaces. If "n" is not specified, the default is 4. Tabulations expansion is done using the pr(1) Unix command.

i Suppresses display of item-id(s).

l Converts unix.file name in lower case before doing the copy.

O Overwrites D3 item(s), if duplicate item-id(s) encountered.

indexer

builds indexes on part or all of the files of the system.

Rebuilds indexes for:

1) The entire file system, if no list is active or "account.name" is specified.

2) A specific account, if the "account.name" is specified with the command.

3) Each account in an "active" list, if a list is "active" when the command is invoked.

This command is useful when an account-restore does not finish properly, or is voluntarily preempted.

Syntax

indexer {account.name} {(options)}

Options

a Uses the active list as the list of files to rebuild indexes.

n Outputs report without pausing at the bottom of each page.

p Directs output to system printer, via the Spooler.

Example

```
indexer
This rebuilds all indexes in all accounts.
indexer prod
This rebuilds all indexes in the "prod" account.
select mds 'dm' 'prod' 'accounting'
[404] 3 items selected out of 37 items.
indexer
```

```
This rebuilds the indexes for the "dm", "prod" and "accounting"
accounts.
```

init-ovf

initiates the D3 overflow initialization.

See the topic, "scrubber", for an explanation of what this process does.

"init-ovf" is the actual verb invoked by the "initovf" command.

init-pibs

assigns all ports a term type by updating attribute 2 in the pibs file.

If no device name is specified, the default "wy-50" term type is assigned to all ports except port zero which is always assigned to "mm-mon"

If attribute 2 in the pibs file item already has term information then it is not overlayed unless the i option is specified.

Syntax

init-pibs {device.name} {(i}

Options

i initialize or overlay existing term type in attribute 2 of pibs item

initovf

checks the overflow table to determine if it has previously been "scrubbed" and invokes the "init-ovf" command if necessary.

inputwait

waits for any character input and falls through or if it times out it executes the command(s).

"delay" The delay time in seconds.

"text" Optional text to be displayed before the input. The text is in the format accepted by the "display" command.

"command" One or more commands, separated by semicolons, executed if a character is entered within the specified time.

Syntax

inputwait delay ; {text} ; command { ; command ...}

Example

```
inputwait 10;@(-13)Type any key to stop logoff ... @(-14)+; off
Displays a highlighted message with no carriage return, waits for 10
seconds for any key to be hit. If no keys are hit within that time,
the 'off' command is executed.
```

install.help

installs or reinstalls pointers to D3REF (Pick Systems Reference Manual On-line help) into an account.

After verifying "install.help" is being run from the D3REF account, "install.help" prompts for the name of the account where help should be installed.

If there are no naming conflicts, between existing md items in the target account, and the md items "install.help" creates, "install.help" displays the names of the items it is creating.

The first time a conflict occurs, "install.help" asks if the installation is an upgrade or reinstallation. A "Y" for yes, tells "install.help" to continue the installation and ignore all further conflicts. A "N" for no, tells "install.help" to ask for permission to overwrite as each conflict it is discovered.

iselect

selects all of the item-id's in the group that the specified item-id hashes to, using the "item" verb.

This verb is useful when dealing with gfe's. It can be used to determine which items are in a particular group, so that they may be copied out of the group.

Syntax

iselect file.reference item-id

Example

```
iselect md who
[404] 26 items selected out of 1 items.
```

isselect

selects and sorts all of the item-id's in the group that the specified item-id hashes to, using the "item" verb.

Syntax

isselect file.reference item-id

Example

```
isselect md who
[404] 26 items selected out of 1 items.
```

item

outputs the base fid of the group to which the specified item-id "hashes", and a list of all item-id's that are currently "hashed" to the same group.

Also displayed is each items' (hexadecimal) byte count field, a count of the total number of items in the group, the total number of bytes, the total number of "full" frames, and the number of bytes used in the "last" frame of the group.

If the given item-id is not found, the group to which it would hash is displayed.

Case sensitivity is an issue in the hashing algorithm used by the D^3 system. If a file has a "d-pointer" type of "ds", item-ids are case-sensitive. This means two things: 1, that the item-ids "dog" and "DOG" are treated as two separate items and 2) they would most likely hash to different groups (unless, of course, the file has a modulo of 1). If case sensitivity is "off", "dog" and "DOG" are the same thing, and there can only be one dog. See the "case", "case-on", and "case-off" verbs, and the topic of "file-defining items".

Items can be displayed in a group. In the columns between the address and the byte size, three possible letters may appear. They have the following meanings:

B Binary item. This normally occurs with FlashBASIC object code found in the dictionary level of source files.

F File pointer. This occurs only with "d-pointers".

P Means that this is a "pointer" (indirect) item. This occurs when a file is designated as having a "dp" in its "d-pointer", or when the item size passes approximately eighty percent of the frame size on the system. (See the "what" command for determining frame size).

Syntax

item file.reference itemlist* {(options}

Options

n Activates nopage function on output to the terminal.

p Directs output to system printer, via the Spooler.

s Suppresses output of item-ids'.

Example

```
item testfile stuff
stuff
item not found
320710.0000     01BC  pc.v
1 items 444 bytes 0/444 frames in group 320710
This example illustrates the effect of not finding the item, yet
showing the group to which the item would hash if it were there.
```

jobs.status

called by the "list-jobs" command to determine the status of a phantom job.

kill (Unix)

sends a Unix signal to a D3 or Unix process.

Signals can be specified by their usual names, rather than their number, and D3 processes by their port numbers instead of Unix process-ids ("pid's").

signalno is the optional signal number in decimal. If not specified, "SIGTERM" is sent.

signame is the optional signal name. Among the valid signal lists are: {SIG}TERM, {SIG}HUP, and {SIG}USR2. If not specified, SIGTERM is sent.

p pib indicates the D3 pib (port.number), prefixed by the letter "p".

'kill' checks that none of the specified processes have a process-id of 0 (zero), which would mean 'kill all processes in the group', usually resulting in an abrupt shutdown of the D3 virtual machine.

Syntax

kill {-[signo|signame] } [pid|ppib] { [pid|ppib] ... }

Example

```
kill p17
Kills the (Unix) process associated with port 17.
kill -sigalrm p33 7143
Sends a sinal, "sigalrm", to the Unix process associated with port 33
and to the Unix process whose process-id is 7143.
kill -1 p0
Sends a "sighup" to port 0.
```

kill-resizing

stops all currently active resizing processes. This command finds all processes on the system which are currently resizing a file. If these processes are phantoms, then they will be terminated.

Resizing processes on regular lines will be terminated, but the line will not be logged off (except in extreme cases where the resizing process cannot be stopped).

last.command

invoked by the "cmdu" command for retrieving the "last" executed TCL command for each active user.

last.tcl.entry

invoked by the "list-errors" command for retrieving the "last" TCL entry for items in the "errors" file.

ld

presents the best guess of what one wants to see in a columnar listing of any dictionary.

The information in the report is sorted by the attribute count of each item in the dictionary.

Syntax

ld {sellist} {(p}

ldf

lists the summary description (attribute 17) of each attribute-defining item in the specified file.

The information in the report is sorted by the attribute count of each item in the dictionary.

Syntax

ldf file.reference {(options}

legend

displays the current status of "legend" output, or toggles the display of the automatic page legend found in the item called "legend" in the "dm,messages," file.

If present, the "legend" item is output on the bottom of each page of OP (Output processor) reports directed to the spooler, unless "legend suppress" ("leg-supp") is in effect.

The "legend" is a static OP document stored as an item in "dm,messages,". This is in contrast to headings and footings, which are dynamic, changing with each page output.

Each user can have their own legend item, using "legend*user-id" as the naming convention for the "dm,messages," item. For example, "legend*bob".

If the legend status is "on", the system searches first for a user legend. If none exists, the default item, "legend", is used.

Syntax

legend {(options}

Options

f Toggles legend off. Suppresses legend on Spooler output. This eliminates the need for the "leg-supp" connective or "k" option.

n Toggles legend on. Outputs legend on Spooler listings.

s Suppresses output of status message.

Example

```
legend
[1303] Legend output is suppressed.
```

```
legend (n
[1302] Legend output is active.
```

legend-off

turns off the automatic page "legend" function. See the "legend" command.

legend-on

turns on the automatic page "legend" function. See the "legend" command.

lerrs

performs a series of reports from the "dm,errors," file.

lf

Invokes "list-files" command.

lfd

produces a sorted listing of all file-defining items in the specified md, along with the description attribute from each file pointer.

Syntax

lfd file.reference {(options}

lfs

contains the actual AQL sentence executed by the "list-file-stats" program. Any options from "list-file-stats" are appended to the macro and included in the sentence.

This macro is invoked by the "list-file-stats" and the "file-save" commands.

li

Invokes "list-item" command.

link-pibdev

creates a logical "link" between the given (or current) "process" and the specified "port.number".

The process and the specified port must not be in "tandem, "converse", or "mirror".

Syntax

link-pibdev {process,}port.number

Example

```
link-pibdev 3
This links the current process to port 3.
link-pibdev 2,4
This links process 2 to port 4.
```

list-abs

produces a report of each D^3 mode in the "dm,abs," file, showing any modes it calls and those modes which are called.

Syntax

list-abs {(options}

Options

* see "options: AQL"

list-acc

Invokes "listacc" command.

list-commands

produces a report showing every executable command in the specified master dictionary; this includes: procs, macros, menus, cataloged FlashBASIC programs ("basic verbs") and D3 verbs.

Syntax

list-commands md.name {(options}

Options

* see "options: TCL"

Example

```
list-commands md
list-commands dm,, (p
This will direct the report to the spooler.
```

list-conn

Invokes "listconn" command.

list-device

displays the currently defined devices.

The devices listed may be used as 'tape' devices through the "set-half", "set-sct", "set-floppy" or "set-device" commands.

Devices are numbered from 0 to 15 as they appear in the configuration file. The current device is noted with an "*". The active device is noted with a "+".

"device.number" is the device number to list, from 0 to 9. If not specified, all currently defined devices are listed.

When using the (l) option to display long form, the following additional elements are displayed:

Blk=xxx Default block size, in bytes.

Opt='XXX' Device options:

s Short label. Valid only on 8mm tape. This option indicates that the device will write D3 labels in 80-byte blocks, and will attempt to read D3 labels in 80-byte blocks. This option may be necessary to exchange 8mm tapes with non-D3 systems. A tape written on a device which has the "s" option set cannot be read without this option. The default is to write the 80-byte label in a 512-byte block. This option can be changed by the "chg-device" command.

r Immediate rewind. On some Unix implementations, some devices support an immediate rewind, which means control is returned to the process immediately, without having to wait for the rewind to complete.

l Link field on tape for ABS restore. This option is an internal flag used at boot time when restoring an ABS from this device.

Syntax

list-device {device.number} {(options}

Options

h Suppresses header display. Only devices are displayed.

l Long form. Displays additional information in the OPTIONS field.

p Directs output to system printer, via the spooler.

Example

```
:list-device
 Tape        Status           16 Jan 1997  14:30:40
 #   Type             Density           Owner   Device Name
-------------------------------------------------------------------------
 0 | Floppy         | 3 1/2" 1.44M    | 191*+ | /dev/rpdsk/4
 1 | Quarter Inch   | High            |       | /dev/rmt/0n
 2 | 4mm DAT        |                 |       | /dev/rmt/1n
 3 | 4mm DAT        |                 |       | /dev/rmt/1un
 4 | 8mm Tape       |                 |       | /dev/rmt/2n
 5 | 8mm Tape       |                 | 141   | /dev/rmt/2un
 6 | Floppy         | Pseudo Floppy   |       | /usr/opt/pick/bin/abs
 7 | Floppy         | Pseudo Floppy   |       | /usr/opt/pick/bin/data
 8 | Floppy         | Pseudo Floppy   |       | /usr/opt/pick/bin/ref
 9 | Network        |                 |       | /home/tmp/pipein
10 | Floppy         | Pseudo Floppy   | 191*  | /home/tmp/floppy
11 | Floppy         | Pseudo Floppy   |       | /n/dev/home/tmp/floppy
-------------------------------------------------------------------------

The "Owner" column displays the pib number that has use of the
device, and a "*" indicates that this pib is the current process. A
"+" indicates that this device is the "active" device (as a pib can
attach more than one device).
```

list-dict

Invokes "listdict" command.

list-errors

produces a sorted report of the error messages logged into the "dm,errors," file, beginning with the most recently logged message.

Syntax

list-errors {(options}

Options

* see "options: AQL".

Example

```
list-errors jobs
report of the error messages logged into the "jobs" file.
```

list-file-access

displays file access statistics for the desired file(s).

This command displays counters which relate the amount and type of file access occuring on a given file (specified by the file name or the file-of-files number). If no file reference is specified, then the program will take the file references from an active select list. If no select list is active, and no file references are specified, then the statistics for the entire file system are displayed.

File statistics are always collected by system routines and have no appreciable impact on file system performance.

The statistics are divided into two columns. The first column shows the total results since the statistics were cleared. The second column shows the average results per day since the statistics were cleared.

The categories are as follows:

Total Reads The total number of reads on the file. Note that this figure includes all reads, some of which are not included in the sub-categories listed below.

Readu The number of reads which initiated an item lock. The percentage of total reads is also displayed.

Readu (Blocked) The number of reads which initiated an item lock, but were blocked by another user. The percentage of total reads is also displayed.

Readu locked The number of reads which initiated an item lock with a "locked" clause. The percentage of total reads is also displayed.

Readu locked (Blocked) The number of reads which initiated an item lock with a "locked" clause but were blocked by another user. The percentage of total reads is also displayed.

Pointer Item Read The number of reads of pointer items. The percentage of total reads is also displayed.

Overflow Read The number of reads which traversed an overflow group. The percentage of total reads is also displayed. If this percentage is greater than 30%, the file probably should be resized.

Total Writes The number of writes to the file. Note that this figure includes all writes, some of are not included in the sub-categories listed below.

Writeu The number of writes which did not release a held item lock. The percentage of total writes is also displayed.

Delete A delete is essentially a write which does not leave any new data. This value is the total number of item deletes performed on a file. The percentage of total writes is also displayed.

Write (Blocked) The number of writes which were blocked by another user's item lock. The percentage of total writes is also displayed.

Overflow Write The number of reads which traversed an overflow group. The percentage of total writes is also displayed. If the percentage is greater than 30%, the file probably should be resized.

Total Selects The total number of sequential operations on a file. Note that a sequential access (like a "count" or "sort") increments this number by 1 rather than the number of items traversed. Sequential accesses include AQL statements, BASIC select's, and utilities like ISTAT, and CREATE-INDEX.

Total Clears The total number of "CLEAR-FILE" operations performed on a file.

Total Opens The total number of times a file has been opened. Note that the "list-file-access" opens the file, thus incrementing this counter, but decrements it immediately afterwards so that the command does not affect the statistics.

Syntax

list-file-access {[file.reference|fof.number]...} (options

Options

c Clear instead of display. This option traverses all files specified (or the whole file system if no file is specified), and clears the statistics. This command should be used (preferably on the whole file-system) before any data is collected.

p Directs output to the spooler.

Example

```
list-file-access bp
Page 1          *** LIST-FILE-ACCESS ***              18 May 1994
================================================================
File:bp           CURRENT TOTALS  AVERAGE PER DAY (from 05/10/94)
----------------------------------------------------------------
Total Reads            384              42
 Readu                   0 (  0% )       0 (  0% )
 Readu (Blocked)         0 (  0% )       0 (  0% )
 Readu locked           28 (  7% )       3 (  7% )
 Readu locked (Blocked)  0 (  0% )       0 (  0% )
 Pointer Item Read     142 ( 36% )      15 ( 35% )
 Overflow Read         141 ( 36% )      15 ( 35% )
Total Writes            15               1
 Writeu                  0 (  0% )       0 (  0% )
 Write (Blocked)         0 (  0% )       0 (  0% )
 Delete                  1 ( 13% )       0 (  0% )
 Pointer Item Write      4 ( 26% )       0 (  0% )
 Overflow Write          0 (  0% )       0 (  0% )
Total Selects            6               6
Total Clears             0               0
Total Opens            125             125
From the above data, one can ascertain that the file is fairly busy. Locking
does not appear to be a problem because none of the (Blocked) categories have
tallied significant numbers.  The overflow group accesses are a bit high and
may indicate the potential need for resizing.
The raw file statistics are archived into the file-of-files by the file-save
process.  Therefore, it is possible to select and display these statistics
using the file-of-files directly.  The following example displays the 5 files
on the system with the greatest overflow group access:
sselect fof by-dsnd stat.ovf
[404] 2347 items selected out of 2347 items.
select fof sampling 5
[4042] 5 items selected out of 5 sampling items.
list fof name
Page 1    fof
fof..... name............
1349      fred*bp*bp
2343      ba*bp*bp
1293      ba*accounts*accounts
1943      ba*memos*memos
1222      ba*log*log
[405] 5 items listed out of 5 items.
These files would be good candidates for further examination. To get the full
access statistics (including any changes occuring after the last file-save),
do the same two "select" statements listed above, and then type "list-file-
access". This displays the full information for these files only.
```

list-file-stats

produces the "file statistics" report generated from the most recently executed "file-save" or "account-save".

Statistics are produced from the "dm,file-of-files," file. The data that is displayed is as follows:

"r#" is the reel number (or volume number) that the file resides on (from the last file-save).

"seq#" is the file sequence number, which numbers from 1 to nn across the whole file-save.

"file#" is the file sequence number in the system's master dictionary.

"fname" is the file name.

"mod" is the modulo of the file.

"bytes" is the size (in bytes) of the file.

"items" is the number of items in the file.

"ovfitms" is the number of items residing in overflow frames (secondary filespace).

"frames" is the total number of frames used by this file.

"ptr-fms" is the number of pointer frames used by this file.

Syntax

list-file-stats {det-supp} {nopage} {lptr} {hdr-supp} col-hdr-supp} {ni-supp} {leg-supp | legend-supp"} {tcl-supp} {(options}

Options

"list-file-stats" is a FlashBASIC program "filter" which limits the command to just the options: b, c, h, n, and p, as well as the modifiers listed.

Example

```
sort dm,file-of-files, by name id-supp reel# seq# file# roll-on acct
"*** 'UVF'" fname mod total bytes total items total ovf-itms total
frames total ptr-fms grand-total "'U'" heading "PAGE 'P'    ** FILE
STATISTICS REPORT ** 'DCLL'" with stat# "2" lptr
PAGE1    ** FILE STATISTICS REPORT **    16 Jan 1997
r# seq#. file# fname...... mod bytes items ovfitms frames ptr-fms
3    89   134               17  4470     8             25       8
4   109   142 01             1   556     1              2       1
4   111   139 TOOLS          1  20632    20             22      21
4   113   141 TOOLS OBSLT    1  15080    30      28     17       5
4   112   140 TOOLS TOOLS    3  23226    30      18     29      21
                            --------- ----- ------- ------ -------
*** atp4                       359333   125     479    366     131
                            --------- ----- ------- ------  ------
*** tutor                      172970   945     355    210     129
                            ========= ===== ======= ====== =======
                              4170691 15323    6229   5546    2645
[405] 132 items listed out of 158 items.
```

list-files

produces a report of all file-defining items ("d-pointers") and synonym-defining items ("q-pointers").

The report that it produces is in two parts. The first part is a listing of all the "local" files. This is a listing of all files with a "d", optionally followed by a second character, in the "d/code" attribute (attribute 1 in the md).

This portion of the report contains the dictionary name, the base frame-id (fid), the dictionary modulo, the data-portions and their base fids and modulos, and the file's d/code if it is anything other than "d".

The second portion of the report contains those items defined as "synonym-defining items", or "q-pointers". This includes the target account and target filename.

Syntax

list-files {lptr} {(options}

Options

f Displays "d-pointer" files only.

p Directs outout to system printer, via the Spooler.

q Displays "q-pointer" files only.

Example

```
list-files (pf
```

list-jobs

produces a report indicating the status of all phantom controlled processes currently in the "dm,jobs," file.

"user-md" is the user-id of the user who submitted the phantom job.

"ln" is the line number from which the job was submitted.

"pib" is the line (port.number) on which the job is queued.

"status" is the current state of the job ("queue", "running", "done"). The number adjacent to the status (enclosed in parentheses) is the total number of seconds the job has been running (or the total number of seconds used if "done").

"start-dt" is the date the job was submitted.

"stop-dt" is the date the job finished.

"tcl-command" is the command or list of commands submitted.

Syntax

list-jobs {(options}

Example

```
list-jobs
PHANTOM JOBS STATUS AT 10:09:07 04 Dec 1994    PAGE   1
user-md ln. pib status..... start-dt... stop-dt.... tcl-command..
pxp pxp   1 122 done(3)     12/04 10:08 12/04 10:09 pxpjob x25p56
pxp pxp   1 121 done(12)    12/04 10:08 12/04 10:09 pxpjob x25p78
pxp pxp   1 120 done(7)     12/04 10:08 12/04 10:08 pxpjob x25p88
```

list-lines

Displays communications protocol information for every port attached to the virtual machine.

The report includes the following headings:

"Line#" is the pib or port.number.

"Baud/rt" is the baud rate of the port.

"Parity" is the state of parity: none, even or odd.

"Stop" is the number of stop bits.

"Data" is the number of data bits.

"DCD" is the data carrier detect status, ON or OFF.

"Xon/off" is the state of flow control, ENABLED or DISABLED.

"TTY" is the Unix tty device attached to the port.

On Unix Implementations, the Unix device name is also displayed.

Syntax

list-lines {(options}

Example

```
list-lines
Line# Baud/rt Parity  Stop  Data  DCD Xon/off       TTY
0     9600     NONE     1     8  OFF   enabled   /dev/tty0
1                              OFF
2     9600     NONE     1     8  OFF   enabled   /dev/tty2
3     9600     NONE     1     8  OFF   enabled   /dev/tty3
4     9600     NONE     1     8  OFF   enabled   /dev/tty4
```

list-lists

produces a sorted reports of all previously saved lists in the "pointer-file"

Syntax

list-lists {(options}

Options

* see "options: AQL".

Example

```
list-lists (p
```

list-lock-que

See "list-lockq".

list-lock-queue

invokes the "list-lockq" command.

list-lockq

lists the processes which are enqueued (waiting) to lock a frame, which is currently locked by some other process.

The display includes the following:

"Waiting line#" is the "waiter's" (the one waiting for the item) port.number.

"Waiting level" is the "waiter's" push-level.

"owner line#" is the "owner's" (the one locking the item) port.number.

"owner level" is the owner's push-level.

"fcb fid" is the File Control Block frame-id number.

"group fid" is the group Frame-ID.

"item id" is the item-id that is currently locked.

Example

```
list-lockq
Waiting         owner     fcb group
line# level  line# level  fid  fid  item id
0000    00                    0096B8
0038    01                    0535F5
```

list-locks

displays the current status of all system, FlashBASIC, Spooler, group and item locks.

All locks are displayed automatically unless one or more of the available options is provided with the command.

"System" Locks are the 10 internal system locks used during saves and etc.:

This lock position is unused.

nnn The port.number using this lock position.

"Basic" Locks are the 64 FlashBASIC program execution locks (used by FlashBASIC programs).:

This lock position is unused.

nnn The port.number using this lock position.

"Spooler" Locks are the internal spooler locks:

mq Master Queue Lock

iq Input Queue Lock

fq Forms Queue Lock

peq Permanent Entry Queue Lock

"Group" Locks:

nnnnn The frame-id, in decimal.

(nnnnnn) The frame-id (in hexadecimal).

Pib# The pib# (port) who has the item locked.

Lvl The level on which this group is locked.

Type The type of group lock. This may be either Rdo (for Read-Only) or Upd (for Update).

Filename The file in which the group exists.

"Item" Locks

nnnnn The frame-id, in decimal.

(nnnnnn) The frame-id (in hexadecimal).

PIB# The pib# (port) who has this item locked.

Lvl The owner's push-level number.

Hash The locked item's hash code.

Item-id The item-id which is currently locked.

Filename The name of the file where the item resides.

"File" Locks are special item locks which affect the whole file. These locks prevent any updates or normal items locks to a file while in effect. File locks are displayed in the item lock listing as having item id's of "*" and hashes of "0". These locks are set with the BASIC "filelock" statement.

When displaying item locks (by using the "i" option, it is optionally possible to display the locks of a remote D3 machine by specifying the host name. If a "*" is specified for the host name, then list-locks attempts to list the locks on all remote hosts.

Syntax

list-locks {host|*} {(options)}

Options

b Displays the 64 FlashBASIC execution locks.

g Displays group locks only.

h Displays host information instead of the group of item locks. This is useful for server systems for displaying the client host, pib, and level for each lock.

i Displays item-locks only.

p Directs output to system printer, via the Spooler.

q Displays the 4 Spooler queue locks.

r Displays remote item locks held by local processes. Note that these are "potential" locks and that they may list locks not actually held on the server if those locks have been removed in an abnormal fashion.

s Displays the 10 system locks.

Example

```
list-locks (i
Item Locks          PIB#  Lvl  Hash  Item-id              Filename
5834   (0016CA)     54   0   7E2C    21086                orders
2746   (000ABA)     68   0   0D1F    roy/month2           memos
183325 (02CC1D)     29   1   5DD7    3242                 entity
183197 (02CB9D)     22   2   7D9F    20946                entity
184453 (02D085)     66   0   279F    73291176             entity
110819 (01B0E3)     42   0   084D    tcl.list-locks       ap.doc
110908 (01B13C)      3   0   3E36    data.representation  ap.doc
133899 (020B0B)     82   0   6C2C    monitor              status
In this example, the display is limited to the current item locks.
list-locks
System Locks
   0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
0 ### ### ### ### ### ### ### ### ### ### ###
Basic Locks
   0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
0 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
1 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
2 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
3 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
Spooler Locks
   mq  iq  fq peq
0 ### ### ### ###
Group Locks        PIB#  Lvl  Type  Filename
442384 (06C00F)     13    1    Upd   xsym
Item Locks         PIB#  Lvl  Hash  Item-id            Filename
123034 (01E09A)     23    1 54B898A5 wm                 email
79174  (013546)     57    0 9472A038 hv                 tcl-stack
In this example, all locks are shown.
```

list-logoffs

produces a report showing the history of logoffs recorded in the "dm,errors," file, along with who logged off and the status of the logged off port.

Example

```
:list-logoffs
logoff user     logged off line     errors          type      lvl
============== ================= ============= ============== ===
50 danny ba     79 jane ba        basic ws bad  normal logoff  0
03/19/92 22:51
50 danny ba     61 joe qa         basic ws bad  normal logoff  0
03/19/92 22:50
50 danny ba     41 jon ba         basic ws bad  normal logoff  0
03/19/92 22:50
50 danny ba     36 rhb ba         glk tbl bad   normal logoff  0
```

```
03/19/92 22:50
57 sm ba        84 ch ba          child.pcb.err reinitialized  0
03/19/92 15:37                    ovf stack bad
6 rob ba2       47 rob ba2        basic ws bad  normal logoff  0
03/19/92 12:40
Legend:
"logoff user" is the port.number, user-id, and account name of the user that
logged  off the other port.  Beneath is the date and time they logged off the
other port.
"logged off line" is the port.number, user-id and account name of the user
that got logged off.
"errors" is the current workspace condition of the port (user) that got logged
off. There may be more than one error condition and may include:
"no errors, pbufbeg fid bad, lcl glk tbl bad, fopen table bad,  basic ws bad,
ibbeg fid bad, ovf stack bad, sws bad, fid bad,  db table bad, dcb bad, scb
bad, pcb bad, pcb links bad, pcb bad fid, child.pcb err".
"type" is the type of logoff command performed:
 normal logoff
 reinitialized
 did not respond
"lvl" is the current push-level of the port that got logged off.
```

list-macros

produces a sorted report of all items defined as macros in the specified account name.

Syntax

list-macros md.name {(option}

Options

* see "options: AQL".

list-menu

produces a documentation quality presentation format of one or more items defined as menus from the specified md.

D^3 menu items contain "me" in attribute one and must reside in the md.

Syntax

list-menu md.name {item.list*} {(option)}

Example

```
select md with a1 "me"
[404] 3 items selected out of 1477 items.
list-menu md (p
```

list-menus

produces a sorted listing of all items defined as "menus" in the specified md.

Syntax

list-menus md.name {(option)}

Options

* see "options: AQL".

list-obj

outputs descriptive internal information about programs compiled for Pick/BASIC and FlashBASIC.

Without any options the "list-obj" verb will output information on the "bp" file.

The displayed information includes the last compile date, last compile time, total frame usage, port number last compiled on, user name who last compiled, and account where last compiled.

Syntax

list-obj {file.reference}

Options

e outputs Pick/BASIC frame usage, FlashBASIC frame usage, total frame usage, beginning fid, and release version last of last compile.

p directs output to the Spooler

list-pibs

produces a report of each active port in the "dm,pibs," file, showing the port.number, the current user of that port and the current terminal/printer characteristics as defined by the "term" command.

The report includes the following:

"pib#" is the Process Information Block Number, or user's port number. The "*" (asterisk) indicates the current port.

"user" is the user-id (found in the "dm,users," file) that this user is currently logged on to.

"location" The value stored in the "dm,users," item in the "location" attribute. This attribute typically contains the user's name.

"device" is the settings used in the most recently executed "term" command for the terminal device this port is attached to.

Syntax

list-pibs {(options}

Options

* see "options: AQL".

Example

```
list-pibs
Page  1      dm,pibs,              09:44:44 21 Apr 1992
pib# user..... location......... device...............

  0                               79,25,0,7,1,8,80,59,IBM3151
  1                               79,24,0,7,1,8,80,59,WY-50
  2  lisa      Lisa Jones         79,24,0,7,1,8,80,59,WY-50
  3                               79,24,0,7,1,8,80,59,WY-50
  4                               79,24,0,7,1,8,80,59,WY-50
  5                               79,24,0,7,1,8,80,59,WY-50
  6                               79,24,0,7,1,8,80,59,WY-50
  7                               79,24,0,7,1,8,80,59,WY-50
  8  am        Andy Meyers        79,24,0,7,1,8,80,59,WY-50
  9  ss        Sam Smith          79,24,0,7,1,8,80,59,WY-50
 10  cm        Christine Mills    79,24,0,7,1,8,80,59,WY-50
 11                               79,24,0,7,1,8,80,59,WY-50
 12                               79,24,0,7,1,8,80,59,WY-50
 13                               79,24,0,7,1,8,80,59,WY-50
 14                               79,24,0,7,1,8,80,59,WY-50
 15                               79,24,0,7,1,8,80,59,WY-50
 16  dp        Dick Pick          79,24,0,7,1,8,80,59,WY-50
 17                               79,24,0,7,1,8,80,59,WY-50
 18  ff        Fred Flintstone    79,24,0,7,1,8,80,59,WY-50
The port numbers which have no user-id or name are not currently in
use.
```

list-procs

Invokes "listprocs" command.

list-ptr

see listptr (Spooler)

list-resizing

lists the status of files currently being resized.

The fields shown are as follows:

account Account where file resides.

file name The file's name.

pib Pib which is (or was) resizing file. Processes which are truly active at the moment will have item locks held for the appropriate resizing item.

current Current relative group being re-hashed.

max Maximum groups to re-hash.

Example

```
list-resizing
Page     1      dm,resizing,
account. file name. pib. current. max...
dm       mydata       20       50    101
This indicates that the "mydata" file in the "dm" account is currently being
resized.  The process is currently about half way through re-hashing the file
because the current relative group is 50 which is about half of the required
maximum, 101.
```

list-restore-errors

Displays errors recorded during restore.

Produces a report from the "restore-errors" file. The "restore-errors" file is created during a file restore process only if there are errors encountered during the restore.

Syntax

list-restore-errors

lre

list-system-errors

produces a sorted report of error conditions logged into the "dm,errors," file.

Since this is a standard macro which executes the access "sort" verb, any options available to sort may be passed through to be executed with the command.

The most recently recorded errors appear first on the report.

Syntax

list-system-errors {(options}

list-tandems

lists the devices currently involved in a tandem association.

Syntax

list-tandems {(p}

Options

P Printer output.

list-users

produces a report of users currently logged on to the system.

The report includes:

pib#: port.number. "*pib" = current port.

pid: The process id number.

user: The user-id.

udate: The date this user logged on.

utime: The time this user logged on.

acct: The account name currently in use.

mdate: The date logged on to this account.

mtime: The time logged on to this account.

location: location (usually the user name).

The report is produced from the "dm,pibs," file.

Syntax

list-users {user-id} {(options}

Example

```
This form shows all current users:
list-users
pib# pid.. user. udate utime account. mdate mtime location...
   1 25102 dp    06/04 10:28 dm       06/04 10:28 Dick Pick
*  2 11884 ff    06/04  6:49 krb      06/04 13:48 Fred Flintstone
   4  7536 am    06/04 13:19 ba       06/04 13:20 Andy Meyers
```

```
   6 10354 sj    06/04  8:21 ts       06/04  8:22 Steve Johnson
   7 13666 ss    06/04 10:28 qa       06/04 10:29 Sam Smith
  10 26534 lisa  06/04 16:02 ts       06/04 16:02 Lisa Jones
  16 19992 bob   06/04 14:30 qa       06/04 14:30 Bob Bogus
  20  8615 cm    06/04 10:14 krb      06/04 15:45 Christine Mills
This form shows a specific user-id:
list-users bob
pib# pid.. user. udate utime account. mdate mtime location...
  37 20381 bob   07/17 10:31 krb      07/17 10:31 Bob Bogus
* 53 18348 bob   07/17  9:35 krb      07/17 9:36  Bob Bogus
This form shows all users whose user-id begins with "s".
list-users s]
pib# pid.. user.. udate utime account. mdate mtime location...
   6 40319 sj     06/04  8:21 ts       06/04  8:22 Steve Johnson
   7 39577 ss     06/04 10:28 qa       06/04 10:29 Sam Smith
```

list-verbs

produces a sorted listing of all items defined as "verbs" in the specified account name.

Syntax

list-verbs {md.name} {(options}

Options

* see "options: Spooler".

list.lines

Invokes "list-lines" command.

listabs

see listabs (Spooler)

listacc

produces a report of system usage statistics from the "dm,accounts," file for all accounts, or for a specified account, or for a specific user-id.

The report contains the following:

"ACCOUNTS" is the item-id column for the report. This contains the user-id or a pcb-fid address.

"MD" is the account name to which the user logged. This value is a "controlling" value for the remaining values in the right columns.

"L-DATE" is the date the user logged on.

"L-TIME" is the time the user logged on.

"ACCT-TIME" is the total hours and seconds that this account was used.

"CPU" is the amount of CPU units used during this session. "Units" will vary from system to system, but is generally measured in 1/10th's of seconds.

"PAGES" is the number of pages of output sent to the Spooler.

Syntax

listacc [{account.name} | {user-id}] {legend-supp} {tcl-supp} {det-supp} {nopage} {lptr} {ni-supp} {(options}

Options

p directs output to the spooler.

Example

```
listacc 'bob'
Page   1      ACCOUNTS               08:34:01 22 Apr 1992
ACCOUNTS.... MD........ L-DATE L-TIME ACCT-TIME CPU. PAGES
bob          REF        04/02  15:38  00:11:13   497     0
             REF        04/02  21:19  05:24:14    55     0
             REF        04/03  06:20  00:07:07  3339  1222
             REF        04/03  13:41  00:46:15   236     0
             REF        04/03  16:38  01:04:08   172     3
```

listbi

see listbi (Unix)

listc

invokes the "list-label" command to produce a columnar display of the item-ids in the specified file, in the order of appearance in the file.

Any valid selection criteria may be passed through to AQL for processing. The default is a 4 column display. If a numeric option is entered, then that number of columns, up to 6, will be displayed.

Syntax

listc file.reference {sellist} {heading "heading text"} {(n}

Options

* see "options: AQL"

Example

```
listc md with a1 "d]"
03 Apr 1992    Items in File md         Page   1
ap.doc.platforms  ap.doc.files   ap.doc.release    HELP-FILE
ap.doc.types      neat.stuff     ap.doc.processor  r83.md
spool-file        dm.newac       ap.doc.status     pointer-file
msb               dm.md          ap.doc.categories doc.old
phone.save        book           temp              ap.doc
wsym              BP             macros            ad.sup
PCPROGS           doc            schedule
[405] 27 items listed out of 1445 items.
```

listconn

produces a report of all items defined as "connectives" (also called "modifiers") in the specified file.reference.

The default file.reference is "md".

Any item in the md with the letter "c" as the first character in attribute one is a "connective". The items in the report are sorted by the value of attribute one.

Syntax

listconn {file.reference} {nopage} {lptr} {hdr-supp} {col-hdr-supp} {ni-supp} {legend-supp | leg-supp} {tcl-supp} {(options}

Options

b,c,h,n,p

Example

```
listconn
22 Apr 1992    Connective Listings For DM
krb,,....... A1.........................
```

```
secondary    c*
fill         c+
by-exp-dsnd  c-
by-exp       c.
by-dsnd      c/
by           c0
&            c1
and          c1
within       c2
each         c3
every        c3
if           c4
with         c4
ifno         c4d
without      c4d
duplicate    c5
only         c6
```

listdict

sorts the given file in order of references to the attribute count (ac) and produces a report showing the contents of the various attribute-defining items.

If no file.reference is specified, the report defaults to using the current md.

Only items which contain "a" "s" or "x" in attribute one are presented. This eliminates showing "d-pointers" in this report.

"list-dict" is an alternate form to this command.

Syntax

listdict {file.reference} {(options}

Example

```
listdict
This produces a report of the attribute-defining items in the md of the
current account and directs the output to the terminal.
listdict ap.doc lptr
This produces a report from the dictionary of the "ap.doc" file and directs
the output to the Spooler.
```

listf

Invokes "list-files" command.

listfiles

produces a report showing all file-defining items and synonym-defining items in an optionally specified filename or account name. The default filename is "md".

Syntax

listfiles {file.reference} {(options}

Options

b,d,h,i,k,n,p,r (standard AQL options, except "c" ("col-hdr-supp")), plus:

c Produces filename listing only, in four-column format.

Example

```
listfiles
This produces a report of all file-defining items in the md of the current
account and directs the output to the terminal.
listfiles (kp
This is exactly like example 1, but the output is directed to the Spooler and
```

```
the "legend-supp" ("k" option) is invoked.
listfiles dm,,
This produces a listing of all files in the "dm" account using a file path.
```

listfs

see listfiles

listpeqs

see listpeqs (Spooler)

listprocs

produces a sorted report of all items defined as procs in the specified filename.

The default filename is "md" when none is specified.

Syntax

listprocs {account.name} {legend-supp} {tcl-supp} {det-supp} {(options}

Options

* See options in "see also".

Example

```
listprocs (p
listprocs dm,, leg-supp
```

listverbs

produces a sorted listing of all items defined as "verbs" in the specified account name.

Syntax

listverbs {account.name} {(options}

Options

p,n,h,c and b (standard AQL options)

lm

invokes the "list-menu" command to produce a listing of menus.

Example

```
lm
```

load.mon

Invokes "load.mon2" command.

load.mon2

displays a graphical map of the "duty load" of the D^3 System, taking a "snapshot" of the system at a given time interval.

"snap delay" is the number of seconds to delay between samples. The default value is ten seconds.

The following information is displayed:

"snapshot delay" is the number of seconds delay between snapshots of the system.

"last" is the time of the last snapshot.

"cnt" is the number of snapshots taken.

"smax" (unknown).

"hmax" (unknown).

The numbers up the side indicate the percentage of system resources consumed. The one in bold indicates the maximum value for the snapshot.

The numbers across the center of the screen are port.number references (0 - 73).

The (-) across the bottom indicate the active ports.

The (+) indicates the current port.

The vertical bars of asterisks and letters are the user-id names, the accounts the ports are in. Asterisks (*) fill out the vertical bar, up to the percentage of system resources consumed by this process.

Syntax

load.mon2

snap delay (10):number.seconds

Example

```
load.mon2
snap delay (10): 5
```

lock-beep

enables/disalbes beeping on lock failure.

By default, any lock failure causes repeated beeps to be emitted. To disable this feature on the current line, use "lock-beep (f"

Syntax

lock-beep {(options}

Options

f Disables beeping.

n Enables beeping

lock-beep-off

disables lock failure beeping.

lock-beep-on

enables lock failure beeping.

Syntax

lock-beep-on

lock-break

Enables or Disables the ability to unconditionally break during a lock failure.

By default, the user is allowed to break during a lock failure even if the break has been disabled by the user. This is to allow escape from deadlock situations.

On communications lines, this can cause trouble since certain hardware can generate erroneous break signals. In this case, it is recommended to disable this function by doing "lock-break (f".

Syntax

lock-break {(options}

Options

n Allows breaking during lock failure.

f Disallows breaking during lock failure.

lock-break-off

disables breaking during lock failure.

lock-break-on

enables breaking during lock failure.

lock-frame

"core locks" a frame into real memory.

This prevents the frame from being flushed to disk for any reason.

The "." optionally preceding the "fid" indicates that the fid is being referenced by its hexadecimal reference. Without the period, the reference is assumed to be decimal.

Syntax

lock-frame {.}fid

Example

```
lock-frame 1672
[450] Frame locked at: 59E0 (hex)
```

log-msg

records the time and date that a particular macro is executed. The log entry is placed into the "dm,errors," file. This command is executed by the "power-off" process.

This verb can be used within a macro, for example, to note when the macro was executed.

Syntax

log-msg message

Example

```
log-msg application event
sort errors by date
page   1      errors                07:31:34 02 Oct 1991
errors.... date. time. r mode-addr........... user   error.........
867553651  10/01 14:54   Coldstart after      dm (0) log-msg
                         abnormal shutdown
867625376  10/02 07:02   Coldstart after      dm (0) log-msg
                         normal shutdown
867626224  10/02 07:17   application event    dm (1) log-msg
[405] 3 items listed out of 3 items.
Legend:
"errors"      is the system-generated job number (item-id) for this item.
"date"        is the date that this error was logged.
"time"        is the time that this error was logged.
"r"           is the register address (0 - f) which triggered the error
condition (e.g. forward link zero, referencing illegal frame aborts, etc.)
"mode-addr"   is the description of the error, or address of the program when
it generated the error.
"user (pib)"  is the user-id who was running the process along with the
port.number.
"error"       is the process which caused this error message to be placed in
the log.
```

log-status

displays the current status of the transaction logger.

Example

```
log-status
[609] Transaction logger not started.
```

logoff

terminates a process on a given port and sends the port to the "logon" prompt.

If the "port.number" is not specified, it is requested.

The logoff process first attempts a "normal" logoff by simulating a "<break>" and "off" sequence on the target port. If that time out, logoff escalates and "forces the process into the debugger and simulates an "off". If THAT fails, the target port is reinitialized, without attempting to clean up or recover its workspace.

"phantom" ports, the scheduler, the transaction loger and those nasty "hung" ports may now be properly logged off.

It is not recommended to logoff the transaction logger before entering "stoplog".

A process' locks are released during a logoff and the tape(s) are detached.

The "errors" file contains a history of logoffs. See "list-logoffs".

Syntax

logoff {port.number}

Options

k kills a shell if process is "shelled out"

Example

```
logoff<return>
line:677<return>
[534] logoff successful.
logoff 677<return>
[534] logoff successful.
```

logoff.error

interprets the logoff reason for the "errors" file.

logon

logs on a port, other than the one currently being used, onto a specific account.

The process prompts with the following:

line,user-id,user-password,md,md-password:

"line" is the port.number to log on.

"user-id" is the user-id (must be in "dm,users,").

"user-password" is the user-password (if defined).

"md" is the account name to log on to.

"md-password" is the account password (if defined).

The port can NOT already be logged on.

This command is also useful for invoking "background" tasks on another port. Background or "phantom" tasks may also be invoked using the phantom processor (see the "z" commands).

Syntax

logon {(p)}

line,user-id,user-password,md,md-password:

Options

p Starts process as a "phantom" line. Note that this does not have the same meaning as what is often considered a "phantom", where the process spawns a new process workspace. Rather, this option requires an existing unused port, just like any other port being used for a remote logon. In this context, "phantom" means that it is treated like a Spooler process. This means that the logon sequence is suppressed, which turns off the display of the time and date logged on, the "welcome" message" and the automatic update of the ACC file.

Example

```
logon<return>
line,user-id,user-password,md,md-password:0,dm,,dm
This logs line 0 (zero) on to the user-id "dm" with no user password,
then on to the account "dm" which also has no password.
```

logon-lock

turns on or off the logon lockout feature.

When the logon lockout is enabled, the logon system will lock up after three failed logon attempts. At this point, it is necessary to type "(space)(return)(return)" to continue. This feature is useful for synchronizing a port which is connected to a device which might otherwise echo the logon message.

With the logon lockout disabled, a 1/2 second delay occurs after 3 failed logon attempts for security purposes, but logon may then proceed normally.

Syntax

logon-lock {({a}{f|n}{starting.port{-ending.port}}

Options

a Affect all lines.

f Turn logon lockout off.

n Turn logon lockout on. This option is assumed if the (f option is not present.

starting.port{-ending.port} Affect a range of ports. If not specified, then the "logon-lock" command assumes the current port.

logto

terminates accounting on the current account, then moves to another specified account. If a password is present, it must be provided. Passwords are case-sensitive.

If the password is omitted and required, the system prompts for it.

It is possible to "logto" another account while at a "pushed" level. When a <return> is issued at the TCL prompt, the process automatically "logs back to" the original account unless the (F option was specified for the "logto" verb. In that case every "pushed" level below the current one also logs to the new account. Those pushed levels are still running the programs they were originally running, they are just in a new account now.

Any tape or magnetic media devices attached to the process when logging to another account at another level remain attached in the new account.

Syntax

logto account.name{,password}

to account.name{,password}

Options

F The F option forces all pushed levels to also log to the new account.

Example

```
logto dm
< Connect time= 217 Mins.; CPU= 46 Units; LPTR pages= 0     >
logto dm,mypassword
< Connect time= 217 Mins.; CPU= 46 Units; LPTR pages= 0     >
Legend:
"connect time" is the number of minutes the current account was in use.
"cpu" is the number of cpu units used by the account. CPU units vary from
system to system, but are generally recorded as 1/10th's seconds. Note: The
cpu units shown are about 100 times smaller than the cpu units shown in the
FlashBASIC 'system( 9 )' function.
"lptr pages" is the number of pages sent to the Spooler.
```

loop

indicates the beginning of a loop in a paragraph.

Note that the "loop" statement does nothing by itself. It must be followed by a "repeat" statement to cause the looping effect.

Syntax

loop

loop

performs any TCL command a specific number of times, or continuously, until any key is pressed on the keyboard.

The "loop.count" indicates the number of times to execute the command. If "loop.count" is omitted, the loop is infinite. To stop the loop, press any key on the keyboard.

The "sleep.count" indicates the number of seconds for the process to "sleep" between each command execution. If "sleep.count" is not specified, the command repeats immediately.

Syntax

loop {loop.count} {\sleep.count\} TCL.command

Example

```
loop 3 \5\ ovf
ovf
overflow: 34860  reserve: 1024  blocks: 7  b-tree levels: 1  leaves:
1
ovf
overflow: 34924  reserve: 1024  blocks: 34  b-tree levels: 1  leaves:
1
ovf
overflow: 34909  reserve: 1024  blocks: 17  b-tree levels: 1  leaves:
1
This executes the "ovf" command 3 times, sleeping 10 seconds between
each execution.
```

loop-on

invokes the "loop" command and performs any TCL command a specific number of times or continuously, until any key is pressed on the keyboard.

Example

```
loop-on where
```

lp

invokes the "list-pibs" command.

lq

see listpeqs (Spooler)

lre

see "list-restore-errors".

lu

invokes the "list-users" command.

maxusers

displays the number of licensed D3 users on the system, and/or sets the system in single or multi-user mode.

The "single-user mode" is defined as the state where only the process which executes the command is allowed on the system.

If the process which executes the command is not port 0, though, port 0 will always be allowed in. This makes the command forgiving to logon, in the case when the terminal which sets the single-user mode becomes unavailable (following a modem problem, for instance).

All other ports are optionally logged off before entering the single-user mode, after sending two warning messages, separated by an optional grace delay. Confirmation is required before logging a user off.

A message is logged in the "errors" file when entering single or multi-user mode.

Syntax

maxusers {(options}

Options

number (integer number) Defines the "grace" period delay in seconds before logging users off when entering single-user mode. The default is five seconds.

f Logs off users currently logged on before entering single-user mode. A warning message is sent to users, unless the "n" option is also specified. When a user is logged on, confirmation is asked, unless the "u" option is also specified.

l Suppresses the message log in the "errors" file when changing the mode.

m Sets multi-user mode. The number of allowed users is set back to the maximum user license.

n Suppresses the warning message to users when entering single-user mode (valid only with "f" option).

p Prints the results.

q Quiet mode. Suppresses all output to the terminal.

s Sets single-user mode.

u Unconditionally logs off users currently logged on when entering single-user mode (valid only with "f" option).

Example

```
maxusers
Number of licensed users            : 128
Number of users currently logged on : 8
Number of users allowed on system   : 128
Number of ports on system           : 128
maxusers (sf30
This enters single-user mode, allowing a grace period of 30 seconds after
issuing the first warning message, before logging users off. This command
displays the following information, after entering single-user mode:
Number of licensed users            : 128
Number of users currently logged on : 1
Number of allowed users             : 1
Number of ports on system           : 128
maxusers (m
Number of licensed users            : 128
Number of users currently logged on : 8
Number of users allowed on system   : 128
Number of ports on system           : 128
Enters multi-user mode.
maxusers (snufq
This enters single-user mode ("s"), logs all users off unconditionally ("f"
and "u"), immediately -- without any warning ("n"), in "quiet" mode ("q") --
so no output displays.
```

md-restore

invokes the "account-restore" command.

mds.type

is called by "list mds" to determine the account type ("q", "x", "d", etc.), so that "list mds" will not list dictionary items, only accounts.

message

see "send-message".

mirror

attaches the current process (the "master") to one or more "slave" processes. All characters input or output to or from the "master" process display on each "slave" process.

The process which initiates the "mirror" command is the "master". The process being attached is the "slave" or "target".

The target "slave" device (the one(s) being "linked" to the current device) must be available on the given "port.number(s)". It (they) may not be already attached to another process.

Multiple "slave" terminals may be designated and must be indicated in ascending order.

The "master" process may add additional "slaves" without terminating existing "slaves".

A "hotkey" sequence disconnects the link. The default "hotkey" sequence is <escape>x. If an "esc-data" has not been issued previously, the process issues it automatically, so the process will not get locked out from being able to suspend "mirror".

Either the "master" or "slave" can terminate "mirror" with the "hotkey" sequence. If a "slave" terminates, only that particular process is taken out of the "mirror" mode. If there are other "slave" processes still active, they are not affected. If the "master" process terminates, all "slaves" are disconnected.

On all Unix implementations, if the D3 process is not connected to the virtual machine, then "mirror" is not allowed, even if the "where" verb with a "z" option shows the port as being available.

Syntax

mirror port.number{,port.number...} {(options}

mirror port.number{-port.number...} {(options}

Options

d Designates the sequence of characters to terminate "converse" mode. Each character is provided in its hexadecimal equivalent of its ASCII value. Each character is separated by a comma. For example, "d/41,42,43,44", indicates that an "ABCD" will disconnect the link.

x Terminates mirror mode on target device number.

Example

```
mirror 2,13,32
Attaches terminals on ports 2, 13 and 32 to the current process.

mirror 2,5-10
Attaches port 2 and ports 5 through 10.
mirror 3 (d/2b,2b,2b
Attaches port 3 and sets the hotkey sequence to "+++".
mirror 42 (x
Terminates mirror mode on port 42.
mirror 1-20
Attaches ports 1 through 20 to the current port.
```

mlist

produces formatted assembly source code listing of the specified itemlist.

Syntax

mlist file.reference itemlist* {(options)}

Options

line.number{-line.number} Specifies (source) line number, or range of line numbers to list.

e Outputs error lines only.

m Outputs macro expansions.

n Activates nopage function on output to the terminal.

p Directs output to system printer, via the Spooler.

s Suppresses object code display.

z Inhibits entry into editor when used with "e" option.

mload

loads assembled object code into the virtual memory address defined in the routine.

Syntax

mload file.reference itemlist* {(options)}

Options

n Returns item checksum data without actually loading item.

v Verifies mismatches and errors only.

mmvideo

tests the visual effect capabilities of a memory-mapped monitor on PC-class implementations only.

modem-off

invokes the "xonoff" command and passes it an "f" option.

This disables x-on/x-off on the current line or a given line.

modem-on

invokes the "xonoff" command and passes it an "n" option. This enables x-on/x-off on the current line or a given line.

monitor-status

produces a "snapshot" listing of information related to memory-resident frames.

Syntax

monitor-status {(option}

Options

* see "options: TCL"

Example

```
monitor-status
Monitor Statistics
==================
process activations    :   0
Idle time              :   0
# of frame faults      :   33
# of writes            :   106
Failed buffer search   :   0
Read queue full events :   0
Write queue full       :   0
Disk errors            :   0
Buffer Table Information
========================
frames with  0  regs attached : 8,064
frames with  1  regs attached : 364
frames with  2  regs attached : 156
frames with >=3 regs attached : 158
Write required                : 1,251
Read queued                   : 0
Memory locked                 : 1,588
Frame referenced              : 5,645
Write queued                  : 0
Top of hash                   : 7,142
```

mono

switches output to a monochrome display from a color display, or sets the visual attributes of the monitor.

On a system with both a color and monochrome display monitor, the "mono" command switches output from the color monitor to the monochrome display.

Switches:

/b or /blink activate blinking

/f or /full full intensity foreground

/h or /half half intensity foreground

/nb or /noblink stop blinking

/nu or /nounderline stop underlining

/nr or /noreverse stop reverse video

/r or /reverse activate reverse video

/u or /underline activate underlining

Syntax

mono {/switch}

move-file

moves a file descriptor from one dictionary to another. The dictionary may be file-level, account-level (master dictionary) or system-level (mds or system).

The verb requests the destination dictionary with the prompt 'TO:'. The following responses are allowed:

(dict.name or (dict.name file.name or file.name

If the "dict.name" is missing, the source file dictionary is assumed.

If the "file.name" is missing, the source "file.name" is assumed.

If both the "dict.name" and the "file.name" are omitted, no action is taken.

"move-file" follows a hierarchical approach to file transfer. A file must be moved to the SAME LEVEL as it was created. That is, a data file to a data file, master dictionary to master dictionary, and so on.

Syntax

move-file file.reference {file.name}

to: (file.reference {file.name}

Example

```
move-file dict bp bp
to: old.bp

This effectively changes the name of the data-level d-pointer for the "bp"
file to another item in the dict of bp called "old.bp". "old.bp" would
subsequently be available using the reference "bp,old.bp". This also removes
the "default" d-pointer to the "bp" file, and any reference to the "bp" data
section would result in the message, "data level descriptor missing".
move-file md bp
to: old.bp
This changes the "bp" file to "old.bp". Note: this does NOT change the name of
the data file within the dictionary of "bp".
move-file md bp
to:(archive,,
This moves the "bp" file from the current account to the "archive" account md.
move-file dict bp bp
to:(dict archive,bp,old.bp,
This moves the data portion of the "bp" file from the current account to the
account called "archive", to a dictionary called "bp", and renames it
"bp.old".
move-file mds,, pa
to:pete
This renames the account "pa" to "pete". Note: "rename-file" does not work at
the mds ("system") level.
```

msg

see "send-message".

mul

multiplies the first integer number provided by the second integer number provided and displays the result as an integer number.

Syntax

mul integer.number integer.number

muld

multiplies the first integer number provided by the second integer number provided and displays the result as an integer number.

Syntax

muld integer.number integer.number

Example

```
muld 13 7
91
```

mulx

multiplies the first hexadecimal number provided by the second hexadecimal number provided and displays the result as a hexadecimal number.

Syntax

mulx hex.number hex.number

Example

```
mulx 8 1C
E0
```

mverify

compares assembled object code in virtual memory against the corresponding itemlist in specified filename. Any frames with "mismatches" are reported.

Syntax

mverify file.reference itemlist* {(options)}

Options

a Outputs columnar listing of all mismatches.

e Outputs errors only.

p Directs output to system printer, via the Spooler.

net-errors

sets/displays error handling behavior when accessing remote files.

With no options, the current error handling behavior is displayed. The following options are included:

Remote Error Logging - When enabled, this logs all remote errors to the errors file.

Remote Error Notification - When enabled, all remote errors are displayed on the users terminal.

Remote Error Intervention - When enabled, all remote errors force the user to select an action from a menu. Note that even with this option disables, intervention is required when an update operation fails.

If options are specified, then the error handling behavior is changed and then re-displayed.

This command affects the current line only.

Syntax

net-errors {(options)}

Options

c Clear all error handling options.

l Enable error logging.

n Enable user notification.

i Force user intervention on all errors.

s Suppress the option display

net-status

displays statistics on local network usage.

The information displayed is identical to that shown by choosing the "Display server status" followed by the "Display server statistics" options from the main menu of "network-setup". See "network-setup" for more explanation on this data.

network-setup

allows the setup and control of a D3 network. If no arguments are supplied, a menu is displayed. This is the normal form of operation. See the section "network, General", for a discussion of the important notions of the network configuration.

- Using the menus

  All operations are controlled through menus. If the terminal allows it, arrow keys can be used where indicated:

ENTER	Validate the highlighted choice.
number	From 0 to 9. Select the corresponding choice. '0' selects the option 10.
CTRL-N	Move cursor down. (down arrow)
CTRL-B	Move cursor up. (up arrow)
CTRL-X	Cancel. Applicable only when input is requested.
ESC	Quit. Go back to previous menu, or back to TCL. This key can be used to terminate all menus.
Q	Quit. Go back to previous menu.
X	Exit. Go back to TCL from any menu.

  When the cursor is moved to a new field, a short help is displayed in the message area.

- Screen layout :

  The screen is divided in two sections:

  - The menu section, where menus are displayed.

- The message section, where results, messages or help are displayed.

- Definitions :

  host—definition for a local or remote D3 machine. There should be one for each system.

- Main Menu

  1 Start network servers
  Starts up all local network servers. Before running this command, it is necessary to define the network using the "Define local hosts" and "Define remote host" options. The server status is automatically displayed after network servers are started.

  2 Stop network servers
  Stops all local network servers.

  3 Server status
  Displays the status of all local servers.

  4 Server statistics
  Displays transaction statistics for all local server processes, such as reads & writes per second.

  5 Server pid status
  Displays the Unix process status of each server process. Note that the Unix processes all appear to have been spawned by Unix command which started the primary server. This is due to the way the servers are configured and does not indicate an error.

  6 Network status
  Displays remote files open by each process.

  7 Define the network
  Allows the user to define the way the network is set up. This sub-menu is documented below.

  8 Exit
  Exits back to TCL.

- Network Definition Menu

  1 List all hosts
  Displays an access listing of all defined hosts, both the local and all remotes.

  2 Print all hosts
  Lists all hosts to the currently selected printer.

  3 Define local host
  Configure the local netork hosts. If you are defining an entire network, use option 9 Define all hosts. The following input fields are requested:

  Host Name :

  Any alpha-numeric string. This will be the D3 name of the local host. This field defaults to the name of the Unix machine if no local host is defined.

  Optional Host Description :

  An optional description of the host. The physical description of the machine or a description of it's location, or the name and number of it's system administrator would be helpful here.

TCP Name/Address :

The TCP Name/Address where the D3 machine exists. For the server, this should be the local Unix host name.

TCP Service Name/Number :

The TCP service number. Normally, the default of "pnfs" is correct as this is the default D3 service name. An alternate service is only necessary if there is more than one D3 virtual machine on the same Unix box.

Security Options :

These tune the security parameters of the server. Without any parameters, remote users may connect and disconnect freely and are given the same permissions as they have on their original client machines. Each open request from a client machine passes the retrieval and updates keys assigned to the user on that client to the server. The server then utilizes those client keys to open the requested file.

If the "A" option is used, the client retrieval and update keys are ignored, and the connection is refused completely if the passed user name and password do not match a user/password combination on the local server. The server looks for "ClientHostName:ClientUserName" or simply "ClientUserName" when looking for a user match.

If the user name and/or password must be different on the local server than on the remote client, the client user can utilize the "SET-REMOTE-USER" command to perform automatic user/password mapping when connecting to that server.

If the "H" option is not used, the server disconnects unused connections after a certain time period. In this situation, it may be unclear to the administrator which connections are being utilized by which remote users. If the "H" option is set, all connections are held until the"H" option is set, then all connections are held until the remote client logs off. This requires one server for each remote client, but helps prevent "snooping" and "spoofing" security problems, and makes system administration simpler as the list-users command always shows the connected users.

Transmit timeout :

Time (in seconds) allowed for each remote operation. After this time, the client assumes the server is down and follows the error path. This value should be increased when the server is slow and 'no response' errors are randomly encountered.

Accept Timeout :

Time (in seconds) during which a server process remains bound to a client when no operations are performed. A setting of 0 disconnects after every operation and is not recommended. For most situations, a value between 1 and 10 will provide a good balance between too many TCP connection in TIMEWAIT state and too few server processes. Note that server process remain bound to it's client when it holds item locks for that client, regardless of the Accept Timeout value.

Initial Server Processes :

The number of server processes that are started when the network is started. To change the number of servers, you must shut down the network, change this number, and then restart the network.

Host ID Number :

The host ID number is used to differentiate between different virtual machines on the same Unix host. If you only have one D3 virtual machine on each Unix machine, this number is not needed. If you have more than one D3 virtual machine on a Unix host, then choose small numbers (less than 128) for each virtual machine on that host.

Confirm (y/n/q) :

'y' to confirm the host modification. 'n' to go back to any of the previous fields. 'q' to quit and abandon.

4 Define remote host
Configure the remote network hosts. After choosing this option, another menu appears asking which host to edit as well as a "New Host" option to create a new host. Select the "New Host" option when intially configuring a network. Next, the following input fields are requested:

Host Name :

Any alphanumeric string. This will be the D3 name of this remote host.

Optional Host Description :

An optional description of the host. The physical description of the machine or a description of its location, or the name and number of its system administrator would be helpful here.

TCP Name/Address :

The TCP Name/Address where the D3 machine exists.

TCP Service Name/Number :

The TCP service number. Normally, the default of "pnfs" is correct as this is the default D3 service name. An alternate service is only necessary if there is more than one D3 virtual machine on the same Unix box.

Options :

These are driver-specific options. These may be left blank in the simplest case.

The host ID number is used to differentiate between different virtual machines on the same Unix host. If you only have one D3 virtual machine on each Unix machine, this number is not needed. If you have more than one D3 virtual machine on a Unix host, then pick small numbers (less than 128) for each virtual machine on that host.

Confirm (y/n/q) :

'y' to confirm the host modification. 'n' to go back to any of the previous fields. 'q' to quit and abandon.

5 Define all host
Configure all hosts on a D3 network. Use this option to configure an entire D3 network from one station. This option, when combined with the next three, allow a user to enter configuration data for all hosts, both this local one and all remotes, then dump the

information to tape. That tape is then loaded on each seperate virtual D3 machine. See menu option 7 for details for each prompt.

6 Dump host file
A tape is selected, and network items in the dm,hosts, file are dumped to it.

7 Load host file
A tape is selected, and items prepared by option 10 above are loaded from that tape.

8 Declare local host
After network host items are loaded from tape during option 11 above, the user must tell the system which host item describes the local D3 virtual machine. This option presents the user with a list of hosts, the user picks one the one that describes this machine.

9 Back to main menu
Exits back to the main menu.

- Non Menu Operation :

  It is possible to perform some operations from TCL by specifying a 'command' on the TCL line. This for is useful to perform some automatic commands in macros.

  'command':

  start

  Start the local server processes.

  stop

  Stop all local server processes.

  status

  Display the status information for all local server processes.

  statistics

  Display the transaction statistics for all local server processes.

Syntax
network-setup {{command}}

Options
Q Quiet. Valid only for the non menu operation. Supresses all messages.

network-status

displays remote files opened by a line.

You can specify one line number, a list of line numbers, a range of line numbers, or all line numbers as follows:

n,m Displays information for line n and line m

n-m Displays information for lines n through m, inclusive.

n,m-o Displays information for lines n and m through o, inclusive.

* Displays information for all lines.

Syntax
network-status [*|ports] {(options}

Options

a Displays all file descriptors in workspace, even closed ones.

r Displays information for all drivers, default is TCP/IP only.

v Displays internal fields - output can change between releases.

nframe-index

produces a total of the number of frames in use by a specific "b-tree" index, or all indices if indicated by an "*" (asterisk).

"a.code" specifies the "a (algebraic)" processing code to be used in forming the keys to the index. The processing code must include an attribute number.

The "*" (asterisk) designates "all" indices. Either an "a" processing code or an "*" (asterisk) must be specified.

Syntax

nframe-index file.reference a.code

nframe-index file.reference *

Example

```
nframe-index invoices a1
Total of 67 frames counted.
This indicates that 67 frames are in use by the index which defines attribute
one of the "invoices" file.
nframe-index invoices *
nframe-index invoices a1
Total of 67 frames counted.
nframe-index invoices a2
Total of 29 frames counted.
nframe-index invoices a3
Total of 53 frames counted.
nframe-index invoices a23
Total of 18 frames counted.
nframe-index invoices a27
Total of 105 frames counted.
nframe-index invoices a0
Total of 26 frames counted.
```

off

stops processing at all levels on the current line and returns control to the initial logon prompt. This command is also valid from the system (virtual) debugger and the FlashBASIC Debugger.

Example

```
off
<  connect time= 230 mins.; cpu= 29707 units; lptr pages=12  >
See the "logto" command for an explanation of connect time, cpu and lptr
pages.
```

okidata

changes characteristics on okidata printers.

op

Invokes OP on specified item(s).

overflow

displays the system overflow table which contains the addresses of available disk frames. All unused frames are available for use by the Spooler, the D^3 file system, and process workspace.

The overflow table is kept in the "B-tree" format. Each node in the "B-tree" may contain up to 120 elements.

Legend: (see examples)

"overflow" is the total number of frames available.

"reserve" is the number of frames held in reserve (must have been previously established with the "set-ovf-reserve" command). These frames are used only when all overflow frames have been used and are intended to provide space on an emergency basis for system functions such as file-saves.

"blocks" is the number of contiguous blocks of storage.

"b-tree levels" is the number of levels in the b-tree that holds the frame addresses.

"leaves" is the number of leaves (nodes on the final level) on the b-tree. All frame addresses are stored at the "leaf" level.

Syntax

overflow {(options}

Options

a Lists the start and end fids for each block as well as their sizes. When using the (a option, the user may filter the fids displayed by specifying a high and low limit. If the (b option is used along with the (a option, then the display will show the block start and end fids plus the sizes, but the user can filter blocks based on size.

b Lists total number of frames for each contiguous block of overflow. When using the (b option, the user may specify a range of sizes as well.

c Only counts frames which are displayed with the (a or (b options in the final frame and block counts.

n (The letter "n") Activates nopage function on output to the terminal.

o Displays the "old" safe table. Before every full or incremental save, the most significant blocks in the overflow table are saved in an alternate area. This area can be displayed using the (o option. This option is useful for analyzing the change in overflow over a time period.

p Directs output to the printer, via the Spooler.

s Displays only "safe" blocks. The "safe" blocks are blocks that will be restored in case of a power-loss or full system crash.

t Displays blocks in the "safe" table with an asterisk.

Example

```
overflow
overflow: 8450 reserve: 1024 blocks: 669 b-tree levels: 2 leaves: 9
This indicates that there are 8450 frames remaining in the overflow table,
1024 marked as the "spare gas tank"; a total of 669 blocks in the table; 2 b-
tree levels with 9 leaves.
overflow (a319843-319934
319838-319844 : 7  319849-319855 : 7  319866-319866 : 1
319885-319885 : 1  319889-319889 : 1  319898-319899 : 2
319902-319902 : 1  319907-319909 : 3  319924-319924 : 1
```

```
319927-319927 : 1  319929-319929 : 1  319932-319932 : 1
319934-319934 : 1
overflow: 8497 reserve: 1024 blocks: 692 b-tree levels: 2 leaves: 9
```

ovf

Invokes "overflow" command.

p

toggles on or off the display of terminal output.

When activated, the output from any process will not appear on the screen.

Example

```
p<return>
who<return>
(no output appears)
p<return>
who<return>
40 bob ref
```

password

allows adding, changing or removing a user or account password.

This must be issued from the "dm" account.

The password is encrypted into a hexadecimal string.

The command prompts for the following:

Change USER or ACCOUNT password (U,<A)>?

User passwords are found in the "dm,users," file. Enter the appropriate user id.

Account passwords are found in the "mds" file, where the item-id is the account name. The password is encrypted in attribute 7. Enter the account name.

If the password for "dm" is being changed and a password is currently defined, the following prompt appears:

Enter old password (you will not see what you type):

The "old" password must be entered before it allows changing or removing the password. If the old password is unknown, the easiest way to change it is to use the Update processor directly on the account name in the "mds" file (u mds account.name password).

Enter new password (you will not see what you type):

Enter the password here. Pressing <return> at this prompt indicates that the password is to be removed.

Re-enter new password for verification (you will not see what you type):

If a password is being added or changed, the "password" command requires that the same password be entered again to verify that it was entered correctly.

If a password is being removed, the following prompt appears:

Confirm that you wish to remove the account password (y/n)

Answer the question accordingly.

penv

displays the current D3 shell variables.

This command displays a list of all user D3 shell variables and the contents of those variables.

Syntax

penv

phantom-reset

Resets the specified phantom lines to 'available' status. No checking is done to ensure that the phantom line is actually available, so this verb must be used with extreme caution only after the phantom line is known to be available.

Syntax

phantom-reset a,b,n-m

Phantom lines can be specified either one at a time (as a,b above), or in a range (as n-m above), or any combination thereof.

phantom-status

Displays all phantom lines and whether they are "busy", "available", or "Transaction logger".

The scheduler's release table is also displayed. This is any workspace not yet released, but would be if the scheduler is running.

pibstat

displays the current pib status bit settings for a given value.

The "value" is the two-byte pib status returned by the "where" command, under the column labeled "PIB Stat".

The PIB Stat is perhaps the most useful indication of resource usage on a D3 machine. The first 2 digits of this value indicate the "state" of a process. Here are some of the more common values:

F3xx Waiting-for-input or displaying text

E3xx Waiting-for-input or displaying text with echo off

F5xx Displaying text, but line is in XOFF (Ctrl-S) state

FFxx Process is currently using the CPU

BFxx Process is currently sleeping

DFxx Process id currently requesting disk I/O

Syntax

pibstat value

Example

```
:where 68
 Ln  PCB    PIB   ABS   Stat  ...
     FID    Stat  Bas
*068 024EE5 FF10 000018    1  ...
This returns the pib status for port 68 as "ff10", which can then be used in
the "pibstat" command:
:pibstat ff10
1111 1111 0001 0000
```

```
Process is running or activatable
           not sleeping
           not frame faulted
           echoing input
           not I/O roadblocked
           not reading
           not writing
           not comatized
           not DCD on
           type-ahead enabled
           modem-control enabled.
```

pick

see pick (Unix)

pick-setup

sets administrative guidelines for the system, including:

- selection of ABS frames
- date format
- monitor keyboard type (for port 0)
- system time and date
- system shutdown delay
- restoring and updating accounts
- creating new accounts
- assigning account passwords

pid

displays the Unix process-id (pid) of a D3 process.

The D3 process process is specified by a "port.number" (pib), all D3 processes currently connected if the argument is "*", or the current process if there is no argument.

The first form displays the Unix process-id, (PID) or the specified D3 "port.number" (pib).

The second form displays the Unix process-ids of all the D3 processes currently connected to the virtual machine. An asterisk ('*') precedes the current process. If a list of PIDs 'pid.to.search' is provided, a plus sign ('+') is displayed before each process found in the list. This form is useful to determine whether a given Unix process, identified by its PID, is connected to the virtual machine.

Syntax

pid {port.number} {(option}

pid * {pid.to.search} {(option}

Options

p Directs output to system printer, via spooler.

Example

```
1)  :pid
34726
"34726" is the current Unix pid.
2)  :pid 22
14525
```

```
"14525" is the Unix pid for D3 pib "22".
3)  :pid * 36978 2345
PIB     PID       PIB    PID
===    ======     ===   =====
0      35002    * 2     6513
+ 3    36978      4     42611
Displays the PIDs of all D3 processes and searches to determine if the Unix
processes '36978' and '2345' are D3 processes. This command was executed on
(D3) line '2'. The Unix process '36978' is connected to the virtual machine on
line '3', and the process '2345' is not connected to this virtual machine.
```

pitch-compile

compiles a pitch file found in the "dm,fonts,pitch" file.

Syntax

pitch-compile fonts,pitch {item-id} {(options}

Options

? Provides brief on-line usage information.

c Writes width in decipoints.

o Overwrites existing item(s).

s Suppresses crt output.

t Writes width in dots. (default).

w Displays as it happens.

x Writes width in hex scaled decipoints.

y Displays statistics when done with item.

pitch-table

builds a pitch table for a font file in the "dm,fonts,descr" file.

The item generated is followed by the extension, ".pitch".

Syntax

pitch-table fonts,desc {item-id} {(options}

Options

? Provides brief on-line usage information.

c Writes width in decipoints.

o Overwrites existing item(s).

s Suppresses crt output.

t Writes width in dots. (default).

w Displays as it happens.

x Writes width in hex scaled decipoints.

y Displays statistics when done with item.

poke

sends either a string of characters or one or more TCL commands to the input buffer of another port.

The "port.number" designates the target port number.

The "text" may contain any characters, except segment marks (x'ff') and may be unlimited in length. The text may optionally be preceded by a space. When the text is omitted, a <return> is sent to the target port. The "text" string may optionally be enclosed in single quotes ('), double quotes (") or backslashes (\).

One or more TCL commands may be sent directly to the target port. Each TCL command must be followed by an <escape>, which is treated as a <return> on the receiving port.

An item containing one or more valid TCL commands may also be transmitted directly from a file. The transmitted item must contain one valid TCL command on each attribute. This feature is accomplished by simply poking an <escape> to the target port. The process prompts for the filename and item-id of the item to transmit and execute.

The "poke" command displays the number of characters it has successfully poked to the target port's input buffer. The number reported may be not match the actual number of characters sent due to the condition of the target port's input buffer.

Since the <escape> key is used by "poke" to terminate commands, a special provision applies for transmitting an actual <escape>. This is accomplished by pressing <escape> twice. A set of two <escape>'s is treated (and poked) as one <escape>.

Syntax

poke port.number{,{ text}}

poke port.number{ }<escape> {file.reference {item-id}}

poke port.number<escape>

poke port.number TCL.command<escape> {TCL.command{<escape>}...}

Example

```
poke 16
[1028] 1 characters poked.
Sends a <return> to port 16.
poke 16,
[1028] 1 characters poked.
This also sends a <return> to port 16.
poke 16,answer the phone!
[1028] 17 characters poked.
Sends the string, "answer the phone!" to port 16, without following the
(poked) string with a <return>.
poke 16,"answer the phone!"
[1028] 17 characters poked.
This is exactly the same as the previous example. Quotes (single or double)
and backslashes have no effect on the string being sent, unless the string
happens to begin with, but not be followed by, a literal quote. See the next
example.
poke 16 "hi
[2] Uneven number of delimiters (' " \).
This fails because the string is preceded by a quote. The next example shows
that a trailing quote is accepted.
poke 16 hi"
[1028] 3 characters poked.
poke 16,don't panic!
[1028] 12 characters poked.
Embedded quotes are allowed.
In all of the previous examples, no <return> was sent at the end of the
"poked" strings.
poke 16 who[
In this form of a "poke", the string being sent is an actual TCL command. The
```

```
"[" character is the character echoed by the
<escape> key. This pokes a "who" to port 16 and issues a
<return> to process the command.
poke 16 who[time[ovf
This pokes the "who", "time" and "ovf" commands to port 16 and executes them.
poke 16[<return>
poke from file-name item-name:md script1
This form uses the port number, followed by an <escape>, then a <return>. The
process then prompts for the filename and item-id of the item to transmit.
Each attribute of the specified item, starting at the first attribute, is
treated as a valid TCL command. For example, "script1" could look like this:
   id:   script1
att 1:   who
att 2:   time
att 3:   ovf
poke 16[md<return>
poke from item-name:script1
This is the same as example 10, but the filename is supplied on the command
line. The process only prompts for the item-id of the item to transmit.
poke 16[md script1<return>
[1028] 13 characters poked.
This is the same as example 11, but the filename and item-id are specified on
the command line.
poke 16[
poke from file-name item-name:[
0001  :who<return>
0002  :time<return>
0003  :ovf<return>
0004  <return>
[1028] 13 characters poked.
In this form, an item is constructed on-the-fly, then poked into the target
port's input buffer. This example executes the "who", "time" and "ovf"
commands on port 16.
poke 16 [[
[1028] 1 characters poked.
Poking two <escape>'s results in one <escape> being sent.
execute "poke 16 " : char(24):char(5):"y"
This illustrates using "execute" to "poke" characters to another port. This
instruction pokes a <ctrl>+x, followed by <ctrl>+e, followed by a literal "y".
This would explicitly exit the receiving port from the current item in the
Update processor.
```

povf

displays the available "linked" and contiguous overflow space.

See "overflow".

Syntax

povf {(p)}

power-off

brings the system to an "orderly shutdown" by logging off all users, flushing memory to disk, and parking the disk heads prior to halting the system.

This is used prior to system shutdown to ensure that all work spaces are returned to overflow and all updated frames in memory are written to disk. "power-off" puts the machine in a HALT state.

The status of the Spooler is displayed and an option to continue is offered . If "y" is entered to continue, any user still logged on is logged off, all process work spaces are released, all printers are stopped, and all updated memory frames are written to disk. The system is shut down.

If the "u" option is not specified and any port, other than the port executing "power-off", is logged on, the system prompts for each port to be logged off.

If the port is not logged off by the time the "power-off" reaches its final phase, the port is automatically logged off.

Syntax

power-off {(u}

Options

u Unconditionally logs all ports off.

Example

```
power-off
Test the logon state of each line...
If any ports are still active, the following message appears:
Line 1 is logged on as lw ts
Do you want to log it off (y/n)?
If the response to the question is "n", the following message appears and the
process stops, and returns to TCL:
Shutdown procedure aborted.
If the response to the question is "y", it attempts to log the port off and
the following message appears:
[534] logoff successful.
This question is repeated for all ports which are still active. After all
ports are logged off, the following messages appear:
Detaching tape...
Spooler status:
  The spooler is inactive.
Do you wish to continue (y/n)?
If the response to the question is "n", the following message appears and the
process stops, and returns to TCL:
Shutdown procedure aborted.
If the response to the question is "y", the following messages appear:
Terminating all print jobs.
Wrapping up process on line 1.
Wrapping up process on line 2.
After all the lines are "wrapped up", the following message appears:
Pause for wrapup processing to complete...
Flushing memory, boot when disk is quiescent...
After the disk settles down, the following message appears:
Flush complete.
```

prime

calculates the next higher and lower prime number from a given number.

If no number is provided with the command, the program requests it.

Syntax

prime {number}

Example

```
prime 13
13 is a prime number
the next highest prime number is = 17
the next lowest prime number is = 11
prime 14
the next highest prime number is = 17
the next lowest prime number is = 13
prime<return>
PRIME - computes the next greater prime number
Enter a positive integer = 13<return>
```

```
13 is a prime number
the next highest prime number is = 17
the next lowest prime number is = 11
```

print-err

provides a means of displaying messages from the "messages" file in their "output" format.

Messages are generally kept in the "dm,messages," file, but the process may specify any file which contains items with a similar format to the items in "dm,messages,". See "messages, file, Definition" for more information on how error messages are constructed and interpreted.

"file.reference" is the name of the file, usually "messages", that contains the message list.

"msg.list" indicates one or more item-ids of items in the "messages" file.

Syntax

print-err file.reference msg.list {(options)}

Options

n No pause. Suppresses pause at end of page on terminal display.

p Directs output to printer, via the Spooler.

s Used in FlashBASIC to allow stacked input.

Example

```
print-err messages 3
[3] The verb 'A' is not defined.
This looks like it did NOT work. It actually did. Message "3" prints when an
invalid verb is issued. The 'A' is where the invalid verb would appear.
print-err messages 333
[333] The file of files cannot be cleared or deleted.
print-err messages 659 536
[659] Line printer ready.
[536] already logged off
data "stuff"
execute "print-err dm,messages, 201"
This FlashBASIC code produces the output:
[201] 'stuff' is not a file name.
```

print-error

prints an error message from stacked data statements.

The "print-error" command is like the "print-err" command, with the difference that this verb, when executed from FlashBASIC allows "stacked" data statements.

Syntax

execute "print-error"

Example

```
data "100"
data "black"
data "white"
execute "print-error"
This program displays:
[100] 'black ' is not 'white'.
```

print-filter

controls the ways attribute and value marks are displayed or printed.

If the "print-filter" command is issued without an option, the current setting status is returned.

In conjunction with the "crt-delimiters" command, the "print-filter" allows attribute and value marks to be displayed or printed in three different ways, according to the following table:

crt- print- AM/VM are displayed as:

delimiters filter PRINT | CRT statements

(f (f unchanged (x'FE')

(n (f unchanged (x'FD')

(f (n '^',']' | CR/LF

(n (n '^' and ']'

The "print-filter" command takes precedence over the "crt-delimiters" command. As a result, when the "print-filter" is set to OFF, the "crt-delimiters" has no effect.

At boot time, the "print-filter" defaults to ON.

Syntax

print-filter {(options}

Options

n Makes attribute and value marks printable.

f Leaves attribute and value marks unchanged.

printronix

sets the "vfu" (vertical formlength unit) for printronix printers only.

The vfu setting include a page length, skip perfs, and vertical tabs.

prompt

prompts the user with a given text string.

The user is then given the option to either continue (by entering the letter "c", or stopping the process (by entering the letter "q"). No other characters are allowed.

The "prompt.text" has the same conventions and syntax as the "display" command. See "display" for more information.

Syntax

prompt {prompt.text}

Example

```
mactest
001 n
002 display Starting file save
003 prompt Insert first reel
004 save (fts
In this example, after displaying "Starting file save", the user is prompted
with:
Insert first reel -- Quit/Continue (q/c) ?
If a "c" is provided, the "save" command is executed. The macro stops
immmediately if a "q" is provided. Only "c" or "q" are accepted by "prompt".
```

psh

executes a Unix command and captures the results in a D3 item.

This command is an alternative to the "sh" verb, which allows capturing the result of the "Unix.command" into a D3 item. The "t" option displays the result on screen, as well as storing it in the specified item.

"Unix.command"

Any Unix command. If it contains a ">" ("greater than" sign), it is passed down to "sh" with its conventional usage. In this case, the ">" before the D3 item specification must be "escaped" by a tilde (~). Only stdout is captured. To capture stderr, redirect it to stdout (see examples).

"file"

Optional D3 file name where the item is stored. If not specified, "lst" is used as a default.

"item"

Item name where the result is stored. If two greater-than signs (>>) are used, with no space in between, the result of the Unix command is appended to the end of the item, if it already exists.

Syntax

psh Unix.command {~}>{>} {file} item {(t}

Example

```
psh cal 1992 > tmp 1992
Prints a calendar for the year 1992 and stores it in the item "1992" in the
file "tmp".
psh cat /usr/pick/myfile > myfile
Transfers the file "/usr/pick/myfile" into the D3 item "myfile" in the file
"lst" (default). If the file does not exist on Unix, the error message is
displayed on the terminal.
psh cat /usr/pick/myfile 2>&1 ~> myfile
Transfers the file "/usr/pick/myfile" into the D3 item "myfile" in the file
"lst" (default). "stderr" is redirected to "stdout" ('2>&1'). Therefore, if
the file does not exist, the error message will be in the D3 item. Note the
"escape" of the >Pick redirection.
psh cc myprog1.c 2>&1 ~> myerrs
psh cc myprog2.c 2>&1 ~>> myerrs
Stores the error messages produced by the two compilations of "myprog1.c" and
"myprog2.c" in the item "myerrs". The second error messages are appended to
the end of the first. Note the "escape" of the ">Pick" redirection and the
">>" to append data.
psh cd /tmp; exec ls -l > mylist
Changes directory, lists it and stores the result in the item "mylist". Note
the use of "exec" to avoid creating an intermediate shell. Note also that "cd"
is not a Unix command, but a "sh" keyword, and thus cannot be "exec'ed".
```

psr

displays formatted output of the status of the currently active Unix and/or D3 processes.

Whenever possible, this command tries to identify the D3 process from the status returned by the Unix command "ps".

If "port.number" is specified, the output shows process information related to the D3 process port and its child processes. This argument supersedes the "r" and "k" options.

This command displays the following:

"PID" is the Unix Process-id.

"PPID" is the Unix process id of the parent process.

"TTY" is the device name where the process is connected.

"S" is the Unix process status:

R : running

S : suspended

T : terminated

W : Waiting for IO

Z : zombie (defunct).

"PORT.NUMBER" is the D3 port number.

"USER" If the process is connected to the current virtual machine, this is its user-id.

"ACCOUNT" If the process is connected to the current virtual machine, this is its master dictionary.

"STAT" If the process is connected to the current virtual machine, this is its D3 status (See the "where" verb).

"MODE/COMMAND" If the process is connected to the current virtual machine, this is the name of the virtual mode it is currently executing. If not, it is the name of the Unix command it

is executing. Note that if this command is "ap", this probably means that the process is connected to another virtual machine.

The form, "psr port.number", is useful to identify all processes spawned by a D3 process. For example, if a process pushes a level and executes another "ap" command, it is not easy to find the relationship between the two D3 processes (one is actually the grand child of the other).

The form "psr (r" allows detecting processes which are consuming CPU. By repeating this command after a few second intervals, it is easy to identify processes looping or having hardware related problems (too many serial interrupts, for instance).

The form 'psr (sr' is an alternative to 'psr (r' to show running processes. It allows taking a sample on a longer period, thus catching processes running often, but not for a long time.

Syntax

psr {port.number} {(options}

Options

k Shows only the processes running in D3.

p Directs output to system printer, via the Spooler.

r Shows only the Running processes.

s{n} Select running processes by a doing two 'ps' commands, separated by a specified sampling period 'n' (default 1s), and comparing the CPU usage of the processes.

Example

```
psr
PID  PPID TTY      CPU S PORT USER ACCOUNT  STAT MODE/COMMAND
1       0 -      18:08 S                       init
1973    1 -       1:04 S                       syncd
3071    1 10/0  421:23 R                       ap
3255    1 -       0:01 S                       errdemon
3762    1 112/0   0:00 S 072               F310 md0:00C
4270    1 108/0   0:00 S 068               F310 md0:00C
4368    1 -       0:00 S                       srcmstr
5028 4368 -       0:00 S                       writesrv
5283    1 89/0    0:00 S 057  dm   dm      F310 tcl.input:000
5615    1 102/0   0:20 S                       pxproute
6053    1 91/0    0:15 S 059               F310 rdl.getchar:000
6314 4368 -       0:00 S                       qdaemon
6513    1 2/0     0:00 S 002               F310 rdl.getchar:000
7110    1 0       0:00 S 128               BF10 sp.sleep:040
psr (d3
PID  PPID TTY      CPU S PORT USER ACCOUNT  STAT MODE/COMMAND
3071    1 10/0      +3 R                       ap
 This command takes two samples at 3 second interval and show the time
accumulated by the running process. In this example, the process 'ap' has
consumed 3 seconds (the whole sampling period), which is an indication of a
potential problem.
```

pverify

verifies the checksum for the object code of a FlashBASIC program.

Each time a program is compiled, a checksum is generated and stored with the object code. "pverify" recalculates the checksum, then verifies it against the stored checksum.

Syntax

pverify file.reference {itemlist*} {(options}

Options

e Suppresses the message [429] program object verifies.

p Directs output to the Spooler.

qselect

creates a list from the specified "file.reference" and itemlist*, similar to the "get-list" command.

The list is constructed from the attributes specified by "attr.ref" (attribute name or number) in an item (or list of items) specified by "itemlist*". Each attribute, value and subvalue becomes a single attribute value in the list.

If no "attr.ref" is specified, all attributes are used.

Syntax

qselect file.reference {itemlist*} {(attr.ref)}

Example

```
qselect invoice-file s12345 (5
This builds a list from the multi-values stored in attribute 5 of the item-id
"s12345" from the "invoice-file".
qselect dict ap.doc 'desc'
This constructs a list out of the "desc" attribute-defining item in the
dictionary of the "ap.doc" file.
```

reblock-ovf

forces contiguous overflow blocks which exist in different internal tables to be compared and combined if possible.

In releases 6.1.0 and above, the overflow is split into a "safe" table which contains the largest available blocks, and the normal "b-tree" table which contains smaller blocks. After a crash, the "safe" table is used to recover the overflow to a known state. Because of this separation, it is sometimes possible to have blocks which appear to be numerically contiguous, but which are not combined because they exist in different tables. The "reblock-ovf" verb will correct this situation.

reboot

reloads the operating system from a currently executing system.

"reboot" can only be executed from line 0. All users must be logged off before "reboot" can be invoked. If any users are logged on, a message is printed and control returns to TCL.

rebuild-ovf

reclaims overflow space by restoring all the "signatured" frames to the overflow table.

This only works if the overflow table has been previously initialized ("scrubbed") with the "init-ovf" command.

reclaim-ovf

allows the user to recover all frames not used by the system to the overflow space, thus eliminating the necessity of a file-save/restore.

The process will be initiated with the verb "start-reclaim-ovf" after system has been shutdown properly and the new overflow table initialized.

Syntax

reclaim-ovf {(options}

Options

b initiates the overflow reclamation process in the background.

f displays the file names as they are scanned.

i reuses the same pre-allocated workspace.

k kills the overflow reclamation process

n user specifies the number of pre-allocated frames (without n-option, 100 frames will be pre-allocated as default).

r resumes a previously suspended RECLAIM-OVF.

s inquires status of the overflow reclamation process.

u unconditionally kills the RECLAIM-OVF process.

x suspend the currently active RECLAIM-OVF process.

recover-fd

recovers an item previously deleted with an (editor) "fd" command.

This must be executed immediately after the "fd" command, otherwise the item is lost. If a list of items is active, there is no chance of recovering the item. This does not work on a list deleted through the "edit-list" command.

Example

```
ed bp print.invoices.bak
top
.fd
'print.invoices.bak' deleted.
recover-fd
recover "bp print.invoices.bak" (y/n)? y<return>
[253] 'print.invoices.bak' recovered.
```

recover-item

recovers the last item deleted by the "<ctrl>+xo" command in the Update processor. It also recovers the last item that was exited without filing.

The process first displays the name of the item to be recovered.

The deleted item is stored in a temporary buffer and can be recovered with this command until another item is deleted.

Example

```
:u test item1
<ctrl>+xo
'item1' deleted.
:recover-item<return>
Recover test item1 (Y/N)? y
[253] 'item1' recovered.
```

rem

used in a macro, designates a "remark" statement and all text which follows on the same line is ignored.

The "remark" statement is used to explain or document the macro. It is provided for compatibility and readability.

Syntax

rem text

remote-cache

increases the performance of updates over a network by sending groups of updates at a time.

Normally, each update made on a remote file is sent immediately to the remote site. Because of limitations in network throughput, this can introduce performance delays in batch update applications. To alleviate this issue, batch updates can be cached and sent in groups. This significantly improves performance almost to the same point as a series of local updates.

To turn on update-caching and/or to flush the cache, do "remote-cache file.reference"

To turn off update-caching and to flush the cache, do "remote-cache file.reference 0"

To set the number of updates to be cached before being sent, do "remote-cache file.reference number.of.updates". The higher the number.of.updates, the better performance will be.

Syntax

remote-cache file.reference

remote-cache file.reference 0

remote-cache file.reference {number.of.updates}

remote-errors

See "net-errors"

rename-file

changes the name of a file, both in the "md" and in the dictionary of the designated file.reference.

The process prompts for the new name. Items within the file and its associated dictionary are not affected.

Syntax

rename-file file.reference

NEW NAME: new.file.reference

Example

```
rename-file test1
NEW NAME:test2
```

renumber

equivalent to running the "bformat" verb with the "R" option and a control table specifying no indentation.

repeat

restarts paragraph execution at the line following the most recent "loop" statement.

Note that paragraph will correctly handle a loop structure within another loop structure.

Syntax

repeat

Example

```
paragraph
loop
time
if <<a,Quit (Y/N)?>> = y then go done
repeat
done:
 This example will repeatedly print the time and ask the user if he/she wishes
to quit.
```

reset-port

restarts suspended output and drains both the input and output terminal buffers.

No other characteristic of the device is altered. Restarting suspending output is useful when a port is stopped by a X-OFF and a X-ON was never sent.

"port.number" is the D3 port.number (pib) associated to the device to restart. This command may not be issued against its own port.

On D3 Unix implementations, the D3 process must be connected to the D3 virtual machine, otherwise the "ttyname" form of the command must be specified. "ttyname" is the device name to reset. This form can be used on any serial device on the system, even if not connected to the D3 virtual machine.

Raw Mode:

Normal D3 operations require that the device is used in 'raw' mode, i.e. where Unix does not do any pre- or post-processing to the data exchanged with the device. However, some Unix operations sometimes re-program the device and leave it in a state incompatible with D3. The most common symptom is a 'double echo' (i.e. a command is echoed one character at a time, as usual, but, when pressing <return>, the command is displayed again and nothing happens). To correct this situation, issue a "reset-port line (r)", which attempts to reset the device to a 'suitable' state.

Syntax

reset-port port.number {(options}

reset-port ttyname {(options}

Options

f Suspends output. This option acts as if the device had sent an X-OFF (i.e. <ctrl>+s). This option is specific to D3 Unix implementations.

i Drains input buffer. All characters waiting to be read from the device are removed. If no option is specified, this option is part of the default. This option is specific to D3 Unix implementations.

n Restarts output. This option acts as if the device had sent an X-ON (i.e. <ctrl>+q). If no option is specified, this option is part of the default. This option is specific to D3 Unix implementations.

o Drains output buffer. All characters waiting to be sent to the device are removed (lost). If no option is specified, this option is part of the default. This option is specific to D3 Unix implementations.

r Resets to 'raw' mode (see notes in "Raw Mode"). This option is specific to D3 Unix implementations.

Example

```
reset-port 25
reset-port /dev/tty26
```

reset-user

Reinitializes a process. It is also called by "logoff" in a worst-case scenario.

Old workspace is cut loose and not returned to overflow. New workspace is retrieved from overflow and the process is sent to logon. Phantom processes are released to overflow.

Logoff has the ability to time out and escalate its duties to perform a reset-user if necessary. This command replaces the logoff (z functionality.

The new initialized workspace receives the abs that the process performing the reset-user has.

resize

resizes a file to the desired modulo.

The "resize" command increases or decreases the apparent contiguous portion (or modulo) of the specified file without requiring a file-restore. It either adds or releases the amount of overflow necessary to reach the new modulo, and re-hashes all of the items. The re-hashing is done by groups and the current relative group being re-hashed is displayed on the screen.

The "resize" command allows a file to be read or modified while the items are being re-hashed. Furthermore, the resizing process does not require a completely new block of overflow with a size equal to the new modulo. Instead, it allocates a new file "segment" which is only as big as the difference between the old and new modulos and maps this segment onto the existing file. The file then appears to have one contiguous block available even though it may really be several blocks internally.

Once a resizing command has begun re-hashing a file, the command can be logged-off or interrupted without problems. The resizing process can then be re-started on the same line or on a phantom.

If the modulo is not specified, then the "resize" command will resize the file according to the "reallocation" attribute in the file's D-pointer.

To see all resizing processes currently active, use the "list-resizing" command.

To temporarily stop all resizing processes on the system, use the "kill-resizing" command.

To restart all resizing processes (on phantoms), use the "check-resizing" command. This command is executed at coldstart time to restart any resizing commands which were interrupted by a shutdown.

After resizing a file, the D-pointer of that file will display a modulo equal to the new modulo. The "reallocation" attribute is also changed to the new modulo. The internal base and modulo of each of the file's segments are displayed in the "segment-base" and "segment-mod" attributes. These attributes cannot be modified.

Note that it is currently not possible to resize below the original modulo.

Syntax

resize file.reference {modulo} {(options}

Options

a Allocate new file space only. If this option is used, then the new segment is added, but no items are re-hashed. This is useful for allocating the new file space in the foreground, and then starting another resizing process as a phantom to complete the re-hashing process.

s Suppress output of the relative group counter during re-hashing.

u Unconditional resizing. Normally, resizing processes will pause temporarily when some other process is accessing the file in a sequential fashion (like the save, or an AQL or FlashBASIC select). This is because items will be in motion during the resizing and sequential processes may find an item twice. The "u" option disables this behavior so that resizing will proceed irrespective of any sequential processes. Also, the "u" option will release extra unused filespace at the end of a resize down irrespective of how many users have that file open.

w{n} Wait after every group. If no numeric parameter is specified, then this option will cause the resizing process to wait for approximately 100ms between each group that it re-hashes. If an optional numeric parameter is specified, then the process will sleep for n seconds between each group. This option minimizes the impact on overall system performance during the resizing process and is strongly recommended.

z Rehash a small number of groups only. This is used by the check-resizing and kill-resizing commands after a resizing process has been terminated unexpectedly. Under these conditions, the resize command will process enough groups to assure that no duplicate items occur in groups which were previously being re-hashed.

Example

```
Assume that a file called "mydata" exists with a modulo of 7, and that the
"istat" command indicates a suggested modulo of 13.
resize-file mydata 13
Allocating 6 additional frames for primary file space.
Rehashing 7 group(s).
7
[188] Resizing complete.
The file now exists with a modulo 13. Now, suppose that a large amount of data
is removed from this file, and "istat" now reports that the file should be a
modulo of 11. The following command will shrink the file:
resize-file mydata 11
Rehashing 13 group(s).
13
Releasing 2 frames back to overflow.
[188] Resizing complete.
```

resize-file

see resize

restore-accounts

restores all accounts which do not already exist on the system from the attached "file-save" tape or a tape created by an "account-save" with an active list of accounts.

This is generally used with the last good "file-save" after upgrading to a new release of D3. It is important to restore the D3 file system from the media on which D3 is provided, since changes may have been made.

Restore-accounts uses "account-restore (az" in its processing loop. The "az" options are described under account-restore.

Syntax

restore-accounts {(options}

Options

frame.size (integer number). Designates the frame size of the source system which created "save" media. This option automatically affects modulos, which are either divided or multiplied by the given number, according to the data frame size on the target machine (see the "what" command for determining data frame size). If the target frame size is bigger than the source frame size, all modulos are divided by the ratio (target / source = ratio). For example, if the target system has 2K data frames and the source system had 500-byte frames, the modulos would all be divided by 4. This file resizing operation occurs after the reallocation parameter has been checked, unless file resizing has been disabled during the initial boot process.

c Used with tapes created on Ultimate Systems. Everything behaves the same as usual, but it's just slower.

d Data-sensitive files. All files except the md will be restored as ds-type files (case sensitive).

n Disables file reallocation process.

u Indicates that tape label is in "Ultimate" format.

Example

```
restore-accounts
This restores all accounts from the current peripheral storage device.
:restore-accounts (cu
This restores all accounts on a tape created by an Ultimate system.
```

ri

Invokes "recover-item" command.

rmbi

see rmbi (Unix)

rnf

Invokes "rename-file" command.

run

invokes FlashBASIC runtime, which attempts to load and execute a compiled FlashBASIC program.

Programs may be created and modified using UP. They can be run when exiting UP by using the <ctrl>+xr command, which files the item, compiles it, and, if there are no errors, runs it.

Syntax

run file.reference item-id {(options)}

Options

a Prohibits entry into the FlashBASIC debugger; aborts on error conditions.

d Enters FlashBASIC debugger prior to execution. Important when parameters are passed in a "call" statement. The FlashBASIC debugger may also be called at any time while the program is executing by pressing the <break> key on the terminal

e Enters debugger on any error condition. This option forces the operator to either accept the error by using the debugger, or exit to TCL. The only valid responses in the debugger are "end" or "off". "g" will NOT work.

i Inhibits variable initialization. This option is not recommended due to reliability, transportability, and data integrity reasons. Causes the values in the variable area to be retained when the program is run. This can be specified only when "run" is being executed from the "chain" command. It allows variables to be passed from one program to the next, via the "common" statement.

n Activates nopage function, on output to terminal.

p Directs output from "print" statements to Spooler. (Has same effect as issuing a FlashBASIC "printer on" statement)

s Suppresses run-time warning messages.

run-list

runs a series of FlashBASIC programs.

As each one concludes the next program is automatically executed.

An "active list" of FlashBASIC programs must be present before invoking this command.

Syntax

run-list file.reference

Example

```
select bp program1 program2 program3
[404] 3 items selected out of 127 items.
run-list bp
```

Runoff

facilitates the preparation and maintenance of textual material such as memos, manuals, etc.

Syntax

runoff file.reference itemlist* {(options)}

Options

* see "options: Runoff".

save

invokes the backup procedure to save the entire file system, an individual account, or to perform an incremental save. It is the process invoked by the "account-save" and "file-save" processes.

The appropriate peripheral storage device must be attached to the current process before invoking this command. See the list of "set" commands.

The following types of saves may be performed:

A "full file-save" is a complete logical backup of the D^3 file system.

An "incremental file-save" saves only those items which have changed since the last full file-save. Changed items since the last save are referred to as "dirty" items. If multiple incremental file-saves are done between full file-saves, the most recent "incremental" file-save contains the accumulated changes since the last full file-save. See the "u" option.

A "list driven file-save" saves all items for a given file that are specified in the "dm,file-of-files," starting at attribute 25 of the item representing that file. Attribute 25 begins the "select list". See the "l" and "m" options.

An "account-save" saves all the files in a single account

An "incremental account-save" is like an "incremental file-save", but only affects the files in a single account.

A "dummy" or "fake" save is a save without the "t" (tape) option. This is useful in cases where the D^3 file system needs to be verified for integrity.

"save" may also be driven from an "active" list, like any other TCL2 verb, although it is truly not a TCL2 verb. If a list is active when save is invoked, each item in the list is treated as an account name. On a "full" file-save with a list active, a fully restorable tape is created. This automatically includes the "dm" account, even if it's not in the list. On an account-save with a list active, the prompt for account name does not appear. This type of save is not "fully restorable", unless the "dm" account is in the list. This type of tape may be used with "restore-accounts". In both of these types of saves, the "dirty" bits are not cleared unless the "v" option is used. There is a potential conflict with this function and the "l" and "m" options. If these options are used and a list is active, the active list replaces the list in the "dm,file-of-files,".

In any type of save process, the "d-pointer" to the file determines the course of action for saving or not saving files. The "d-pointer" type is attribute 1 (one) of its corresponding file-defining item. Files are saved according to the following rules:

"d", "dc", "dl", and "dp" files are always saved.

"dx" files are not saved. Any files "beneath" a "dx" file are also not saved. This can be overridden using the "x" option.

"dy" file pointers are saved, but NOT the data within them. After a restore, a "dy" file will be empty. This can be overridden using the "y" option.

The system does not save frames which are in the overflow table (unused frames), so the amount of backup media required is related to the number of frames currently in use.

When saving to multiple reels, the system prompts with:

Load volume #n and type 'C'

label 08:00:00 16 Jan 1997 DATA "account.name" # -

The prompt accepts either the letter "c", which tells it to continue after inserting the next sequential reel, or the letter "q", to quit and stop the save.

"Dirty Bits" - The item deletion scheme:

In order for the "incremental save" to work properly, the system automatically "marks" (or "dirties") any item that is changed in any way. This mark is normally cleared at the end of a complete save, and displays the message "clearing dirty bits" as it does so, unless the "v" option is specified.

If a file has a "dx" or "dy" code and an item in it is updated, that item is not marked as changed when it is filed. Therefore, even if the "x" or "y" is removed before the save, that item is not saved by an incremental file-save.

The save processor locks groups as it saves them. While the group is locked, no process may access any item in that group. Locking groups prevents spurious Group Format Error messages that would occur if another process changed an item while it was being saved.

Up to four groups may be locked at one time. These groups are the ones containing: the mds dictionary pointer, the md pointer, the file dictionary pointer, and the group for the data currently being saved.

Any changes made to the file system during the save (either a change or a deletion) are NOT written to the save.

If the save is being done to streaming cartridge tape (SCT), an explicit end-of-data sequence must be written, since SCT cannot back up. Both the "t-weof" TCL command and the FlashBASIC "weof" statement write the second eod needed to indicate the end-of-file. Without the eod, "account-restore" or "sel-restore" fail to see the end of the tape.

At the end of the save, the system displays the message "clearing dirty bits...", and a series of frame numbers display, showing the groups whose "marks" are being cleared. During this time, several files are reopened and updated, renamed and cleared. "Deleted" flags are cleared, and "save#" fields are reset.

Deleting Items:

The scheme by which D3 removes items from a file are as follows:

1) If the file is a "dx" or "dy" type, the group adjusts immediately to compensate for the deleted item.

2) If the file is NOT a "dx" or "dy" type, the item is deleted and replaced with a "deleted-item/item", which includes the item-id and the "delete" flag. These items are later deleted during the "clearing dirty bits" phase of a full save.

3) If the group has an update lock, the item delete bit is set and the last user to release the group lock compresses the group according to the previous two rules.

The reason that this was implemented has to do with incremental saves, which have to know that the item no longer exists.

The following scenario illustrates why this is handled this way.

1) Suppose there is an item called "fred" in the file "flintstones", and it is saved on a full file-save.

2) After the save, "fred" is deleted.

3) An incremental save is performed.

4) A meteorite strikes the system, necessitating a full restore (this, of course, is STRICTLY hypothetical.)

5) A full restore is performed, which brings back "fred".

6) At the end of the full restore, the option to perform an incremental restore is offered and accepted. It is during this process that "fred" is deleted.

Syntax

save {(options)}

Options

a Creates saves for D3 systems prior to release 7.0. On release 7.0 and above, systems contain added information which cannot be understood by old releases. Such older releases may get "tape format error"'s when restoring from newer saves. The (A option supresses the added information (including all update stamps and binary item sizes).

b Saves the b-tree indexes. After a full-restore, this prevents having to regenerate indices.

c Creates R83-compatible file-save tapes, generating a file-save media that will restore on R83 machines. D3 files have dictionary codes that are not recognized by R83 and vice versa.

Accounts should be "updated" when restored. This option does not work with incremental saves, only file and account saves. "dl", "dp", and "ds" pointers are saved as "normal" "d-pointers". "dc" and "dy" pointers are saved as they are. Binary items are properly dumped as R83 "CC" pointer items. Items larger than 32K and not in "dc" files are skipped, with a message displayed. Binary code items not in "dc" files are skipped.

e Saves external "qs" pointers. "qs" pointers are special q pointers which point to an OSFI data source. Note that only the spooler and Unix OSFI drivers support the save functionality at this point.

f Displays the file names as they are saved. If omitted, only the mds file and individual master dictionary (account) names are listed.

g Displays gfe (group format error) when found, but skips the group and continues the save. Errors are logged in the "dm,errors," file in attributes 14, 15, and 16 of the error item. Attribute 14 is the save date, 15 is the primary group fid (from gfe.sr), and 16 is the "account > dict > data" display information. At the end of the save, a message displays indicating the total number of gfe's encountered, if at least one was found. If used with the "b" option, index corruptions are also skipped.

i (account-save) Individual master dictionary save. The system prompts for the account name, if it is not specified at TCL. Using the "i" option with an active list generates multiple account-saves. If a list is active and the "i" option is specified, the system generates a tape that may be restored with the "restore-accounts" command.

j Suppresses the dirty bit counter display.

k Reorganizes all "file-of-files" numbers sequentially on the tape. This is also useful when frame one is corrupted or some file numbers are negative. The incremental restore will abort with negative file numbers.

l Each file may have a list of specific item-ids within the file to save. This option saves only the list of item-ids provided in the associated item for the given file, located in the "dm,file-of-files,", beginning at attribute 25. Any number of item-ids may follow in attributes 26 and beyond. An "*" (asterisk) in attribute 25 saves all items in the file. If there is no list, only the file-defining item is saved, as if the file had a "dy" d/code. In addition, the "l" option does not "unmark" (clear "dirty" bits) items that have changed. These items will be saved on subsequent incremental file-saves or account-saves. This is similar to the "v" option for the full file-save.

m This is the same as the "l" option, but it additionally "unmarks", or "clears" the "dirty" bits.

p Directs console listing to the printer, via the Spooler.

q Changes the terminal display from hierarchical format to indented format.

r This is the same as the "l" and "m" options, except the list is also saved with the file as a separate item in "dm,file-of-files,".

s An item is stored in "dm,file-of-files," for each file saved. The "s" option generates additional statistics which may be reviewed with "list-file-stats".

t Outputs to the "tape" (magnetic media) which has been previously attached to the process. If the "t" option is not specified, "save" does all the processing, but nothing is written to the media. The system prompts for the file-save tape label which is written on the tape as part of the tape label. Omitting the "t" option is useful for regenerating system statistics and/or finding gfe's, and is the basis of "fake" or "dummy" saves. See "tape label, definition".

u Performs an "incremental" file-save or account-save. This saves all file items and data items that have changed since the last full file-save or account-save.

v Performs a "full" file-save, but does not "unmark" the items that have been changed. These items will continue to be saved on subsequent incremental file-saves. Unless the "i" or "u" options are specified, the system does a "full" file-save.

x Saves files with a "d-pointer" type of "dx". See "file-defining items" regarding "dx" pointers.

y Saves files with a "d-pointer" type of "dy". See "file-defining items" regarding "dy" pointers.

Example

```
to dm
set-sct
Tape attached block size: 16384
[1082] Tape device is attached to quarter inch Streaming Cartridge Tape
save (bft)
file-save tape label = daily-backup
1   35  > dm
1   36  > dm > newac
1   37  > dm > newac > newac
.
.
Load volume #2 and type 'C'
label  08:00:00  16 Jan 1997  DATA dm daily-backup   #
.
.
2  142  > lastaccount > lastfile > lastitem
Clearing dirty bits ...
```

save-list

writes the list of item-ids generated by a "select", "sselect", "qselect" or "get-list".

If no file is specified, the pointer-file is used.

"save-list" must be invoked immediately after the verb that created the list.

"file.reference" is the file in which to save the list. If no file.reference is given, the list is saved in the pointer-file.

"listname" is the name of the list-item. If there is already an item with the specified item-id, it automatically overwritten by the new list. If no listname is specified, the default name, "%user.name", is used.

Syntax

save-list {{file.reference} {listname}}

Example

```
sselect entity with area.code "714" by name
[404] 16782 items selected out of 272876 items.
save-list orange.county
List 'orange.county' in 'pointer-file' saved.
```

scrub-ovf

see init-ovf

search

searches a file for the existence of one or more strings of characters in any attribute and optionally creates and saves a list.

After invoking the command, the prompt "search for: " appears on the screen. Any of the following responses to this prompt are valid:

- A string of characters

- ?? (double question mark). This displays on-line help.

- <return> Begins the search, after at least one string has been entered.

The item-ids can be saved in a list when the "l" option is specified.

As the "search" progresses through the file, it displays all lines that match the given search string(s) by default.

Syntax

search file.reference {item.list*} {(options)}

Options

a Outputs in standard assembly listing format.

d Outputs entire item on any match.

f Outputs only the first line containing the string in an item.

i Suppresses the item-id output. Does not affect other output.

j Outputs only one "match" per item (overridden by "d" option).

l Prompts for list name and saves the resulting list under the given name. This does NOT show matches or leave an "active" list at the end of the process.

n Outputs the report without pausing at the bottom of each page.

o Displays item-ids with "matches" and does not create a list.

p Directs output to the system printer, via the Spooler. (overridden by "l" option).

s Suppresses line number output.

u Outputs two attributes after, and one line before, the matching line.

Example

```
search md (l
search for:vb<return>
search for:<return>
list name: vb.list<return>
logoff
sselect
td
...
[2000] 15 out of 1512 item(s) found.
This example searches for "vb" in the "md" and saves the list as
"vb.list" after completing.
```

search-file

searches a file for the existence of one or more strings of characters in any attribute and optionally creates and saves a list.

The file name and search strings are requested by the program. After entering the search string(s), a <return> at the next "search string" prompt begins the search.

The program requests a list name. If provided, the list will be saved in the "pointer-file". Otherwise, the item-id's of the items containing the search string(s) appear on the screen.

Syntax

search-file {file.reference} {(options}

Options

? Displays usage information.

a Displays every ("all") occurences of specified string(s).

search-system

searches every file in the D3 file system for the existence of one or more strings of characters in any attribute and optionally creates and saves a list. If an md name is provided, only that account is searched.

It scans each item in each file looking for a string match in any attribute. If a match is found, it displays the item-id, and the attribute containing the matching string. Any number of unique strings may be specified. The search is successful if any one string matches any portion of the attribute.

This process prompts with the following messages:

enter the output file name?

Enter the name of the file where the found items will be saved (it must already exist). As items are found, "control" items are added to the "output file", using sequentially-assigned numeric item-ids beginning with 1.

enter the search string?

Enter the string(s) without any enclosing delimiters. The system will repeatedly ask for additional "strings". A "?" (question mark) may be used for a "wildcard" character in the string. Up to 15 different strings can be specified.

The "?" (question mark) character may be used for a "wildcard" search character. To locate a string which contains a "?", enter as "??" (two question marks). After the last string has been entered, a <return> at the "...string?" prompt advances to the next prompt.

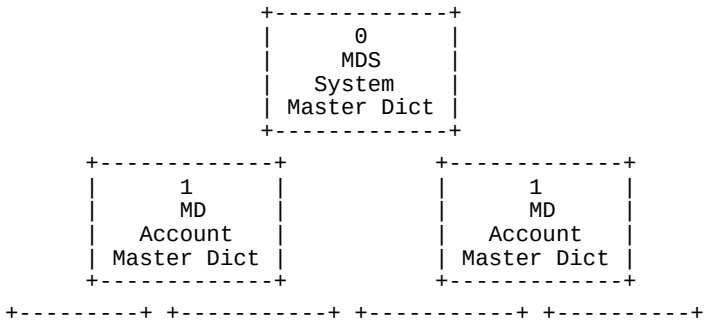
START LEVEL 0-Mds, 1-Master Dictionary, 2-File Dictionary, 3-Data File?

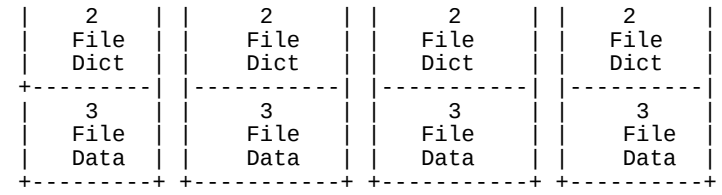
This tells the process at which "level" of the file system to begin the search. "search-system" has hierarchial level control. The four levels are:

MDS, MD, Dict File, Data File.

Any range can be specified. A data file search can also be specified to search system-wide horizontally

Enter the "level" to begin the search.





Next, the system asks either:

STOP LEVEL 0-Mds, 1-Master Dictionary, 2-File Dictionary, 3-Data File? or

LATERAL LEVEL 0-Mds, 1-Master Dictionary, 2-File Dictionary, 3-Data File?

This is the level to continue the search to. The routine can either traverse a single branch of the hierarchy (such as a single account and all its files), or it can traverse "laterally" across the branches (such as all the data files ONLY of all the accounts). The STOP/LATERAL level must be equal to or lower than the START level.

Output is an item of the format:

001 mds (account name)

002 dict, data file name

003 item-id

004 attribute number in which text was found

005 attribute text

006 the search string

Syntax

search-system {destination.file} {(options)}

Options

? Displays on-line usage information about the process.

a Displays all items searched, including items in which the string was not found.

c Clears output file prior to execution.

m Checks for multiple occurrences.

p Directs output to system printer, via the Spooler.

s Suppresses display on crt.

sel

see select

sel-restore

selectively restores items from either a "file-save", "account-save", "incremental save", or "transaction log" media to a specific file.

Normal backup procedures ensure all the items in all the files are saved from time to time. Critical files can be backed up continuously using the transaction logger.

The media used for this backup can be SCT, half-inch tape, floppies, etc. For convenience, the save media is referred to as "tape", or simply "media".

This verb can be invoked after the desired physical save media is put in the appropriate I/O device and attached with the appropriate command.

"destination.file.reference" indicates the file to restore into. The file must already exist.

"tape.file.reference" indicates the name of the file on the offline media.

The system prompts for the "file name" of the file to restore from the save media. If a null "file.reference" is entered, the account's master dictionary is restored.

"item.list*" designates which items to restore from tape. A null list restores all items.

The specified "account.name" and "file.reference" must match exactly with the appropriate name on tape.

To check the contents of the tape, the files can be listed by using a "sel-restore" and specifying a "dummy" account name and "file.reference" on the tape. (A "dummy" account/filename is one that does NOT exist in the system, such as "oxymoron"). The entire tape is searched looking for the "dummy" account, printing the names of the files on the tape.

In restoring both the dictionary and data section of a file, restore the data section first. Although the dictionary name appears before the data section name on the screen, the data is saved (and restored) before the dictionary.

At any point, the media may be moved backward ("t-bck (n)"), or forward ("t-fwd (n)") to position it (assuming that the "save" media device supports the command), and a "sel-restore" with the "a" or "n" option may be started. This may be faster than restarting the tape from the beginning when restoring both the dictionary and the data sections of a file, or when restoring multiple files. Unfortunately, however, SCT's do NOT allow "t-bck"; the only recourse is rewinding ("t-rew") and moving forward to the correct file ("t-fwd").

Account dictionaries (master dictionary items) follow all other files for each md on the tape.

If "s" is specified when the save tape is created, the contents of the tape can be seen using "list-file-stats" anytime before another file save using the "s" option.

Syntax

sel-restore destination.file.reference {item.list*} {(options)}

Account name on tape:? account.name

File Name:? tape.file.reference

Options

a Indicates that the media is already positioned at the correct account. The "a" option is used to avoid rewinding and searching. If the order of the file on the tape is known (from the "list-file-stats" report normally generated following the save) then multiple files may be restored without rewinding the media. After the first file is restored on the account, the next "sel-restore" is done using the "a" option. The "sel-restore" process prompts directly for file.reference. The system next prompts for "full, incremental, or transaction restore". As the tape is searched, the file names on it are printed, along with the file numbers. Names are indented one space for account names, two spaces for dictionaries, and three for data file names. At the completion of the restore, the system again prompts for incremental and transaction restore.

c Causes every item in current file to be a candidate for restore. This ensures that data can be restored even if a "d-pointer" is damaged on the tape. This option is used with the "n" option.

f Displays all file.references for all accounts. This is not compatible with the "n" option.

i Item-ids of the restored items are not to be printed.

n Restores file by its number. The process prompts for the number of the file to restore. This option is particularly useful for restoring from a multi-reel backup media, so that the restore can begin from the reel where the file resides (or begins). The file number can be found on the "file statistics" report for the appropriate save media. See "list-file-stats".

o Overwrites, if a duplicate item-id is found on disk. "d-pointers" are protected from being overwritten.

s Suppresses the "'item-id' exists on file" message.

Example

```
sel-restore dict devices
Restore from (F)ull account, (I)ncremental, (T)ransaction log: _ f
Account name on tape: dm
File name: dict devices
  :
  Scans the file-save tape, and finds the 'dm' account
  Scans the DM account and finds the 'dict devices' file
  Scans the DEVICES file and restores the requested item(s)
  :
Restore from Incremental save tape? _ no
Restore from transaction log? _ no
```

send-message

directs a text message to a specific port, a specific user-id, all current users, or every device attached to the system.

There are four valid verbs for sending messages: "send-message", "sm", "message" and "msg". They all behave exactly the same.

The "user-id" form sends the message to any port where the given user-id is logged on.

The "*" (asterisk) form sends the message to all users currently logged on to the system.

The "!port.number" form sends the message to the specified port.

The "!*" form sends the message to every terminal (and serial printer!) connected to the system, whether or not the device is logged on. (Even terminals sitting at the LOGON screen will receive a message.

The recipient(s) of the message see the time/date, the user-id and port.number of the user who sent the message, and the text of the message. The text need not be enclosed in quotes.

If the recipient(s) of the message is in the Update processor or in a menu, they will be prompted to "Press <return> to continue" before the screen is redisplayed. This message times out in 60 seconds.

If the recipient(s) of the message is shelled out to Unix, the message will be written directly to the approprate tty device.

The @(x,y) and @(-x) functions may be used within message text. For example "msg !2 @(-1) @(40,20). This message will clear the screen, then display this message at cursor position 40,20." When using the @ function, the beep and the 5-second delay are disabled on the process receiving the message. To enable these functions, see 'options'.

Options

b force beep on receiving process

d force delay on receiving process

Syntax

send-message user-id text

send-message * text

send-message !port.number text

send-message !* text

Example

```
send-message !0 are you there?<return>
```

Sends the message to port zero only.

```
send-message !5 @(-1) @(40,10) Here is some text
```

Sends the message to port 5. The message will first clear the screen, then print "Here is some text" at cursor location (40,10).

set

modifies the value of a D3 user shell variable.

If the "set" command is executed with no arguments, then it functions identically to the "penv" command. This displays a list of the currently set user variables and their values.

If the "set" command is executed with a variable.name, but with no variable.value, then the variable is assigned a null value. To completely remove a variable, use the "unset" command.

It is possible to create variable values which contain other variable names for later expansion. To do this, enclose the entire variable.value within single quotes. The quotes prevent the variable names from being translated immediately by the set command.

See the tcl "@" token for more information on using D3 shell variables.

Syntax

set {variable.name{={"|'}variable.value{"|'}}}

Example

```
set mycv=1
set
mycv=1
display @$mycv
1
```

set-8mm

see set-8mm (Tape)

set-abs

designates a file as the data section associated to the boot abs, or control the boot abs specifications.

If no options are specified, the ABS file is set up to define the boot ABS area.

This verb is intended for use only by system development personnel.

Syntax

set-abs file.reference {(options)}

Options

number Designates the number of frames to load into boot ABS. "number" should be large enough to ensure the entire code area in the ABS area is loaded. If "number" is not specified, the entire ABS area is loaded. This option is relevent only with the "l" option.

l Copies the ABS area from the ABS file to boot ABS. PCB0 is not overwritten.

s Sizes the ABS file to the size of boot ABS. If boot ABS is larger than specified in the ABS file, frame specification items are added to the ABS file as needed.

Example

```
set-abs abs (l950)
This loads 950 frames from the ABS file into boot ABS.
```

set-batch

displays or changes the current sensitivity value for the interactive/batch trigger.

"port.number" is the D3 pib number, which must be an integer number between 0 and the maximum number of ports. (see "maxusers").

"*" in place of a port.number sets/displays the data for all ports on the system.

"trigger" is a sensitivity value which defines how long it takes for a process to become a "batch process". If "trigger" is not specified, the current setting is displayed. The trigger can be one of the following:

0 Force the process to stay in "interactive" mode, giving access to all of the memory.

[1,127] Set the process to become batch after "trigger" disk reads without any keyboard input. After that point, the process will be prevented from using excessive amounts of memory and will be disabled for a short period, set by set-batchdly, if it attempts to do a disk read when an interactive process is accessing the disk. This tends to limit the impact of batch jobs reading a lot of data (eg an access statement) on the overall system performance.

[-128,-2] Set the process to become batch after abs("trigger") disk reads, as above. The difference is that, in addition to having the same limits as in the previous case, the process will also be disabled for a short period every abs("trigger") disk reads, even if there is no concurrent disk access by an interactive process. This setting is useful for processes which modify a large amount of data. Note the performance penalty on these processes can be severe, depending on the setting. This option should be used only on background tasks. "trigger=-2" is the most stringent setting (disable process every other disk access); "trigger=-128" is the least penalty (disable process every 128 disk reads).

The display includes the total number of reads, and the total number of reads done as a batch process since the last boot or since the last time the counters were cleared.

Syntax

set-batch {port.number{,trigger}} {(options}

set-batch {*{,trigger}} {(options}

Options

b Displays batch processes only.

i Displays interactive processes only.

p Directs output to system printer, via the Spooler.

q Quiet mode. Suppresses all terminal output when changing values.

Example

```
set-batch
PIB TRIG STATUS READS TOT.READ TOT.BTCH %BATCH  05 Mar 1992 PAGE 1
29  127 INTER     11      744        0   0.00
In this example, port 29 is "interactive", has a batch trigger of 127, has
read 11 times since the last keyboard input, has read 744 times since the last
boot time (or since the counters were cleared), and has done "0" (zero) reads
while defined as a "batch" process.


Legend:
"PIB" is the D3 port.number (Process ID).
"TRIG" is the trigger value.
"STATUS" is the current status of the process, either: INTERactive, or BATCH.
```

```
"READS" is the number of disk reads since the last keyboard input.
"TOT.READ" is the total number of disk reads since the last boot, or since the
counters were cleared (logon).
"TOT.BTCH" is the number of disk reads done while it was a batch process.
"%BATCH" is the percentage of disk reads done while it was a batch process, as
compared to the total number of disk reads.
set-batch 3,0
Sets the process 3 to forced interactive mode. Whatever the number of disk
reads the process does, it will remain 'interactive' (highest priority).
set-batch 5,100
Sets the batch trigger counter of process 5 to 100. After 100 disk reads
without any keyboard input (for a select, for instance), the process will
become 'batch', and have a lower priority than interactive processes.
set-batch 10,-50
Sets the batch trigger counter of process 10 to -50. After 50 disk reads
without any keyboard input (for a select, for instance), the process will
become 'batch', and will be disabled for a short time every 50 disk reads.
```

set-batchdly

displays or changes the current sensitivity value for the interactive/batch trigger, as well as the weight of the trigger and the trigger cancel value.

Adjusting the relative priority of batch and interactive processes is done by a dynamic adjustment of both memory usage and disk access.

The setting affects the whole virtual machine, and remains in effect until the virtual machine is rebooted.

trigger.value

Number of concurrent disk reads performed by interactive processes that must be occurring for a batch process requesting a disk read to be suspended. This value should normally be set to one (1).

trigger.weight

Bias value, expressed in milliseconds, to determine how important interactive processes are over batch processes. Its default value is 0. A null value minimizes the difference between interactive and batch processes. Batch processes are simply prevented from using too much memory.

cancel.trigger

Number that allows the entire machine to become batch when no interactive activity is taking place. Its' value is derived by taking the total number of memory buffers and dividing by 2.

Syntax

set-batchdly {{trigger.value}{,{trigger.weight} {,{cancel.trigger}}}}

Example

```
set-batchdly
Trigger : 1. Delay : 1000 ms. Continous batch reads trigger : 9000
Display the current setting.
set-batchdly 1,100
Old trigger : 1. Delay : 1000 ms. Continous batch reads trigger :
9000
Set the a new 'trigger.weight' to 100 ms. In effect, this would artificially
suspend batch processes which attempt doing a disk read while an interactive
process is doing a disk read. The continuous batch trigger is unchanged.
:set-batchdly ,,500
Old trigger : 1. Delay : 100 ms. Continous batch reads trigger : 9000
Set the the 'cancel.trigger' value to 500. This indicates that when there are
```

```
more than 500 disk reads performed by batch processes without any intervening
interactive reads, normal priority will assigned to batch reads.
```

set-baud

sets the baud rate for the current port or for another port.

The form 1 is valid for all D3 platforms.

The form 2 is valid for all D3 Unix platforms only.

"port.number" is the serial port to change. If the port.number is not specified, the baud rate is set on the current port. A "-1" may also be specified to indicate the current port.

The baud "rate" may be one of the following supported baud rates: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200. The default baud rate for each port is 9600.

On D3 Unix systems, the port.number may be specified either by a number or by a device name, such as /dev/tty3, in which case the port does not have to be connected to the D3 virtual machine. /dev/ can be omitted. This command is provided for compatibility with non-Unix implementations.

"word.len" is the data length and may be any number between 5 and 8, inclusive.

If no Unix process is connected to the device, "set-baud" displays:

Process not connected

In this case, the device is probably not initialized properly or not connected to a physical terminal. The form with a tty name instead of a port number should then be used.

"parity" is the data parity. The following values are permitted:

0 (zero) or "n" indicates no parity.

1 (one) or "o" indicates odd parity.

2 or "e" indicates even Parity

There is no "Mark" or "Space" parity supported.

"stop.bits" is the number of data stop bits. May be 0, 1, or 2.

Syntax

set-baud {port.number{,rate,parity, stop.bits,word.len}}

set-baud {tty{,rate,{parity,stop.bits,word.len}}}

Example

```
set-baud
Line number :        4 (/dev/tty4)
Baud rate   :     9600
Parity      :     NONE
Stop bits   :        1
Word length :        8
```

set-break

displays or changes the value of the break character used as an optional substitute for the <break> key for the current port.

The initial value of this character is set for the entire virtual machine by the "brkchr" statement in the configuration file.

Without any argument, this command displays the current value.

"set-break off" disables the <break> key.

The value to assign to the <break> key may be referenced by its ASCII or hexadecimal value.

Using "esc" as the value assigns 27 (x'1b').

Using "off" as the value assigns 255 (x'ff').

When this function is enabled by assigning a value other than x'ff' to this character, and when the command "brk-level" has been issued, hitting the corresponding key pushes one more TCL level. When the "brk-debug" command is issued, hitting the corresponding key enters the FlashBASIC or system debugger.

Assigning an "unreasonable" value to the break character will have unpredictable results. For example, "set-break 13" makes the <return> key push a level or <break>.

Syntax

set-break {decimal value} {(options}

set-break {.hexadecimal value} {(options}

set-break {off} {(options}

set-break {esc} {(options}

Options

q Quiet. Suppresses all terminal output.

Example

```
set-break
Break value x'1C' (decimal 28)
Displays the current definition for the <break> key.
set-break 3
Break value x'03' (decimal 3)
Previous value x'1C' (decimal 28)
Defines the <break> key as <ctrl>+c.
set-break esc
Break value x'1B' (decimal 27)
Previous value x'03' (decimal 3)
This defines the <escape> key as the <break> key.
```

set-cmem

displays or changes the size of the data area used for the BASIC/C or BASIC/GM Assembler interface for the current process.

"size" is the new data size. The valid range is from 1 to 128, in Kilobytes. If size is not specified, the current size is displayed. The new size remains valid for the current process until the system is shutdown. At boot time, the default size is 32K.

Syntax

set-cmem {size} {(options}

Options

q Quiet mode. Suppresses all terminal output.

Example

```
set-cmem
BASIC/C data size: 32768 bytes (32 K)
```

set-date

sets the system software date, and optionally the system hardware clock.

The date displays automatically at the TCL prompt by simply pressing <return>, <enter> or <ctrl>+m.

The "time" command displays the time and date.

On D3 Unix, the D3 system date is set as a difference from the Unix system date. Normally, the difference is zero, making both dates equal.

Synonym for "set.date" and "set.dt".

Syntax

set-date mm/dd/yy{yy}

set-date dd mmm yy

Example

```
set-date 01/01/97
```

set-date-eur

toggles the "standard" (American) date format to "European/International" format: "dd/mm/yy" for input and output purposes.

set-date-format

allows changing the domestic time/date format using items from the "DATE-FORMAT" file.

The valid "country.code" list includes:

US United States.

GR German.

IT Italian.

SP Spanish.

UK English/United Kingdom.

Syntax

set-date-format {country.code} {(options}

Options

s This option sets system date format using the item "DATE-FORMAT" in the "SYSTEM-CONFIG" file.

u This option updates the item "DATE-FORMAT" in the "SYSTEM-CONFIG" file using the given "country.code".

set-date-std

toggles the "European/International" date format to "standard" (American) format: "mm/dd/yy" for input and output.

set-decimal

Designates that a numeric expression of 1 is equal to 100 units.

Syntax

set-decimal {(options}

Options

u set system default

Example

```
:set-decimal
:u bp test
01 print oconv(300,'u01b6');* Unit input-conversion user exit
02 print oconv(5,'u11b6');* Unit output-conversion user exit
This program will display:
3
500
```

set-device

attaches the sct, floppy or half-inch or special devices.

This command is a more generalized form of the standard "set-half", "set-sct", "set-8mm" or "set-floppy" commands.

"device" may be expressed as a number, through options or by up to three keywords extracted from the fields DEVICE, TYPE or OPTIONS in the list-device command. Devices are numbered from 0 to 9 as they appear in the configuration file. Note that the options may not allow complete selection. If a system has two identical devices, for example, two 5 1/4" floppy drives, the options will select only the first. The form with a device number must be used for these cases, using either the device number or the Unix device name as keyword. If no argument is provided, "set-device" simply lists the currently defined devices, as list-device.

"device" is a device number, from 0 to 15, obtained from the "list-device" command or can be composed of up to three keywords identifying the device.

The available keywords are: 720k, 1.2M, 1.44M, 3 1/2", 5 1/4", floppy, sct, 8mm and "half inch". 2.88 megabyte diskette drives are supported.

Syntax

set-device {?} {(options}

set-device {device} {(options}

set-device {keyword} {(options}

Options

3 3-1/2" floppy.

5 5-1/4" floppy.

9 9-track, 1/2" tape.

a 5-1/4" floppy.

b 3-1/2" floppy.

c Changes the 8mm tape block size. The current tape block size is displayed and the operator is prompted for the new block size (0 or 512). On D3 Unix Systems, the default block size is set at "0" (zero) at install time. Zero means variable-length.

e Selects 2.88mb floppy.

f Floppy (3-1/2" or 5-1/4" as default)

h High density (1.44M for 3-1/2" floppy, 1.2M for 5-1/2" floppy, 6250 bpi for 9-track tapes, 150M for Quarter Inch tapes).

i Suppresses display of devices after the command is complete (used in macros).

k Hard disk "pseudo" tapes. These devices are removable or fixed hard disks or regular Unix files. Their size is fixed and determined by the device or by the maximum file size for a user. The main usage for these devices is for small, fast t-dumps/saves, transaction logging or incremental saves.

l Low density (720K for 3-1/2" floppy, 1600 bpi for 9-track tapes).

m Medium density (720K for 5-1/4" floppy, 3600 bpi for 9-track tapes).

n Network "pseudo" tape. This device has an 'infinite' size. It is assumed another system is reading the data at the 'other end' of the network.

q quarter Inch tape.

r Prevents an automatic rewind on the device. This option must be used to set the device to a floppy with an unformatted floppy disk in the drive.

s Standard density (360 for 5-1/2" floppy, 60M or 120M for quarter-inch (SCT) tapes.

Example

```
set-device floppy 1.44m
Selects the 3 1/2" floppy.
set-device rmt0
Selects the tape device associated with the Unix device /dev/rmt0.
set-device 0
Block size: 500
[1701] Tape device is assigned to 3 1/2" high density (1.44M) floppy
drive.
Tape        Status           16 Jan 1997  14:30:40
 #   Type             Density           Owner   Device Name
-------------------------------------------------------------------
 0 | Floppy        | 3 1/2" 1.44M    | 191*+ | /dev/rpdsk/4
 1 | Quarter Inch  | High            |       | /dev/rmt/0n
 2 | 4mm DAT       |                 |       | /dev/rmt/1n
 3 | 4mm DAT       |                 |       | /dev/rmt/1un
 4 | 8mm Tape      |                 |       | /dev/rmt/2n
 5 | 8mm Tape      |                 | 141   | /dev/rmt/2un
 6 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/abs
 7 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/data
 8 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/ref
 9 | Network       |                 |       | /home/tmp/pipein
10 | Floppy        | Pseudo Floppy   | 191*  | /home/tmp/floppy
11 | Floppy        | Pseudo Floppy   |       | /n/dev/home/tmp/floppy
-------------------------------------------------------------------
Selects the tape device associated with the unix device /dev/rpdsk/4

The "Owner" column displays the pib number that has use of the
device, and a "*" indicates that this pib is the current process. A
"+" indicates that this device is the "active" device (as a pib can
attach more than one device).
```

set-device

attaches the sct, floppy or half-inch or special devices.

This command is a more generalized form of the standard "set-half", "set-sct", "set-8mm" or "set-floppy" commands.

"device" may be expressed as a number, through options or by up to three keywords extracted from the fields DEVICE, TYPE or OPTIONS in the list-device command. Devices are numbered from 0 to 9 as they appear in the configuration file. Note that the options may not allow

complete selection. If a system has two identical devices, for example, two 5 1/4" floppy drives, the options will select only the first. The form with a device number must be used for these cases, using either the device number or the Unix device name as keyword. If no argument is provided, "set-device" simply lists the currently defined devices, as list-device.

"device" is a device number, from 0 to 15, obtained from the "list-device" command or can be composed of up to three keywords identifying the device.

The available keywords are: 720k, 1.2M, 1.44M, 3 1/2", 5 1/4", floppy, sct, 8mm and "half inch". As of release 6.0, 2.88 megabyte diskette drives are supported.

Syntax

set-device {?} {(options}

set-device {device} {(options}

set-device {keyword} {(options}

Options

3 3-1/2" floppy.

5 5-1/4" floppy.

9 9-track, 1/2" tape.

a 5-1/4" floppy.

b 3-1/2" floppy.

c Changes the 8mm tape block size. The current tape block size is displayed and the operator is prompted for the new block size (0 or 512). On D^3 Unix Systems, the default block size is set at "0" (zero) at install time. Zero means variable-length.

e Selects 2.88mb floppy.

f Floppy (3-1/2" or 5-1/4" as default)

h High density (1.44M for 3-1/2" floppy, 1.2M for 5-1/2" floppy, 6250 bpi for 9-track tapes, 150M for Quarter Inch tapes).

i Suppresses display of devices after the command is complete (used in macros).

k Hard disk "pseudo" tapes. These devices are removable or fixed hard disks or regular Unix files. Their size is fixed and determined by the device or by the maximum file size for a user. The main usage for these devices is for small, fast t-dumps/saves, transaction logging or incremental saves.

l Low density (720K for 3-1/2" floppy, 1600 bpi for 9-track tapes).

m Medium density (720K for 5-1/4" floppy, 3600 bpi for 9-track tapes).

n Network "pseudo" tape. This device has an 'infinite' size. It is assumed another system is reading the data at the 'other end' of the network.

q quarter Inch tape.

r Prevents an automatic rewind on the device. This option must be used to set the device to a floppy with an unformatted floppy disk in the drive.

s Standard density (360 for 5-1/2" floppy, 60M or 120M for quarter-inch (SCT) tapes.

Example

```
set-device floppy 1.44m
Selects the 3 1/2" floppy.
```

```
set-device rmt0
Selects the tape device associated with the Unix device /dev/rmt0.
set-device 0
Block size: 500
[1701] Tape device is assigned to 3 1/2" high density (1.44M) floppy drive.
Tape        Status           16 Jan 1997  14:30:40
 #   Type             Density           Owner   Device Name
-------------------------------------------------------------------------
 0 | Floppy        | 3 1/2" 1.44M    | 191*+ | /dev/rpdsk/4
 1 | Quarter Inch  | High            |       | /dev/rmt/0n
 2 | 4mm DAT       |                 |       | /dev/rmt/1n
 3 | 4mm DAT       |                 |       | /dev/rmt/1un
 4 | 8mm Tape      |                 |       | /dev/rmt/2n
 5 | 8mm Tape      |                 | 141   | /dev/rmt/2un
 6 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/abs
 7 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/data
 8 | Floppy        | Pseudo Floppy   |       | /usr/opt/pick/bin/ref
 9 | Network       |                 |       | /home/tmp/pipein
10 | Floppy        | Pseudo Floppy   | 191*  | /home/tmp/floppy
11 | Floppy        | Pseudo Floppy   |       | /n/dev/home/tmp/floppy
-------------------------------------------------------------------------
Selects the tape device associated with the unix device /dev/rpdsk/4

The "Owner" column displays the pib number that has use of the device, and a
"*" indicates that this pib is the current process. A "+" indicates that this
device is the active device (as a pib can attach more than one device).
```

set-dozens

Designates that a numeric expression of 1 is equal to 12 units.

Syntax

set-dozens {(options}

Options

u set system default

Example

```
:set-dozens
:u bp test
01 print oconv(36,'u01b6');* Unit input-conversion user exit
02 print oconv(5,'u11b6');* Unit output-conversion user exit
This program will display:
3
60
since 3 dozen = 36, and 5 dozen = 60.
```

set-dptr

changes the first attribute of file-defining D pointers, to set or reset special attributes, like DX or DY.

The attribute list consists of a plus sign (+) to SET an attribute or a minus sign (-) to REMOVE an attribute, followed by a string of one character attribute(s).

"attributes" One or more characters among the valid attributes:

X Y L S P N U

If no attribute change is specified, then the types of all the specified files are simply displayed. It is not allowed to turn ON the (L) attribute and either of (X) or (Y). When trying to set the (L)

attribute, files currently DX'ed or DY'ed will not be altered. Similarly, files currently DX'ed or DY'ed, will not have the (L) attribute set. The N and the U options are mutually exclusive.

"item.list" List of elements to change. If not explicitly stated, a select list must be used. If the (A) option is used, the specified items are a list of accounts. ALL files in the account, dictionaries and data levels are altered. The D pointer defining the account in the "mds" file, is also altered. If the (F) option is used, the specified items are a list of files. If specifying a simple file name, like 'bp', then the dictionary and ALL data levels are altered. If specifying an explicit data level, like 'bp,bp', then only the specified data level is altered. Explicit path names are accepted.

At the end, the number of files actually changed is displayed. In case of error, the error messages are also displayed at the end.

Syntax

set-dptr {[+|-]attributes} {item.list} {(options}

Options

A Change account(s). "item.list" is a list of account(s).

F Change file(s). "item.list" is a list of file(s). The (F) option is the default.

P Print all changes to the printer.

Q Quiet. Suppress user message.

Example

```
set-dptr +p bp (f
Set the (P) attribute to all data levels of the file 'bp' in the
current account.
set-dptr +x pa,entity, (f
Set the (X) attribute to all data levels of the file 'entity' in the
account 'pa'
set-dptr +x -l pa,entity,backup (f
Set the (X) attribute, remove the (L) attribute to the data level
'backup' of the file 'entity' in the account 'pa'
select mds
set-dptr -xy (a
Select all accounts in the system, and remove all (X) (Y) attributes
in all files in all accounts. Note it is not necessary to select
items in the "mds" file which are real D pointers. "set-dptr" ignore
q-pointers, and items which do not 'look' like D pointers.
select mds # "dm"
set-dptr +l -y (a
Select all accounts in the system, except "dm", add the (L) option
and remove all (Y) attributes in all files in all the selected
accounts.
```

set-esc

specifies the value of the key to treat as the <escape> key. This key subsequently pushes a level when pressed.

The key to trigger a level push may be redefined by indicating the decimal or hexadecimal value of the character to use in place of <escape>.

Valid arguments:

".hexnum"

Indicates the key, designated by its corresponding hexadecimal value, to trigger a level push.

"decnum"

Indicates the key, designated by its corresponding decimal value, to trigger a level push.

"off"

Disables this function by defining escape as x"ff".

"esc"

Defines escape as x"1B", the normal value for escape.

"?" Displays command usage information without changing anything.

Syntax

set-esc {argument} {(option}

Options

q Quiet mode; no output is displayed.

Example

```
set-esc
Escape value x'1C' (decimal 28)
This checks the current setting for the escape key.
set-esc 3
Escape value x'03' (decimal 3)
Previous value x'1C' (decimal 28)
This defines the <escape> key as <ctrl>+c.
:set-esc esc
Escape value x'1B' (decimal 27)
Previous value x'03' (decimal 3)
This defines the <escape> key as the <break> key.
```

set-file

creates or updates the synonym-defining item called "qfile" in the md of the current account.

This provides access to the specified filename in the specified account.name by using the filename, "qfile", in place of the filename parameter during any file-handling commands.

Each time "set-file" is invoked, it overwrites the previous "q-pointer".

In D3, the same functionality may be achieved using "pathnames". See "file.reference".

Syntax

set-file {account.name {file.name}}

Example

```
set-file dm bp
'qfile' updated.
list qfile
```

set-floppy

attaches the specified diskette drive to the current process for subsequent input or output activity. The drive indicator and density are indicated with the options provided.

This is used after the "t-att" command to designate the density to read/write diskettes and which floppy drive to use, when more than one drive is present.

Drive choices are typically "a" or "b".

The default values are drive "a", high density.

The default block size is 500.

Syntax

set-floppy {(density,drive)}

Options

e Selects 2.88mb floppy.

h High density (1.2mb for 5-1/4 inch, 1.44mb for 3-1/2 inch)

s Standard density (360kb for 5-1/4 inch, 720kb for 3-1/2 inch)

D3 Unix implementation-specific options:

a Select 5 1/4" floppy.

b Select 3 1/2" floppy.

h Select 1.2M if 5 1/4", or 1.44M if 3 1/2".

s Select 360K if 5 1/4".

l Select 720K if 3 1/2".

Example

```
set-floppy (ah
This attaches floppy diskette drive "a" at high density.
```

set-flush

displays or changes the value of the period of the flush and of the forced flush.

"period" is the new period (in seconds) with which the flush process will wake up automatically to write "dirty" (write-required) frames back to disk. The valid range is from 500 to 32000 milliseconds (0.5 to 32 secs). If "period" is not specified, the current flush period is displayed.

The default period is defined at boot time by the "flush" statement in the configuration file.

"force.period" is the new period (in seconds) with which the flush process will wake up automatically to write all buffers containing data belonging to the file system (item, indexes, etc...). This period is the guaranteed time after which an update to a protected file will be written to disk (see "flusher" for a more detailed explanation of this functionality). The following values have a special meaning:

0 All writes are done by the process itself. In other words, when a process updates an item, for example, the process waits until the write is done. The keyword "imm" can be used to specify this option.

-1 The forced flush mechanism is disabled. With this option, there is no guaranteed time for an update to the file system to be written to disk. The keyword "off" can be used to specify this option.

The default period is defined at boot time by the "fflush" statement in the configuration file.

Syntax

set-flush {{period}{,force.period}} {(options}

set-flush {{period}{,off}} {(options}

set-flush {{period}{,imm}} {(options}

Options

q Quiet mode. Suppresses all terminal output.

Example

```
set-flush
Flush period 10 s. Forced flush period 5 s.
This indicates that the current "flush" period is set at 10 seconds,
and the forced flush period is 5 seconds.
set-flush ,imm
Flush period 10 s. Forced flush IMMEDIATE.
Set the forced flush in immediate mode. The normal flush period is
not changed.
set-flush 30,off
Flush period 30 s. Forced flush OFF.
Disable the forced flush in immediate mode and set the normal flush
period to 30 seconds.
```

set-func

defines the function keys available on port 0 (zero) of PC implementations only. Note that the filename is required, and is usually called "funckeys".

The function keys are initially defined by the default keyboard definition item. "set-func" overrides the default settings or the previously executed "set-kbrd". However, if "set-kbrd" is executed after "set-func" is executed, the function keys are reset according to the keyboard definition item used.

Each function key may be assigned one character.

Function keys are undefined by default.

Syntax

set-func file.reference item-id

Example

```
set-func funckeys ibm
```

set-half

see set-half (Tape)

set-imap

defines a keyboard input and/or a terminal output translation table, through which any sequence of keyboard input and terminal output characters can be translated into any other sequence of characters.

Input translation can be used to translate special key sequences, like "ESC [A", into a sequence understandable by the application.

Ouput translation can be used to convert a character into an appropriate escape sequence, for example, to change fonts on a printer, print the character, and change the fonts back.

The translation is based on the notion of 'input sequence', which is a variable-length series of characters which must be received completely within a given time, typically 1/10th of a second, to be recognized as one key stroke, and converted into an output sequence. If an input sequence is not received within the specified time, or if a character received is not part of a valid sequence, the sequence is aborted, and all characters received so far are 'de-sequentialized' and passed to the application or displayed as a series of discrete characters.

When applied to a terminal output translation, there is no notion of timeout, but there are some restrictions (see the section 'Warnings' below).

The translation is described in a table which is associated to each port. See the Reference Manual 'keyboards' entry for the format of the table. A table can be shared among different ports. Each input and output table contains a 'main' translation table and an optional 'alternate' table. The two tables have identical structure and capabilities. The main table is active when the translation is activated. A special input sequence can be defined to switch to the alternate table until it is switched manually back, or for one translation (keystroke) only.

An input sequence can have from one to 127 characters. An output sequence can be from 0 to 127 character long. If an output sequence is null, the corresponding key is made inoperative (input) or the data is not displayed (output).

Without a numeric option, the current port is affected. Optionally, a specific port can be specified by using the port number as a numeric option.

A <BREAK> key aborts any pending sequence and sets the main translation table as the active one.

If there is no argument, the translation mechanism, if currently active, is disabled.

"item.id" The item-id of the translation table, located in the 'keyboards' file in the 'dm' account. The format of the keyboard item is described in the Reference Manual entry 'keyboards'. If the item has already been compiled into a translation table and stored as a binary item in the dictionary of the file, the translation item is not re-compiled, unless the "c" option is used.

"time.out" Value of the time out, expressed in milliseconds, after which an incomplete input sequence is aborted. If not specified, the value defined in the item "item.id" by the 'timeout value' modifier in attribute one, is used. If none is specified, the default value is 100 milliseconds. The "time.out" should be adjusted to the baud rate and possible special conditions, like network delays, to detect sequences of characters properly. If "time.out" is 0, the translation mechanism will wait indefinitely between characters of a sequence until either a valid sequence is received or until an unexpected character 'breaks' a sequence. The timeout does not apply to output translation. See the section "warning" below for information about adjusting the timeout value.

If the one-to-one input/output translation defined by the TCL command 'set-iomap' is also active on this port, the translation defined by 'set-imap' is processed first, and each character of the resulting string is passed through the one-to-one conversion table set by 'set-iomap'.

When used on PC-based D3/SCO, the keyboard setting defined by 'set-kbrd' is processed first.

set-imap can be called automatically by the TCL command TERM with the (K) option, if the item defining the terminal has the name of a keyboard translation item in the value 4 of attribute 1.

Syntax

set-imap {item.id {time.out}} {(options}

Options

port.number Port number in decimal. If not specified, the current port is used.

c Compiles the item. This option must be used when "item.id" is modified. "item.id" is compiled into a binary item and stored in the dictionary of the "keyboards" file.

v "verbose". Displays information about the translation table (name, size) and the modifiers used in attribute one.

Example

```
set-imap ibm3151
  Sets the keyboard translation to an ibm3151.
set-imap wy-50 100 (24
  Sets the keyboard translation to a wy-50, changing the timeout to
100 ms, on port 24.
set-imap att605 (c
  Sets the keyboard translation to an att605, recompiling the item
first.
set-imap
  Disables the keyboard input translation.
```

set-incremental

Enables or disables the incremental save feature.

By default, the incremental save is enabled at every boot. Some users who are not using the incremental save may wish to disable this feature as this makes the "clearing dirty bits" phase of the standard save take far less time.

Typing "set-incremental" with no options displays the current status.

If the user wishes to use the incremental save after disabling it with this command, it is necessary to turn the incremental feature on, do a FULL save, and then do an incremental save. Failure to do the full save before the first incremental will produce invalid results.

To keep the incremental save disabled permanently, it is necessary to place the line "set-incremental (f" in the "user-coldstart" macro in "dm,md,".

Options

f Turns off the incremental save.

n Turns on the incremental save.

set-incremental-off

Disables the incremental save.

Syntax

set-incremental-off

set-incremental-on

Enables the incremental save.

Syntax

set-incremental-on

set-iomap

allows translatable input and output for each port on the system.

"port.number" is the number of the port (or pib) for which a keyboard table is installed.

"id" is the four-character item-id of the user-defined keyboard item. It must be numeric, and is not required when used with any of the following combinations of options: "ir", "ri", "or" or "ro".

The Virtual software of D3 reads the keyboard item and passes it to the monitor. "id", the item-id, is copied to the ID field of the I/O table translate. The counter in the second field is incremented or decremented by one depending on the option. The character set is copied to the third field of the entry. Depending on the option specified by the user, either IT or OT pointer is

replaced by the address of the entry. The new keyboard table overwrites the old one even if a match is found when comparing the new item-id with the one in the table.

Monitor notifies virtual if an installation is not possible and virtual, in turn, notifies users of such results.

There is a file in the "dm" account called "iomap-file". This file contains keyboard items that are defined by users. The item-id should be numeric and should be in the range of 0 <= item-id <= 2147483647. The keyboard item has 34 attributes and has the following format:

attr 0 : item-id

attr 1 : user comments

attr 2 to 33 : ASCII code of characters from 0 to 127 in ASCII hex format. For example: character 'A' has its ASCII code is x'41' and should be entered in the item as '41'. Each attribute from 2 to 33 has exactly 8 entries each. User can put his/her comments after the 8th entry on that attribute.

The following example shows how to translate the character 'A' to 'B':

- ASCII code of A is x'41' = d'65'

- ASCII code of B is x'42' = d'66'

- 65 mod 8 = 8 with remainder 1

- Translated character, which is B, should be entered at attribute 8+2= 10 and at column 1+1 as 42. The reason that 2 is added to 8 is because counting begins at 0 and attr 1 is used for comments.

Syntax

set-iomap port.number{,id} ([i|o|ri|ro]

Options

? Provides on-line help

i Installs input keyboard.

o Installs output keyboard.

ir (or ri) Removes an installed input keyboard.

or (or ro) Removes an installed output keyboard.

set-kbrd

defines the type of keyboard character set in use on line 0 (zero) of PC implementations only.

The standard keyboard definitions provided by D^3 reside in the "keyboards" file and include : British (English), French, French 2, German, Italian, Spanish, Spanish 2 and USA (the default).

Syntax

set-kbrd file.reference item-id

set-keys

defines function keys on port 0 (zero) of pc-class systems only.

Syntax

set-keys {keyboard.name}

Options

? Displays on-line help text.

set-lptr

adjusts communications parameters to optimize parallel printer performance.

To obtain an optimum blend of printer and system performance, Pick Systems advises experimenting with different settings.

The values for the parameters depend on the printer type, the number of users on the system, and the aspect of system performance that is most important. Some values increase the amount of work the printer can do, but lengthen the time it takes for the system to respond to users. Other values shorten the time it takes for the system to respond to users, but decrease the amount of work the printer can do.

If no parameters are specified, the current settings for all printers are displayed. If only the printer number is specified, the current settings for that printer are displayed. If parameters are entered, only those that are being changed need to be specified. However, commas are needed to indicate missing values. Like the "term" command, existing parameters may be left intact by providing two successive commas where the parameter would normally appear.

"number" is the printer number and may be from 0 through 3, inclusive.

"delay" is the number of times to attempt to send a character to the printer. May be in the range from 1 through 4095. This number relates to the speed of the printer and the number of users that are sending jobs to the printer. If the number is too small, the printer performance degrades. If the number is too large, response time for the users degrades. The default is 300.

"burst" is the number of characters sent to the printer during one transmission. It may be in the range from 1 through 255. This number relates to the size of the printer buffer. In general, 15 provides good performance. The default is 200.

"slice" is the number of timeslices the printer waits before receiving its next time slice. May be in the range from 1 through 15. This number relates to the number of users and number of printers on the system. In general, the number should be equal to or greater than the number of users on the system (max is 15). The default is 1.

Syntax

set-lptr

set-lptr number

set-lptr {number{,{delay},{burst},{slice}}

Example

```
 set-lptr
[1025] Printer 0 Delay=300 Burst=200 Slice=1
[1025] Printer 1 Delay=300 Burst=200 Slice=1
[1220] Printer 2-No controller is installed.
This is sample output from an AP/Native implementation.
```

set-num-format

changes the currency sign, thousand, and decimal point in an 'm' conversion, at a local or global level. All of these signs must be a single character.

"port.number" specifies the port number to be set. If both "port.number" and "d" are not specified, the current port is implied.

"cur" is the currency sign.

"thou" is the thousand mark.

"dec" is the decimal mark.

The defaults for these are the previous settings.

The local setting overrides the global default setting. No local setting defaults to global setting. Local or Global setting may be cancelled by not specifying the "cur", "thou", and "dec" parameters. Logging off cancels the local setting.

Syntax

set-num-format {{cur}/{thou}/{dec}} {(options}

Options

d Sets system-wide default: "$/,/.".

port.number Indicates that the setting will only affect the desgnated port.

set-ovf-local

sets and displays the local overflow cache size. The presence of a local overflow cache enhances both performance and reliability, and is automatically set up to a reasonable default by the system. Users who wish to further tune their overflow usage may use the "set-ovf-local" command to change this cache to tailor it to specific needs.

When given without any options, the "set-ovf-local" command displays the cache status for the current line. The display shows a 2 by 2 grid of numbers. The explanation for these is as follows:

Legend: (see examples)

"Current" is the current number of frames actually held in a given cache.

"Max" is the maximum number of frames that the cache may hold. Any overflow released to the cache when it has reached its maximum size will be deposited directly into the main overflow table.

"WS" is the row showing the current and maximum workspace cache sizes.

"File" is the row showing the current and maximum file cache sizes.

Parameters:

Besides displaying the current cache status, "set-ovf-local" can be used to modify the current cache maximums by specifying the following numeric parameters on the command line:

ws.max This sets the maximum workspace cache size.

fs.max This sets the maximum filespace cache size.

Cache descriptions:

The workspace cache is a generalized cache used for virtually all memory needs. It is automatically set to a default so that simple TCL commands and FlashBASIC programs will not need to access the global overflow table and can thus avoid the performance cost of doing so. If the user is repeatedly using EXECUTE's of more complex programs, then he/she can try boosting the workspace cache size to see if performance improves.

The file cache is used only when update-protection is active. The update-protection scheme deposits frames into this buffer which it guarantees are syncronized so that no other file or workspace on the disk points to it. This virtually eliminates the posibility of so-called "doubly

linked frame" where two files are attached to the same frame after a power outage or crash. This cache should be large enough to cover the largest group or largest pointer item that is protected by the update protection scheme. For example, if the user has a file with 30K byte items on a 2K frame system, then the file cache should be set to at least 30K/2K = 15 frames.

Cache tuning:

Ideally, a system should be able to operate out of private overflow caches as much as possible. The less the system accesses the global overflow pool, the greater the system performance, and the less chance there is for the overflow to become corrupted in the event of a power outage. To tune the cache sizes, users can try different cache settings, and run the "buffers (s" program to see the change on overflow access. The field "WS OVF locks" shows the number of system-wide references per second to WS overflow tables. The field "FILE OVF locks" shows the number of references to the global overflow table per second.

Syntax

set-ovf-local {ws.max}{, fs.max} {(options}

Options

s supresses the display.

f flushes all caches. All frames held within the current cache are released back to the main table. Note that this operation is automatically invoked when the maximum values are changed. The act of logging off the system will automatically flush all local overflow caches.

g Copies the current maximum cache settings into the global default. Whenever a new user logs in, he/she will automatically aquire this default value. Note that the global defaults are automatically preset to a factory specified value upon reboot, so if a permanent global change is desired, the user should place the "set-ovf-local" command in the system-coldstart macro.

Example

```
set-ovf-local

Private Overflow Cache Status:
           Current   Max
WS            2      20
File         10      30
This shows that there are currently 2 workspace frames in the overflow cache
with a maximum capacity of 20 frames, and 10 file space frames in the overflow
cache with maximum capacity of 30 frames.
set-ovf-local 100,300 (s
This command will set the maximum workspace cache size for the current line to
100, and the maximum file space cache size to 300.
set-ovf-local (fs
This command releases all frames in the current overflow cache to the global
pool.
set-ovf-local 100,300 (sg
This command will set the maximum workspace cache size for the current line to
100, and the maximum file space cache size to 300. The settings will also be
copied into the global default area. All users logging into the system after
this command has been executed will automatically get caches sized to this
specification.
```

set-ovf-reserve

sets aside a specific number of frames as the "spare tank" in the event that the overflow table runs out of frames.

Sets the overflow reserve to "reserve.quantity" frames. The default is 1024.

Syntax
set-ovf-reserve {(reserve.quantity)}

set-port

displays or changes the communication speed and protocol for a given port or displays the current setting if none of the communication paremeters are given.

set-port with tty format is for D3 Unix.

"baud" is the baud rate. Legal baud rates are: 110, 150, 300, 600, 1200, 2400, 4800, 9600, and 19200.

"parity" is "n" or "0" (zero) for no parity. "o" or "1" for odd parity. "e" or "2" for even parity. "m" for mark parity. "s" for space parity.

"stop.bit" is the number of stop bits. Valid numbers are 1 or 2.

"word.length" is the data length. Valid numbers are 7 or 8.

If no options are entered, the current port settings are displayed. If any option is entered, then all options must be entered.

D3 Unix Systems:

The port can be specified either by its serial port.number, or by a device name (e.g., /dev/tty03), in which case the port does not have to be connected to the D3 virtual machine. /dev/ can be omitted. This command is provided for compatibility with non-Unix implementations.

If no Unix process is connected to the device, "set-port" displays:

Process not connected

In this case, the device is probably not initialized properly or not connected to a physical terminal.

"status-port" is a synonym of "set-port"

Syntax
set-port tty{,baud,parity,stop.bits,word.length}

set-remote-user

The set-remote-user command can be used to allow the local user to utilize a different user and/or password when connecting to a remote OSFI server which has been started with the "A" option to force user/password authorization.

When the client process attempts to open a file on a remote OSFI server which has been started with the "A" option, the server refuses the request unless the passed user and password match a user and password present on that server. By default, the client passes its local user and password to the remote server. Provided the same user and password combination is defined on that server, then access is granted. However, it is often not realistic for user names and/or passwords to be identical accross different machines. In this situation, set-remote-user allows the client user to specify that an alternate user/password combination be used when opening files on remote servers.

The first and second syntax forms display the current user mappings.

The third form allows the user to enter (or destroy if used with the k option) a mapping entry for the specified remote OSFI host. The program prompts for the remote user and password to be

used on all future connections to that host. The authorization of this information is performed automatically when the first file from that data source is opened. An authorization failure causes the open to fail.

Note that the way the authorization information is used depends on the remote server configuration. If the OSFI server is not booted with the "A" security option (for "Authorization"), then the passed local user permissions are applied directly to obtaining resources. If the "A" security option is used, then the server first looks for a user on that server called "ClientHostName:ClientMappedUser" where "ClientHostName" is the local host name defined on the client and "ClientMappedUser" is the mapped user specified by the client. If this user is not found (or the password is not correct), then the server tries to locate the local user "ClientMappedUser". When a local user match is found, the file permissions are derived from that user definition on the server. The client permissions are ignored in this case.

Syntax

set-remote-user

set-remote-user ?

set-remote-user host

Options

k Kills the specified host entry.

r Re-verify password. If this option is used, the password is requested twice to ensure its correctness.

s Saves authorization information in the users file. Note that although this information is stored in an encrypted format, it could theoretically be decrypted by a determined individual. It is suggested that security-critical information be entered at every logon rather than have that information stored in the users file.

v Verbose output. This option displays the current user mappings.

Example

```
set-remote-user macha
Remote User      :mark
Remote Password  :
```

If the local user name is "joe", this allows user "mark" access to open files on the MACHA server, with no password.

set-runaway-limit

dislays or establishes the maximum number of runaway overflow frames. If a process acquires 5000 frames or the specified limit, the system detects an "overflow runaway" condition and reports it.

If no parameters are provided with the command, the current runaway overflow limit is displayed.

"number" may either be "0" (zero), which disables the overflow runaway mechanism, or any number over 1000.

Using "d" as the limit sets the limit to the "default" setting of 7000 frames.

The "number" or "d" may be specified with or without a left parenthesis.

Syntax

set-runaway-limit {number | d} {(number | d}

Example

```
set-runaway-limit
[110] Overflow runaway limit set to 7000 Frames
```

set-sct

see set-sct (Tape)

set-sct-dma

see set-sct-dma (Tape)

set-shutdown-delay

displays or changes the number of seconds prior to shutdown.

The "set-shutdown-delay" command is provided primarily for installations which have disk-caching hard-disk controllers. This provides time for the disk to flush.

The delay must be a positive integer between 1 and 65536. Without any parameters, the current shutdown delay is displayed. The default is 512.

Syntax

set-shutdown-delay {number.seconds} {(options}

Options

s Sets monitor shutdown delay using "SHUTDOWN-DELAY" item in the "SYSTEM-CONFIG" file.

u Updates "SHUTDOWN-DELAY" item in the "SYSTEM-CONFIG" file with the given "delay" value.

Example

```
set-shutdown-delay 300
This sets the shutdown delay to five minutes.
```

set-sound

sets the system console sound level on port 0 (zero) only.

set-sym

indicates the symbol table to use for references to system variables from within the system, or virtual, debugger.

The global table, "gsym", is the default.

Syntax

set-sym file.reference {(option}

Options

p Sets "psym" as the symbol file.

t Sets secondary file pointer to "tsym" file.

set-thousands

Designates that a numeric expression of 1 is equal to 1000 units.

Syntax

set-thousands {(options}

Options
u set system default

Example
```
:set-thousands
:u bp test
01 print oconv(3000,'u01b6');* Unit input-conversion user exit
02 print oconv(5,'u11b6');* Unit output-conversion user exit
This program will display:
3
5000
```

set-time

sets the current system time.

The hours are specified from 00 (midnight) to 23 (11:00 p.m.). The current time can be displayed by pressing <ctrl>+m, or <enter> or <return> at the TCL prompt or by typing the TCL verb "time".

Synonym for "set.time".

Syntax
set-time hh:mm{:ss} {(u)}

Options
u This updates the hardware real-time clock, as well as the system (software clock).

set-units

Designates that a numeric expression of 1 is equal to 1 unit.

Syntax
set-units {(options}

Options
u set system default

Example
```
:set-units
:u bp test
01 print oconv(5,'u01b6');* Unit input-conversion user exit
02 print oconv(5,'u11b6');* Unit output-conversion user exit
This program will display 5 for both user exits, as there is a
one-to-one correspondence between units and the numeric expression.
```

set.lptr

see "set-lptr".

set.time

"set.time" is the actual TCL verb invoked by the "set-time" program.

setpib0

resets the system's port 0 (console) to the "proper" terminal type. The default terminal type varies from platform to platform:

Platform	Default Type
D3/RS6000	IBM 3151
D3/SCO	ANSI

D3/SINIX	S97801
D3/HP9000	HP
Others	WYSE 50

setport

sets all ports (except port 0) to 9600 baud, no parity, 1 stop bit and 8 data bits.

setup-printer

downloads fonts to a LASER printer attached to a formqueue.

The form-queues are items in the "dm,devices," file with fonts 1 through 16 defined at the proper table entry (x-125 through 140.)

The list of font item-ids used by a form-queue are in the file "dict dm,fonts," with the item-ids the same as the form-queue.

If a form-queue is indicated, it downloads the fonts in that list.

If a printer number is indicated, for all form-queues defined by that printer, it downloads the fonts in each form-queue list. Thus, by specifying the item-id of the form-queue, it retrieves the list of fonts in "dict dm,devices,".

"form number" is the form-queue number to which the fonts are downloaded.

"printer number" is the printer number to which the form-queue fonts are downloaded.

Fonts reside in the "dm,fonts," file.

The list of fonts assigned to a form-queue resides in "dict dm,fonts form.queue.number"

Setting up a form-queue downloads all the fonts defined for that form queue.

Setting up a printer does a form-queue download for each form queue defined for that printer.

Each (downloaded) font is stored in the printer's memory, and is referenced by a numeric font-id number. To insure that an existing font isn't accidentally overwritten, the beginning font-number of "10" is used, and is incremented by "10" for each additional downloaded font.

The fonts may be invoked using the same "@" functions available to FlashBASIC. See the "@ function" and the arguments between (-146) and (-267), or by the OP ".font<font.number>" command.

setup.rtc

invokes the "start.rtc" command.

sh

Invokes DOS or Unix shell.

shp-kill

terminates a D3 printer shared with Unix and all the associated processes.

"number" is the D3 printer number, as specified in the TCL 'startshp' command.

This command executes the TCL command 'sp-kill D number '.

This command relies on the existence of the log files created in "/tmp/lppick" by the 'startshp' command. It will:

- Do a regular 'sp-kill' of the D3 printer process,

- Send SIGTERM to the underlying D3 process, waiting up to 5 seconds before sending a SIGKILL.

- Make sure the associated 'lppick' filter process is terminated. If it is not the case, a SIGTERM followed by a SIGKILL will be sent to this process.

Syntax
shp-kill number

Example
```
shp-kill 0
```

shp-status

displays status information about printers shared with Unix, started by the TCL command "startshp".

The following information is displayed:

'VMname'

Name of the D3 virtual machine which uses the shared printer.

'Prt'

D3 printer number, as specified in the startshp copmmand.

'Port'

D3 port number the printer process was started on.

'PID'

PID of the 'lppick' process associated to the D3 printer process. This process acts as a filter between the D3 printer output, which is a continuous data stream, separated by 'end of job' sequences, and the Unix spooler which accepts separate jobs.

'Spooler command'

Unix command used to spool data. Only the first 32 characters of the command are displayed.

'Status'

This field report the activity or the D3 printer process and the existence of the lppick filter. The possible values are:

'OK'

Both processes are alive.

'prt OK'

The D3 printer is alive and seems in a normal state.

'prt ERR'

The D3 printer is alive but appears in a wrong state.

'prt DEAD'

The D3 printer process has been killed.

'lp OK'

The lppick process is alive.

'lp DEAD'

The lppick process has been killed.

For normal operations, the status should be 'OK'. However, if the the filter process gets killed, the output of the D3 printer process will not be able to be sent to Unix. Issuing the 'startshp' command again should clear the situation and restart the necessary processes.

If the (T) option is used, the trace information recorded is displayed, starting with the newest trace entry. The following information is displayed:

tr# Date Time Description

where:

tr# : Trace number in decimal.

Date : Date.

Time : Time (Note: The D3 time is displayed)

Description:

"Start job"

Beginning of a job

"End job, size=N (-1,10)"

End of a job. 'N' is the size in decimal. The values between parentheses are internal return codes.

"Write data, size=N"

Write 'N' bytes of data to the 'lp' command.

"Read error, errno=X"

Encountered a read error on stdin. 'X' is the decimal value or 'errno'

"Write error, errno=X, expsz=N, sz=M"

Encountered a write error on stdout (to the lp command). 'X' is the decimal value of errno, 'N' is the number of bytes we attempted to write, and 'M' the actual number of bytes written.

"Raw read, size=N,'<texte>'"

Raw data read from stdin (coming from the D3 printer). Non printable characters are replaced by a '.' 'N' is the total size. This trace is available only when the trace level 3 is used.

"Raw write, size=N,'<texte>'"

Raw data written on stdout (to the lp command). Non printable characters are replaced by a '.' 'N' is the total size. This trace is available only when the trace level 3 is used.

Syntax

shp-status {(options}

Options

t{n} Display trace information. If 'n' is nort specified, all traces are shown, starting by the most recent. If 'n' is specified, only that number is shown for each printer.

Example

```
shp-status
VMname Prt Port  PID Spooler command      Status
------ --- ---- ---- -------------------- ---------------
pick0    0  126 1763 'exec lp -onobanner' OK
dev      0   32 1765 'exec cat >> xx'     prt OK  lp DEAD
```

```
prod     0   44  345 'exec lp -s'         prt ERR lp OK
The first line indicates that the first shared printer is the printer
0 of the virtual machine 'pick0', started on the port 126, sending
its data to the normal 'lp' command, suppressing the banner. It
appears to be ok.
The second line indicates that the second shared printer is the
printer 0 of the virtual machine 'dev', started on the port 32,
sending its output to the Unix file 'xx' (on whatever directory the
command 'startshp' was issued on). The D3 printer process is still
up, but the 'lppick' filter is dead, possibly because the file 'xx'
was write protected, or may be because the Unix file system is full.
The third line indicates that the second shared printer is the
printer 0 of the virtual machine 'prod', started on the port 44,
sending its data to the normal 'lp' command, suppressing lp messages.
The D3 printer process is still up, but in an incorrect state,
probably because a 'sp-kill' command was issued, which sent the
process back to logon, without terminating the UNix process. The
'lppick' process appears OK. To terminate a shared printer properly,
use the TCL command 'shp-kill'.
shp-status (t
VMname Prt Port  PID Spooler command       Status
------ --- ---- ---- -------------------- ---------------
pick0    0  126 1763 'exec lp -onobanner'  OK
                     tr# Date. Time.... Description
                     3  03/03 13:14:00 End job, size=1223
                     2  03/03 13:13:55 Start job
                     1  03/03 13:14:00 Start lppick
Display trace information.
```

shpstat

examines or changes the status of shared Pick/BASIC programs that have been compiled with FlashBASIC.

The default behavior of "shpstat" is to list the status of all FlashBASIC programs currently running in shared memory. The following statistics are displayed:

"Block" is the memory block number.

"Usage" is the total number of users currently executing the program.

"Size" is the total size (in bytes) of the memory block.

"Name" is the Pick/BASIC program name. If the memory block is available, this column displays "(free)".

At the end of the listing, the following statistics are shown:

Free. This gives the total free space in shared program memory. Initially, this number should be slightly smaller than the number specified in the "pick0" configuration file. A program requires a contiguous block of memory to load and free space may become fragmented. Therefore, a given program may not load even though the total amount of free space seems sufficient.

"Max Free" is the largest contiguous free block available to the system. Since FlashBASIC programs must be loaded into a single block, this indicates the maximum size of a program that can be accommodated at the present moment.

"Used" is the total number of bytes used by all shared loaded modules. Use this to determine the amount of shared Pick/BASIC space required. Initially, the machine should be booted with a large shared memory space, then brought up to normal usage. The total shared memory may then be examined. Then, shut down the machine, resize the "basic" option in the "pick0" file

accordingly, and reboot. The basic shared memory segment should now be properly sized for your system.

If "shpstat" is invoked with the "l" option, the list is displayed interactively with the following added options:

< Scrolls back one page.

> Scrolls forward one page.

i Initializes shared program memory. This should be used only in the event of serious problems. It completely cleans out shared program memory, causing any users currently executing within it to abort.

k Kills a module. This option is used to free a module from memory if no one is currently using it, to free up some shared space. Any users executing a module which is killed will display "undefined behavior", which could involve aborting.

q Quits the program.

r Refreshes the screen to reflect the current memory status.

s Sets the sleep period in between updates. Smaller sleep times will give a more accurate view of system performance, but will also use more resources.

The program displays some useful statistics at the bottom of the screen:

"Current Page". This shows part of the program list that is currently being displayed.

"Total Number of Pages" is the number of screens the entire list takes. "<" and ">" scrolls through the pages.

"Sleep Time" is the current sleep period in seconds.

To reduce fragmentation, "sticky" programs start loading at the high end of shared memory, while "non-sticky" programs load at the bottom. Often-used programs may be compiled with the "k" option to make them "sticky".

Syntax

shpstat {(options)}

Options

i Initializes the entire shared program status area. This should be used only in case of emergencies. It causes all programs running in the shared area to abort. When used with this option, "shpstat" clears the memory and completes. No status information is displayed.

l{number}

Loops on displaying status information. This repeatedly updates the status information displayed on the screen. The default time between updates is 5 seconds. The "l" option may be followed by a number to override the default delay time between loops.

p Directs the output to the system printer, via the Spooler.

s Suppresses detail output when used without the (l option.

z "Zaps" shared program memory. This is a stronger version of the "i" option.

Example

```
shpstat (l1
This causes the status program to redisplay the current status of
shared programs every second.  This setting causes a very noticeable
drain on system performance and should only be used when tuning or if
```

```
problems occur.
shpstat (i
This completely reinitializes memory.  This may be used if  several
processes running shared code have aborted abnormally, and the shared
memory table seems corrupted.  Signs of corruption include long wait
times when loading programs, and having "shpstat" with no options
seem to hang indefinitely.
shpstat
Block  Usage   Size  Name
1      1    106k  REPORT.bp eep.help
2      1     15k  REPORT.bp eep.security
3      1      3k  REPORT.bp btree.root
4      1      8k  cmarc.library cbw.open
5      1     18k  cmarc.library cbw.place
6      1      7k  cmarc.library cbw.box
7      1     27k  REPORT.bp eep.format.doc
8         2,374k  (free)
Free:  2,374k     Max Free:  2,374k     Used:    185k
```

shutdown

initiates an orderly system shutdown sequence and should be executed before turning off or rebooting any D3 system.

Any user logged on is logged off, all process work spaces are released, all printers are stopped, all phantom jobs are logged off, and all updated memory frames are written to disk. The system is shut down.

The macro "user-shutdown" on the 'dm' account is executed, if it exists. This macro should contain any user-defined procedure to be run at shutdown time (application cleanup, network shutdown, etc...).

"Do you wish to continue (Y/N/0)?" is displayed, the "0" option is to check if this process is running on port zero before continuing. "Y" continues the shutdown, "N" stops the shutdown.

Options

f Invokes a :files at the end

Example

```
shutdown
Shutting down "pick:pick0"
Do you wish to continue (Y/N/0)?
"pick:pick0" is the name of the virtual machine.  This is only
displayed on Unix implementations.
y - continue shutdown
n - back to Pick.
0 - shutdown only if line 0
```

sleep

suspends further processing until the specified time expression is exhausted or reached.

"expression" specifies the number of seconds to suspend processing or the time to resume processing in the format hh:mm{:ss}. Hours are specified from 00 (midnight) to 23 (11:00 p.m.). Minutes and seconds are specified from 00 to 59.

Releases 6.1.0 and above support the specification of milliseconds in "expression". For example, to sleep for a 1 and a half seconds, the user can use "sleep 1.5".

Syntax

sleep expression

Example

```
sleep 600
This puts the process to sleep for ten minutes.
sleep 23:59:59
This puts the process to sleep until one second before midnight.
```

sm

see "send-message".

snake

prints a slithering reptile on the screen. Use <break> and "end" to get out of it.

The snake command is not found on all releases.

sort-list

retrieves a previously saved list, sorts it, then rewrites it to the file from which it was retrieved.

If the "file.reference" is omitted, the process defaults to "pointer-file".

If the "list.name" is omitted, the "default" list name of "%user-id" (percent sign followed by user-id) is used.

Syntax

sort-list {dict|data} {file.reference} {list.name}

Example

```
sort-list dict customers delinquent
This sorts the list of strings in the item "delinquent", found in the
dictionary of the "customers" file, then writes it back to item
"delinquent" in the dictionary of the "customers" file.
sort-list backorders
This sorts the list, "backorders", found in the "pointer-file".
```

sort-users

invokes the "list-users" command but sorts the output by "user" rather than by port number.

sortc

invokes the "sort-label" command to produce a columnar display of the item-ids in the specified file, in the order of appearance in the file.

Any valid selection criteria may be passed through to AQL for processing. The default is a 4 column display. If a numeric option is entered, then that number of columns, up to 6, will be displayed.

Syntax

sortc file.reference {sellist} {heading "heading text"} {(n}

Example

```
sortc messages with a0 > "0]" and with a0 < "2]" (n)
16 Jan 1997         Items in File messages      Page   1
1325           1079           166            1120
1161           1202           1243           1530
1080           167            1121           1162
1203           1531           1081           168
1122           1532           1204           1000
1082           169            1123           1205
1001           1083           170            1534
1124           1002           130            10
1084           171            1003           11
.
.
178            1501           1132           1543
1174           1215           1051           18
1527           1323           164            1118
1200           1241           1610           1324
1078           165            1160           1201
1242           1119
[405] 310 items listed out of 969 items.
```

sortu

invokes the "list-users" command.

soundex

demonstrates the use of the soundex conversion.

Syntax

call soundex(value)

Example

```
string = "jon"
call soundex(string)
crt string
This returns "J5".
```

speller

enables or disables the spelling checker for the current user/port only, or displays the status of the speller if no options are provided.

The automatic spelling checker is used with the Update processor.

Syntax

speller {(option)}

Options

f Toggles spelling checker off.

n Toggles spelling checker on.

s Suppresses output of status message.

speller-off

disables the automatic spelling checker for the current line.

speller-on

enables the automatic spelling checker for the current line.

The automatic spelling checker is used with the Update processor.

See the "access" function number 22 for temporarily invoking the spelling checker on one or more items.

stack

enables or disables the TCL-stacking function for the current port only, or displays the status of the stacker if no options are provided.

Syntax

stack {(options)}

Options

f TCL stacker OFF.

n TCL stacker ON.

s Suppresses output of status message.

stack-off

disables the TCL-command stacker for the current port.

Example

```
stack-off
[1309] the tcl-stack is off.
```

stack-on

enables the TCL command stacker for the current line.

Example

```
stack-on
[1308] the tcl-stack is on.
```

start-reclaim-ovf

The overflow reclamation process will be initiated on the current line or as a phantom process if b-option was specified with RECLAIM-OVF.

This process will scan all files and indexes in the system and remove used frames from the new overflow table.

Syntax

start-reclaim-ovf

start.rtc

initiates the system real time clock.

This happens automatically during the boot process, provided a real time clock is present. If one is not present, the current time is requested from the person standing closest to the console.

On D3 Unix systems, this command sets the D3 system date and time equal to the Unix system date and time, clearing up any discrepancies caused by "set-date" or "set-time" commands.

start.rtc

sets the D3 time to the same as the Unix time.

This command is normally included in the macro "system-coldstart".

start.ss

starts the Spooler and the phantom Scheduler.

On Unix implementations, this commands also spawns the Unix processes which support the D3 phantom processes, if they are not connected to the virtual machine already. The Monitor the phantoms use is controlled by the Unix environment $PATH and the Unix file '/usr/lib/pick/PICKNAME', created by the installation procedure.

On a configuration where more than one virtual machine is running, it is important to ensure that all processes on a given virtual machine run the same version of the Monitor. Normally, when the virtual machine is booted, the 'start.ss' command creates processes running the same Monitor, found on '/usr/bin', as the one currently executed by the line 0. This is ensured by executing a 'which' command, which shows the Monitor version like:

6.0.0.M0

Monitors are deemed compatible if the first two digits are similar ('6.0'). If not, then the Unix variable $PATH is scanned, and a compatible monitor is looked for on the given directories. If it fails, then 'start.ss' complains and does not start the spooler nor the phantom scheduler.

The (L) option instructs "start.ss" to use the Monitor on the current directory. For this option to work properly, the 'system-coldstart' macro should be modified as follows:

cd /usr/mydir

start.ss (l

When the (l) option is in effect, the $PATH Unix variable is not used to find a proper Monitor.

Syntax

start.ss {(option}

Options

l The phantom processes are started using the Monitor located on the current (local) directory.

Example

```
Consider a system which has two virtual machines, running the versions
Advanced Pick releases 6.0 and 6.1. The Monitor are incompatible. The
installation should proceed as follows:
1) Install first the virtual machine which will NOT be the default
Machine (Advanced Pick 6.0, for example)
2) Move the Advanced Pick 6.0 Monitor to another directory:
cd /usr/bin
mkdir 6.0
mv ap 6.0
cd /usr/lib/pick
mv pick0 pick.6.0
cd /
```

```
3) Install the Advanced Pick 6.1 (default) virtual machine.
4) Boot the Advanced Pick 6.1 (default) virtual machine:
ap -0
4) To boot the 6.0 virtual machine, do:
PATH=/usr/bin/6.0:$PATH
export PATH
ap -0 -n pick.6.0
If a regular user types 'ap', he will get to the 6.1 virtual machine. To
access the 6.0 virtual machine, the user has to specify the 6.0 Monitor,
either by changing the PATH, as before, or by explicitly giving a full file
name:
/usr/bin/ap.6.0 -n pick.6.0
```

startlog

activates the transaction logging subsystem. All items in files with a "dl" type are automatically logged to the attached magnetic media on each add, change or delete to the item.

Before invoking this command a tape device must be already attached to the current port because the transaction logger steals the tape device.

The "port.number" argument indicates the port.number on which the logger is to be started. The default is one before the last port.

"wait.time" specifies the wait time interval (in milliseconds) to flush the tape buffer when the queue is empty. The default is 30 seconds. A negative number indicates no flush.

See the "txlog" menu for a simplified interface to the transaction logging subsystem.

If the transaction logger has not been initiated since the system was booted, the system does a "double-check" prompt.

If the transaction logger has been initiated previously, the prompt is not displayed.

"startlog" attaches the tape, if available, then logs the appropriate transactions.

If the tape is needed by another process, the transaction logger can be interrupted by attaching the tape using the "u" option with "t-att". In this case, the transaction logger must be restarted with the "startlog" command, as it will not automatically restart.

Any transactions that occur while the logger is interrupted are held on disk until the transaction logger is restarted.

All transaction tapes should be saved until the next full "file-save" or incremental "file-save" is performed. When the transaction logger encounters a tape problem it sends a message to port 0.

Syntax

startlog {{port.number}{,wait.time}} {(options)}

Options

a Starts, then activates. No prompt appears.

e Enqueues all files (except "tcl-stack") for logging, even if they are NOT dl-types.

f Sets flush time when accompanied by a number.

l Enqueues dl-type files only.

o Changes status of logger. Used only with the "l" and "e" options.

u Restarts the transaction logger and asks for new backup media. Backup media can only be specified after a system restore with this option. If no backup media has been specified, startlog prompts for one.

Example

```
startlog (eo
This changes the status of the logger to enqueue writes for all files, except
"tcl-stack". "tcl-stack" items may be forced into the logging queue by editing
and filing the item(s).
startlog (lo
This changes the status of the logger to enqueue writes for dl-type files
only.
```

status-port

Invokes program "set-port".

steal-file

only valid way to move a file from another account into the current account.

This physically moves a file to the current master dictionary from another master dictionary. "steal-file" prompts for the account name and requires "sys2" privileges. It also requires that the user invoking "steal-file" is not protected against updating the file at any level, master dictionary, file level dictionary, or data file level.

After "stealing" a file, the former owner has no reference to it, not even visitation rights. He or she can get visitation rights by building a "q-pointer" to where the file moved, or by referencing it via a "file path".

Syntax

steal-file file.reference

FROM ACCOUNT: account.name

Example

```
steal-file goods
FROM ACCOUNT: victims
```

stoplog

stops the transaction logger and detaches the transaction logger tape.

Any subsequent (loggable) updates are held on the disk. When the transaction logger is restarted, these updates will be written to the transaction logger tape.

Options

k Releases the queue back to overflow. Used for recovery purposes in cases where the transaction logger consumes all available overflow space.

stopsched

halts the phantom scheduler process. No phantom processing can occur if the phantom scheduler is deactivated.

"startsched" is used to restart the scheduler.

sub

subtracts the second integer number provided from the first integer number provided and displays the result as an integer number.

Syntax

sub number number

subd

subtracts the second integer number provided from the first integer number provided and displays the result as an integer number.

Syntax

subd number number

subr.adaptor.test

returns a non-zero value in card.exists if the video adaptor specified in adaptor exists.

Syntax

call subr.adaptor.test(adaptor, card.exists)

subx

subtracts the second hexadecimal number provided from the first hexadecimal number provided and displays the result as a hexadecimal number.

Syntax

subx hex.number hex.number

Example

```
subx 400 8A
376
```

summ

is the subroutine is called from the item "summary", which is provided in all account md's and is used for "list" and "sort" commands on the md file itself.

Example

```
list md summary
```

syschk

checks a system running under Unix to detect 'abnormal' situations.

The "syschk" utility starts a phantom process which periodically checks the system is behaving 'normally'. If a system parameter goes beyond a threshold, defined by the System Administrator, a message is sent to one or more users, and, optionally, an entry is put in the "errors" file. All elements are optional. The following elements are controlled:

Unix Swap: When "syschk" is started, it always ensures that the Unix swap (or paging) space is at least equal to twice the physical memory. Then it checks periodically the swap usage does not go beyond a predefined level (90% of the total available space by default).

Total System CPU: The percentage of the CPU spent in System mode (Kernel, drivers, etc...) must stay below a predetermined level (25% by default).

Runaway Processes: Each time a sample is taken, the CPU time of the active processes is controlled. If a process has consumed more than a predefined percentage of the sampling period, a warning is issued. For example, if the sampling period is 10 minutes (600 seconds), and if the process consumed more than 5% of this time (30 seconds, which is an enormous amount of CPU), this process is probably in an abnormal tight CPU loop. "syschk" displays the Unix status ("ps") and the result of a "where", if it is a D^3 process. If a process exceeds the limit more than three times in a row, the reporting of the error stops. This is to ensure that the System

Administrator will not receive constant messages for a process which is running a large report, for example. If the process is still running after nine samples, then the reporting restarts three more times, and the reporting cycle restarts.

Unix File System Usage: The state of a predetermined list of Unix file systems is controlled, to make sure they do not get full (used over 90%). This is to prevent Unix crashes due to the filling up of critical file systems. By default, only '/' is controlled.

Overflow Usage: When the used D3 overflow exceeds a predetermined percentage of the total overflow space, a message is generated. The default level is 90%. Overflow is reported only once a day, at the first sample taken after noon.

Basic Usage: This monitors the usage of the FlashBASIC "basic" area. When the used "basic" space exceeds a predetermined percentage of the total "basic" space, a message is generated. The default level is 90%. To examine the "basic" space in more detail, use the "shpstat" command.

This command must be run on the "dm" account. Only one instance of a phantom process running "syschk" is supported at any given time.

The form "syschk start" starts the syschk command running as a phantom with the same argument as the last time it was started. If necessary, the process is stopped before it is restarted. If the process was never started, a set of defaults is applied.

The form "syschk edit" enters the Update processor to allow editing the arguments used by the "start" command.

The form "syschk now" instructs the "syschk" phantom process to take a sample immediately, and send the anomaly messages, if any, to the the terminal requesting the sample, instead of the normal notify mechanism.

The form "syschk stop" stops the syschk command running as a phantom.

The form "syschk" without any argument, simply reports whether syschk is currently running and displays when it was started, the parameters and some of the current system parameter values.

With one or more arguments, "syschk" creates a phantom process and returns immediately to TCL. Arguments can be specified in any order.

"keyword{=value}" specifies which system parameters to control:

sampling=[sec | hh:mm:ss]

Specifies the sampling delay. The delay can be expressed either in seconds of in hh:mm:ss format. If "sampling" is not specified, the default is 30 minutes.

notify=[user{,..},!n{,..}, /dev/ ttyXX{,...} ,* | OFF]

Specifies the list of D3 users, of D3 port number or Unix devices to notify in case of abnormal situation. The users can be specified as a list of explicit D3 user ids, D3 port numbers, in decimal, prefixed by an exclamation mark (!), Unix devices, prefixed by a slash (/), or any combination. If '*' is used, all users logged on the system are notified. If a period ('.') is used, it is replaced by the tty name. If 'OFF' is specified, notification is disabled. If "notify" is not specified, or if the specified users are not logged on at the time the anomaly occurs, a message is sent to "dm" or "sysprog". If a Unix device is specified, it must exist and be writable when "syschk" is started.

syscpu{= percentage {%}}

Specifies the maximum percentage of total CPU usage Unix is allowed to spend in System mode. If 'percentage' is not specified, the default is 25%.

proccpu{= percentage {%}}

Specifies the maximum percentage of CPU a process is allowed to take. If 'percentage' is not specified, the default is 25%. This trigger point may be a little difficult to evaluate. For example, on a system with only one active user running a FlashBASIC cpu intensive program, the process will, naturally, takes 100% of the CPU, since there is no other running process. To avoid false alarms, select a sampling period large enough. It is probably unusual to have a process doing 100% of CPU for 15 minutes.

swapusg{= percentage {%}}

Specify the acceptable swap usage at any given time. 'percentage' is the amount of swap actually used. If 'percentage' is not specified, a swap usage above 90% of the total swap will be considered abnormal.

ovfusg{= percentage {%}}

Specify the acceptable overflow usage at any given time. 'percentage' is the amount of overflow actually used. If 'percentage' is not specified, an overflow usage above 90% of the total

D3 space will be considered abnormal. Errors are reported only once a day.

basicusg{= percentage {%}}

Specify the acceptable 'basic' usage at any given time. 'percentage' is the amount of 'basic' space actually used. If 'percentage' is not specified, a 'basic' usage above 90% of the total 'basic' space will be considered abnormal.

diskusg= filesystem{,filesystem,..}

Specify the list of Unix file system which should never get full (over 90%). The list should always include '/'. Depending on the system, '/usr', '/tmp' might have to be included. Some Unix system, will not be able to boot with a full '/'. If not specified, '/' is the only Unix file system to be checked.

log

Log messages in the "errors" file. This keyword is equivalent to the (L) option.

nolog

Do not log messages in the "errors" file. This keyword is equivalent to not having the (L) option. It supersedes the (L) option.

When started as a phantom, "syschk" runs for ever, until the system is shutdown. To stop it, use "syschk stop".

Syntax

syschk keyword{=value} {...} {(options}

syschk edit

syschk now

syschk start

syschk stop

syschk

Options

F Start syschk in foreground on the current process, as opposed to a phantom process. With this option, the only way to stop the process is to do a break / end, or a logoff.

L Log a short summary of the error messages to the "errors" file. The initial swap space control is always logged. Can be specified by using the "log" keyword.

Q Quiet. Suppresses some user messages ("started", "stopped")

V Verbose. This option can be used if "syschk" is run in foreground, instead of a phantom.

Example

```
syschk sampling=00:30:00 syscpu=10%
         notify=/dev/tty0,bob,!0
         swapusg=60% proccpu=10% log
  Start a phantom process to check the system every 30 minutes for a CPU
system usage above 10%, a swap usage above 60% and a process runaway limit of
10%. In case of anomaly, a message is sent to the Unix terminal '/dev/tty0',
the D3 user 'bob' if he is logged on, and to the line 0 whether it is logged
on or not, and a short message is logged in the 'errors' file. The other
parameters are left to their default values.
syschk
  Check whether syschk is running. This would display, for example:
    syschk is running on port 132
    Started on 03/11/94 at 08:20:21
    Current running parameters:
      Sampling period           00:30:00
      Notify list               /dev/tty0  bob  !0
      Maximum system CPU %      10
      Maximum CPU % per process 25
      Maximum % of swap         60
      Maximum % of overflow     90
      Unix file systems         /
      Log messages (0=no;1=yes) 1
    Current System Status:
      User CPU usage            3%
      System CPU usage         11%
      Waiting for IO           82%
      Idle CPU                  4%
      Total swap space        128 Mb
      Used swap space          76 Mb (59%)
syschk stop (q
  Stop the phantom running "syschk" as a background process, suppressing the
message "stopped". This command could be included in the "user-shutdown"
macro.
syschk start
  Restart the "syschk" phantom with the same parameters.
syschk edit
  Edit the syschk command line to change the arguments. Use the Update
processor command to edit the command line.
```

sysid

displays system configuration, options, and features.

The "sysid" program displays various information about the current platform including the host system, processor type, and implementation.

Any host O/S options available from D3 are shown under "System Options".

Finally, "sysid" displays a list of D3 features which may be either "Available" or "Not Available". The activation code received when the system is initially installed determines the availability of these features.

For a description of specific features available for a particular release, please refer to the "READ ME FIRST" document included with your documentation.

Syntax

sysid

system-coldstart

executes system initialization commands.

Invoked after the diagnostic and setup portion of the D3 boot process completes.

The commands executed by this macro vary from platform to platform, so its exact behavior can only be determined by looking at the macro, which usually exists in the "md" of the "dm" account.

Some of the commands typically executed by "system-coldstart" are attached as related subjects.

Any site-specific commands to be executed at boot time should be placed in the "user-coldstart" macro, which is automatically called after "system-coldstart" completes. See "user-coldstart" for more information.

Example

```
n
run dm,bp, setpib0
term-type (s
run dm,bp, start.rtc
run dm,bp, start.ss
initovf
coldstart.log
run dm,bp, cleanpibs
:reset-async
set-abs dm,abs, (s
maxusers (mql
verify.system
run dm,bp, start.printers
coldstart
off
```

t-copy

Copies data from one device to another.

file.count specifies number of file segments to copy. Default is 1 file segment. For instance, a file in "t-dump" format is written on tape as 1 file segment. A file-save or account-save on tape is written as 2 file segments, plus 1 file segment for each account. Therefore, to copy up to the first account of a save, you would specify a file.count of 3; to copy the first two accounts, specify a file.count of 4; etc.

Syntax

t-copy source.device dest.device {file.count}

Example

```
:who
15 dm dm
:list-device
Tape        Status           17 Jan 1996  11:46:41
```

```
 #   Type             Density           Owner   Device Name
-------------------------------------------------------------
 0 | Floppy         | 3 1/2" 1.44M    |  15*+ | /dev/rpdsk/2
 1 | Floppy         | 5 1/4" 1.2M     |       | /dev/rpdsk/3
 2 | Quarter Inch   | High            |  15*  | /dev/rmt/0n
:t-copy 0 2 3
This will copy the contents of device 0 to device 2, up to the 3rd EOF mark.
If device 0 contains data in "save" format, this will copy the first account
on tape.
:t-copy 0 2
This will copy up to the 1st EOF mark (default file.count is 1). In this case,
if device 0 contains data in "t-dump" format, this will copy the t-dump from
device 0 to device 2.
```

t-status

refers to the most recent "t-att" command, and displays the current settings of the peripheral storage device characteristics consisting of its type, density and drive number.

The media does not have to be attached to the current process in order to use this command.

ta-off

invokes the "type-ahead" verb and passes it an "f" option, disabling the "type-ahead" feature on the current port.

ta-on

invokes the "type-ahead" verb and passes it an "n" option, enabling the "type-ahead" feature on the current port.

tabs

displays tab stops previously set via the editor (ed or edit) process, or assigns new tabstop positions for input (i) or output (o).

Output tabstop positions are useful only on printing terminals that have a physical tab facility and should not be defined on normal crt's.

Tabstops may be set up automatically for each user by including the "tabs" command in the users logon macro in the "users" file .

Syntax

tabs

tabs i {s}

tabs o {s}

tabs {i} {tabstop{,tabstop...}

tabs {o} {tabstop{,tabstop...}

Options

i Disable input tabstops.

o Disable output tabstops.

s indicate the previous tabstop settings are to be used.

tandem

links the current port to another process. Any input or output to either screen affects both screens.

The port initiating the "tandem" command is the "master". The process being attached is the "slave" or "target".

If the slave process is a "phantom" process, the output from the phantom is redirected to the master.

The target device (the one attached to the slave) must be available on the given port. It may not be already attached to another process.

A "hotkey" sequence disconnects the link. The default "hotkey" sequence is <escape>x. If an "esc-data" has not been issued previously, the process issues it automatically, to prevent getting locked out from being able to suspend "tandem".

See the "trap" command for handling output message handling. (e.g. turning off the "TANDEM STARTED" message.)

Syntax

tandem port.number {(options}

Options

d Designates the sequence of (up to four) characters to terminate "tandem" mode. Each character is provided in its hexadecimal equivalent of its ASCII value. Each character is separated by a comma. For example, "d/41,42,43,44", indicates that an "ABCD" will disconnect the link. The first two characters of the termination sequence must be different. For instance, it will NOT accept "d/2b,2b,2b" (+++), but it will accept "d/2a,2b,2b" (*++).

f Turns off the ability to tandem on the current port. This keeps others from being able to tandem the current port. Specifying "port.number" allows turning off tandem on another port, if the user has a SYS2 privilege.

n Turns on the ability to tandem on the current port. This gives others the ability to tandem the current port. May NOT be used with the "u" option. Specifying "port.number" allows turning on tandem on another port, if the user has a SYS2 privilege.

s Suppresses the message when the process terminates. If this is used with the "d" option, the "s" option must precede the "d" option.

u Unconditionally turns off tandem when used with the "f" option.

x Terminates tandem mode on target device number.

y Set the 'Notify' option. When tandem is started or terminated on the process, the command specified for the 'tdmon' or 'tdmof' signals in the TCL "trap" command.

Example

```
tandem 13
Attaches the current process to port 13.
tandem 33 (d/2a,2b,2b
Attaches to port 3 and sets the hotkey sequence to "*++".
tandem (n
Enables ability to be "tandemed" on the current port.
tandem 42 (uf
Unconditionally disables "tandem" capability on port 42.
tandem 13 (x
Terminates tandem on port 13.
tcl 42 bob tandem (n
tandem 42
The "tcl" command can be used to enable tandem on another port.
```

tape-socket

defines a tape system across a network. This section is a detailed reference of the "tape-socket" TCL command. See the section "tape socket, General" for an introduction to the fundamental notions necessary to the set up and utilization of this system.

The "tape-socket" TCL command is used to create the input or output server and control their activity. A tape-socket log file "ts.log" is created the first time the command is started to record the process activity, and a permanent log "ts.log,log" keeps all messages.

This command can be executed only on the 'dm' or 'sysprog' account. It requires a SYS2 privilege.

Commands :

"cmd" is one of the following, where allowed abbreviation are shown between parenthesis:

check (ch) Check remote Server.

drain (dr) Drain (empty) pipe.

query (q) Query a server.

setarg () Change argument.

setup () Setup Server parameters.

show () Show Server parameters.

shutdown () Stop BOTH Servers.

start (ss) Start a server by name..

startsend (ss) Start the output server.

startreceive (sr) Start the input server.

status (sta) List the server status.

stop (sto) Stop a server.

stopall (stopa) Stop all servers.

traceon (tron) Turn traces on.

traceoff (troff) Turn traces off.

Without any argument, a menu is displayed, showing the most useful options to manage the default server. The menu is described later in this section.

Arguments :

The arguments are specified by a series of statements "keyword=value". Arguments can be specified in any order. If "value" is a question mark (?), the user is asked to enter a value for the specified keyword, at which point a question mark (?) will display some help and 'q' will quit. If "value" is a dot (.), the value used last time is substituted. These forms are used in macros.

"keyword" is one of the following:

callrep Number of attempts the output server will do to call the input server. After this number has been reached, and if the input server does not respond, the output server will terminate. '0' means infinite number of attempts to call. The default value is 0. A 5 second delay occurs between each attempt.

cmd Command to be executed on the remote system. See the list of valid commands below in the section "Check Command". This form is used to check the communication link. See the example section.

host Network name of the host where the input server resides. This argument is required only for the output server to be able to reach the input server. The host name must be defined in the '/etc/hosts' file in the sender.

ndisc Maximum number of network disconnection(s) either server will tolerate. A value of 0 means infinite number. When a network disconnection occurs, the output server will try to call the input server again. The default value is 0.

notify Name of one or more D3 user(s) to whom error messages are sent. The notify list may have one of the following forms:

OFF : Disable the notification.

user : D3 user name.

! line : D3 line number

* : All D3 users.

exec cmd : Run 'cmd'.

More than one user can be specified, by separating each user by a comma (eg notify=bob,sam). If the form 'cmd' is used, the entire notification list must be enclosed between quotes, because of the space which follows the 'exec' key word. For example, notify="dm,!0,exec run bp send-mail". The text of the error message is added after 'cmd'. The users must exist in the "dm,users," file.

pib D3 port number of the server. This option can be used as an argument to the "query" and "stop" commands as a quick alternative to the form "pipe=device".

pipe Define the Unix pipe. A full path name must be provided (eg /dev/tapein). The pipe MUST exist and have appropriate read/write permissions. A raw device device name will be accepted as well.

poll Defines the period with which the transaction logger is tested. "poll" is expressed in seconds, or in HH:MM:SS. A value of 0 disables the transaction log test polling. See the section "tape socket, General" for a detailed description of this feature.

port Socket port number in decimal of the input server on the receiver's side. The socket port number is a convention between the input and outputserver. It is a decimal number between 1024 and 32767.

prot Network protocol. The following protocols are supported:

"inet" : Internet.

server Name of the server. If left empty, the default server is used. The Server name can be any string.

servertype Type of the server. This option is required when using the "setup" command to set up the running parameters of a Server. "servertype" is either 'IN' for the Input Server, or 'OUT' for the Output Server.

trace Maximum number of traces either server will keep in the log file. The default value is 4. Without the (V) (verbose) option, only major events are recorded. The (V) option records ALL data. When this number is exceeded, the oldest trace entries are discarded.

txlog Specify whether the Server is to be linked automatically to the Transaction Logger. "txlog" is either 'ON', or 'OFF'. See the section "tape socket, General" for a detailed description of this feature.

txopt Specify whether the Transaction Logger should log updates to all files, or only to files with the (DL) attribute. This option is valid only for the Output Server, if 'txlog=ON'. 'txopt' is either 'DL' or 'ALL'.

txpriod Specify the period, in seconds, with which the transaction log queue is emptied. A value of 0 specifies the default value.

Status Command :

The "status" commands lists the following information:

id Port number of the server, in decimal.

T Type of the server:

s : Send (output) server

r : Receive (input) server

Pipe Pipe name

Host Host name (input server only).

Port Socket port number in decimal

S Status of the server;

E : Error

C : Completed

L : Logoff

Q : Queued

R : Running

S : Aborted

Time Time of the last trace entry

Date Date of the last trace entry

Message Trace message. Each trace is prefixed by the current message number in decimal. The servers exchange message number information to make sure no data loss occurs. The following are the main messages:

ACK timeout The input server did not respond to a message. The output server retries.

Accept err=n accept() system call error. n=errno.

Bad header 'X' A network message had an incorrectly formatted header. 'X' is a hex dump of the header.

Bad msg num 'X' A network header contained an incorrect message number 'X'

Bind err=n Input server could not 'bind' with the specified port. n=errno.

Broken pipe Input server tried to write into pipe, but associated tape process detached from it.

Call err=n Output server can not call input server. n=errno.

Cannot find jobid Server could not find its job id in the phantom log files 'jobs'.

Check ERR The input Server responded to a 'check' command but found an error.

Check OK The Input Server responded to a 'check' command.

Clear pipe Clear the pipe, if NO (C) option.

Connect accept The input server accepted an incoming connection.

Connect err=n Output server failed to establish connection. n=errno.

Hread trunc=n Network msg header truncated. n=msg length.

Listen err=n listen() system call error. n=errno.

Local query Server responded to a 'tape-status query' command.

Lost msg=n Input server detected a message loss. n is the message received. Messages from the current message up to n are lost.

Lost POLL n A Transaction Log test item has not been received by the Input Server.

Malloc err=n malloc() system call error. n=errno. Server could not obtain memory for buffers.

Nread err=n Network read error. n=errno.

Nread n xxxxx Network read. n=msg length, 'xxxxx' trace.

Nwrit err=n Network write error. n=errno.

Nwrit n xxxxx Network write. n=msg length, 'xxxxx' trace.

Nwrit trunc=n Network write truncated. n=msg length.

Open pipe Wait for pipe open.

Pclear err=n Error while attempting to purge the pipe. n=errno.

PEOF err=n Input server failed to write a EOF marker in the pipe.

Popen err=n open() pipe error. n=errno.

Pread err=n Pipe read error. n=errno.

Pread n xxxxx Pipe read. n=msg length, 'xxxxx' trace.

Pread trunc=n Pipe read truncated. n=msg length.

Pwrit err=n Pipe write error. n=errno.

Pwrit n xxxxx Pipe write. n=msg length, 'xxxxx' trace.

Pwrit trunc=n Pipe write truncated. n=msg length.

Re-sync n Input server receives a SYNC message from output server. n=new starting msg number.

Send resync Output server was asked to send a sync message.

Remote shtdwn The Input Serer receives a request to stop.

Running Server is running. This status is stored every 5 mn on a busy system. This message is not stored in the permanent log.

Sent POLL A Transaction log test item has been sent to the Input Server.

Seq error n An message was received twice. n=old message number. The msg is discarded.

Started Server is started.

Stop on req Server stopped due to a 'tape-socket stop' command.

Stop refused-Txlog up A request to stop the input server was refused because the transaction logger is still active. Repeat the stop request.

Stopped Server is stopped due a spontaneous termination. The cause of the termination is indicated in a previous trace entry.

Socket err=n socket() system call error. n=errno.

TLOG not off The Input Server failed to abort the transaction restore process following a request to stop.

TLOG Restarted The Input Server restarted the transaction restore process following a request from the remote Output Server.

TLOG Terminated The Input Server aborted the transaction restore process following a request to stop.

Too many disc Server detected disconnects in excess of 'ndisc' and terminated.

Too many errors Server detected too many errors

Total on dd/mm Total number of kilobytes transferred since the first time a message was logged the morning of the specified day.

Txlog OK Transaction log test item has been received by Input Server. This message is not stored in the permanent log.

Unexpected msg 'X' The message 'X' on the network is not a 'tape-socket' msg.

Unknown cmd 'X' Server received an unknown command 'X'

Wait ack Output server is waiting for an ACK.

Wait connect Input server waits for incoming call.

Query Command Result :

The "query" command returns the running parameters, and the following information:

Next poll time Time, if activated, of the next Transaction Log test polling.

Total Data transferred Total number of kilobytes transferred that day. This number is approximate.

Last msgnum Last message number, at the time of the last query and the average number of messages per second since the last query. This count does not include the protocol message.

msgin Total number of messages input to the server. For an input server, this is the number of network messages, including the protocol messages. For an output server, this is the number of tape blocks read from the pipe.

msgout Total number of messages output from the server. For an input server, this is the number of tape blocks written into the pipe. For an output server, this is the number of messages sent on the newtork, including the protocol messages.

curmsg Current message number. During normal operations, the values of 'curmsg' for both servers should be equal. Should they diverge, the input server will log the incident and re-synchronize.

Status Short description of the server current status:

Open pipe

The server is waiting for the associated tape process to open the pipe. This is the quiescent state of both servers when no tape process has opened the associated pipe.

Reading network

The input server is waiting for incoming data from the network. This is the quiescent state of the input server.

Reading pipe

The output server is waiting for data from the associated tape process. This is the quiescent state of the output server.

Wait 1st call

The output server is waiting for the answer to its first call to establish connection. Wait incoming connect. The input server is waiting for an incoming call.

Wait subsequent call

The output server is waiting for the answer to repeated call(s) to establish connection. This is an indication of failure to establish communication with the input server.

Stopped

The server is stopped.

Drain Command :

The "drain" command empties the specified pipe. This command is implicitly executed when starting a server, unless the (C) option is specified or if the Server is linked to the Transaction Logger. Emptying the pipe is sometimes necessary to re-synchronize the processes. The data which is drained out is saved in the file "ts.log,backup" for later processing.

Check Command :

The "check" command is used to send requests from the local Output Server to the remote Input Server. The main purpose is to check the communication link and to do some remote control of the input server. "chk.com" is the command to be executed by the remote input Server. If there is more than one word, it must be enclosed in quotes (eg cmd='exec where 0 (h'):

exec tcl.com Execute the TCL command 'tcl.com' on the remote. This command should a simple one, like 'who' or 'time'. Only the first line of the result is returned.

msgnum Returns the last message number received by the Input Server.

query Query the remote Input Server for its status.

shtdwn Shutdown Input Server. The Input server terminates immediately. If it is linked to the Transaction Log Sub-System, the transaction restore process is aborted and sent back the tape-socket menu.

test -f fn Test if the file 'fn' exists. If so, a string "1 File 'fn' exists" is sent back. Else a string "0 File 'fn' missing" is sent back.

test -r{d} fn id Test if the file 'fn' exists, and if the item 'id' is in this file. If so, a string "1 <item body>" is sent back. Else a string "0 File 'fn' missing" or "0 Item 'id' missing" is sent back. If the 'd' flag is present, the item is deleted.

tlchk Check the Transaction restore on the backup system. Valid only if the Input Server is linked to the Transaction Log Sub-System. This command makes sure the process doing the transaction restore is in a 'normal' state (attached to the tape, not waiting for input, not in the debugger, etc..), and will make some attempt at correcting the problem (answering to the prompts). This command is executed automatically by the Output Server when a transaction log polling test fails, and by the input Server if it appears that the transaction restore is not emptying the pipe.

tlstrt Restart the Transaction restore on the backup system. Valid only if the Input Server is linked to the Transaction Log Sub-System.

tron n Turn traces on on the remote. 'n' is the number of traces.

troff Turn traces off on the remote.

System Administrator Messages :

This section lists the messages that may be sent to the D3 users designated in the "notify" parameter, the likely cause and the possible actions to correct the situation:

Communication to host is re-established.

The Output Server succeeded in re-establishing the connection. This message is issued only once.

Input Server stopped due to an error.

The Server encountered a fatal error. Use the "Status" command on the receiver's side to find the last error. This is likely to be due to a serious condition like, the Unix pipe does not exist, or does not have the proper access rights, the TCP port number is already in use, etc...

Network Back On Line.

After a network error was detected, the communication was re-established. This message is sent only once, to indicate the end of a problem.

Network Error. Check Error Log.

A network error was detected. Check the error log to see the cause of the failure. Check the Input Server is up. Use the Unix command "ping <host>" to make sure the remote host is reachable. This message is sent only once, the first time an error is detected.

Network is disconnected. Re-trying to call host

The Output Server failed to establish or re-establish the connection after three attempts. This message is issued only once. Check the error log to see the cause of the failure. Check the Input Server is up. Use the Unix command "ping <host>" to make sure the remote host is reachable.

Output Server stopped due to an error.

The Server encountered a fatal error. Use the "Status" command to find the last error.

Transaction logger problem. Test item not sent.

The Output Server found that none of the Transaction Log Test items reached the remote system. Make sure the communication is up and that the enqueuing of transactions is active. If the transaction log queue is large, it may be because the test items are still in it. If the queue is empty or small, check the transaction logger with the TCL command "txlog". This message is likely to be an indication of a serious problem. This message is issued at every failed attempt

until a test succeeds. If this becomes a nuisance, change the polling period, using the menu option "Change TX LOG Polling", in the "Special Operations" sub-menu.

Transaction logger problem. Lost poll.

The Output Server found that one of the Transaction Log Test items did not reach the remote system, even though some test items made it on the other system. This is likely to be a temporary condition, due to a large queue. This message is issued at every failed attempt until a test succeeds. If this becomes a nuisance, change the polling period, using the menu option "Change TX LOG Polling", in the "Special Operations" sub-menu.

Transaction log back on line.

The Transaction Log Test polling resumed its normal operations after an incident was discovered in a previous test. This message is issued only once to indicate the end of the problem.

Transaction logger not attached to tape

This message indicates that the transaction logger was detached from the tape without the Output Server being notified. This was probably caused by using the "txlog" menu instead of the "tape-socket" stop command or menu option. Stop the Output Server and re-start it to correct this situation.

Transaction Restore not Re-Started on Receiver.

This message indicates that the Input Server failed to restart the transaction restore. The process doing the Transaction Restore on the receiver is probably waiting for a user prompt due, most likely, to the stopping of the transaction logger by a mean other than the "tape-socket" menu or command. If the tape has been detached manually from the transaction logger, which can be shown by the "txlog" command, it might be possible to restart the transaction logger from the MASTER system, by selecting the option "Send Command to Remote" in the sub-menu "Special Operations", and send the command "tlstrt" which instructs the Input Server to restart the transaction restore. If this remote command succeeds, then reattach the tape to the transaction logger on the sender side by using the "txlog" menu. If this fails, the System Administrator must act on the receiver's side, by answering whatever question is asked on the transaction restore process (eg "Mount Next reel", or "end" if an abort occurred, etc...). Then re-start the Input Server. The pipe might have to be drained on the receiver's side, using the "Drain Pipe" option in the "Special Operations" sub-menu.

Transaction Restore problem. diagnostics

This message indicates that the Transaction Restore, on the receiving side, is in an abnormal state. This message is issued by the Output Server when, after having detected that a transaction polling test failed, an attempt at correcting the situation failed. This situation will probably require the System Administrator to intervene on the backup system (or use the remote command execution to act on the transaction restore).

Default Server Menus :

Without any option, a menu is displayed. This menu allows operations on the default Server. All optional arguments are set to their defaults, when using the menu. This should suit most configurations where there is only one server, either input or output. When an argument is missing, the user is prompted for it.

Network Tape (1.5.0)

1) List Status 4) Stop Server 7) Show Server

2) Query Server 5) Special Operations 8) Shutdown

3) Start Server 6) Setup Server 9) Other Servers

The "Special Operations" sub-menu allows performing seldom used operations, to test a new installation, for instance.

Network Tape (1.5.0) : Special Operations

1) Turn Trace ON on Server 7) Start Server with NO clear

2) Turn Trace OFF on Server 8) List Permanent Log

3) Change TX LOG polling 9) Clear Permanent Log

4) Change notify user 10) Test transaction Log

5) Drain Pipe 11) List pipes

6) Send Command to Remote

Each option in the menu has some on-line help. See the example below for how to use the menu.

Syntax

tape-socket {cmd {keyword=value} {(options}}

Options

C Do NOT clear the pipes before starting a server. This option should be used only if data in the pipe should be preserved. This situation normally arises only when the server is stopped in the middle of a communication. Extreme care should be taken when using this option.

Q Quiet. Suppress some user messages and confirmation prompts when stopping servers and draining pipes.

R Show only 'Running' servers in the "status" command

S Suppress the synchronization of clocks at startup time.

V Verbose. Record all events in the log file.

Example

```
Hot Backup Setup Example :
Assume a TCP/IP configuration over Ethernet, between two systems. The sender
is the production system 'PROD' and the receiver a back up system 'BACKUP'.
The two systems are to be setup in a 'hot backup' configuration. Both systems
are defined in the Unix '/etc/hosts/ file.
'BACKUP' setup:
  - Create a pipe (from Unix):
    mknod /dev/tapein p
    chmod a+rw /dev/tapein
  - Declare the pipe as a pseudo tape in the D3 configuration file of the
receiver, by inserting the statement:
    tape /dev/tapein 500 c lx
  - Boot the D3 virtual machine on the 'BACKUP' system. You MUST have at least
TWO terminals connected to the 'BACKUP' systems. One is going to be used for
the transaction restore, and the second one will be used, temporarily, for
system administration.
  - Select an unused TCP port number. The list of currently used port numbers
can be found, usually, in the Unix file "/etc/services", or by using the
"netstat -a" command. The number can be anything (>1024 and <32767), as long
as both servers agree on it. This example uses 3000.
  - Set the default Server by selecting the option 'Setup Server' in the menu.
Enter the following:
```

```
    Server type : in
    TCP/IP port number : 3000
    Protocol : inet
    Unix pipe name : /dev/tapein
    D3 User to notify : bob
    Start transaction logger : on
  - Start the input server on 'BACKUP' by selecting the option "Start Server",
or by typing, at TCL:
    tape-socket start
'PROD' setup:
  - Create a pipe (from Unix):
    mknod /dev/tapeout p
    chmod a+rw /dev/tapeout
  - Declare the pipe as a pseudo tape in the D3 configuration file of sender,
by inserting the statement:
    tape /dev/tapeout 500 c lx
  - Boot the D3 virtual machine on the 'PROD' system.
  - Set the default Server by selecting the option 'Setup Server' in the menu.
Enter the following:
    Server type : out
    Remote HOST name : BACKUP
    TCP/IP port number : 3000
    Protocol : inet
    Unix pipe name : /dev/tapeout
    D3 User to notify : bob
    Transaction Log test polling : 00:10:00
    Start transaction logger : on
    Log (DL) files or ALL : dl
    Transaction log queue period : 3
  - Start the output server on 'PROD' by selecting the option "Start Server",
or by typing, at TCL:
    tape-socket start
On both systems, list the server activity by selecting the option "List
Status". Both servers should show a status 'Started'. Query the servers by
selecting option "Query Server". The Output Server should show a status
"Reading Pipe" and the Input Server should show a status "reading Network". If
not, refer to the section 'Troubleshooting' below. To check the remote Input
Server is active, select the "Special Operation" sub-menu, option "Send
command to Remote" and, in answer to the question "cmd=", type 'query', or,
from TCL, use the 'check' command:
  tape-socket check cmd=query
The input server should respond with a short message. The 'check' command can
also be used to do some short commands on the remote input server. For
instance, to set the date on the remote Input Server (note the usage of quotes
around the command):
  tape-socket check cmd="set-date 10/06/93"
Stopping a Server :
When attempting to stop a server, a warning is issued if the pipe served by
this process is not empty. Unless absolutely required, it is not recommended
to stop the server while data is in the pipe. In addition to this control,
stopping the Input Server while the Output Server has not been stopped, will
also issue a warning if the server has been linked to the transaction restore.
Detaching the Tape on the Master system :
On releases prior to 6.1, to be able to use the tape on the master system, it
is necessary to stop the Transaction Logger. This can be done without any
human interaction on the backup system. Do NOT use the "txlog" menu to do
this. Select the tape-socket menu option "Stop Server". This command will
detach the tape from the transaction logger, make the remote machine aware of
the fact that the transaction logger is stopping temporarily, and stop the
Output Server.
To re-attach the tape to the Transaction Logger and restart the data transfer,
again,  do NOT use the "txlog" menu to do this. Select the tape-socket menu
```

```
option "Start Server". The option will restart the Output Server and the
Transaction logger process.
Detaching the Tape on the Backup System :
This operation should also be done from the MASTER system, to make sure all
operations are done in the proper order. It involves stopping the transaction
logger on the master system, stopping the transaction restore on the backup
system, and then stopping the Servers. All this is accomplished by the
'Shutdown" menu option on the MASTER system. After the remote shutdown has
completed, the process which was doing the transaction restore is sent back to
the tape-socket menu, after having detached the tape on the backup system. The
tape can then be used.
To restart the system, first restart the Input Server. Remember that by
starting the Input Server linked to the Transaction Restore, the process on
which the "Start Server" menu option is run, BECOMES the process which does
the transaction restore, thus not freeing the terminal. Then restart the
Output Server on the Main system. Note that if it is attempted to stop the
Input Server without first stopping the Output Server, the Input Server will
complain. If the stop command is repeated, then the Input Server will stop,
even if the Output Server is still running.
Other usages :
When not linked to the Transaction Logger Sub-System, the tape-socket Servers
can be used for a variety of functions. Be sure the Server are setup so that
they are NOT linked to the Transaction Logger by using the "Setup Server" menu
option.
- The 'tape-socket' command can also be used to provide remote access to a
floppy. To achieve this, on the sender's side, start the output server using
the floppy device name instead of a pipe. The output server will start reading
from the floppy and write its content over the network into the pipe on the
receiver's side, which can then do a T-LOAD, for example. After the floppy has
been sent, stop both servers. It will not handle multiple volumes.
- There is no obligation that the tape process and server be in the same D3
virtual machine. One application is to do full file restores across the
network. To implement this, the receiver's system should have a small D3
virtual machine, in which the input server is running. The data it receives
from the network (the file save), is written into the pipe which is read by
the real D3 virtual machine doing its file load.
```

tcl

suspends the process on the specified port, pushes a level on that port, then executes the TCL command.

To suppress messages on the current port, see the "tcls" command.

Syntax

tcl port.number user-id TCL.command {(option)}

Example

```
tcl 33 dm count md
Executes the "cound md" command on port 33, provided that the current user-id
is "dm".  If the current user-id is not specified, or is invalid, the command
does not take place.  The results of the "count" command appear on both port
33, and on the port originating the "tcl" command.
```

tcl edit commands

uses the Update processor to enter and edit TCL commands.

The UP commands listed under "see also" are valid when editing items on the TCL command line.

tcl-hdr

enables or disables the display of TCL commands on all output directed to the Spooler, or displays the current header status if no options are provided.

The default setting is "active", or "on". This means that every report that is directed to the Spooler will automatically display the TCL command used to generate the output on the first page of the output, unless otherwise suppressed with a "tcl-supp" modifier.

Syntax

tcl-hdr {(options}

Options

f Toggles OFF. Suppresses the default TCL header output.

n Toggles ON. Outputs the default TCL header on Spooler listings.

s Suppresses output of status message.

Example

```
tcl-hdr (f
[1305] TCL header output is suppressed.
tcl-hdr (n
[1304] TCL header output is active.
```

tcl-hdr-off

invokes the "tcl-hdr" command and passes it an "f" option, disabling the display of TCL commands on all output directed to the Spooler.

Example

```
tcl-hdr-off
[1305] TCL header output is suppressed.
```

tcl-hdr-on

invokes the "tcl-hdr" command and passes it an "n" option, enabling the display of TCL commands on all output directed to the spooler.

Example

```
tcl-hdr-on
[1304] TCL header output is active.
```

tcl-prompt

sets the TCL prompt.

This sets a prefix to the TCL prompt. To set the actual prompt character, append a comma to the end of the prompt followed by the new character. To set the select list prompt character, add another comma followed by the new select list prompt character.

This verb sets the @$PROMPT TCL shell variable.

Syntax

tcl-prompt {prompt} {(options}

Options

f Resets the prompt setting to the default.

tcl.stack.desc

used by the "list-errors" command to display the last 3 TCL entries stored in the errors file.

tcls

suspends the process on the specified port, pushes a level on that port, then executes the TCL command. This is different from the "tcl" command in that "tcls" suppresses commands on the current port. Available only on releases 7.0.5 and higher.

Syntax

tcls port.number user-id TCL.command

Example

```
tcls 33 dm count md
Executes the "cound md" command on port 33, provided that the current
user-id is "dm".  If the current user-id is not specified, or is
invalid, the command does not take place.  The results of the "count"
command appear on port 33, but not on the port originating the "tcls"
command.
```

term

changes or displays the current terminal and printer output characteristics for the current line.

Any parameter may be left intact by entering a null (two consecutive commas). The only exception to this rule is the "tt" (term type) parameter which defines the terminal name. The system recognizes terminal names, so the name may be entered first or last.

The term type parameter may be specified without any other arguments, or in any position of the argument list, since it is the only non-numeric argument.

The arguments allowed are as follows:

"tw" is the terminal width, or the number of print positions per line on the terminal. It must be between 10 and 140. Although most terminals are capable of printing 80 characters per line, the typical setting is 79 to avoid problems with carriage return/line feeds when displaying at the 80th position.

"td" is the terminal depth, in lines.

"ls" is the "line skip". This is the number of blank lines at the bottom of the screen. The value in "td" is added to the value in "ls" and must equal the actual number of lines on the terminal.

"ld" is for "linefeed delays". This is the number of null characters to output after each linefeed. This is typically set to 0 (zero).

"fd" is for "formfeed delays". This is the number of null characters to output after a top-of-form is executed. The "fd" parameter also determines when to eject a page (or clear the screen) between output pages. The following settings are valid:

0 Suppresses page ejects between output pages on both the terminal and the printer.

1 Suppresses page ejects on the terminal only.

2 "Normal mode". Any value of 2 or more clears the screen (or ejects a page on the printer), and sends the corresponding number of nulls.

"bs" is the "backspace" key value. This is the decimal value of the ASCII character to echo to the terminal when the backspace key is pressed. This is typically set to 8 or 21.

"pw" is the current "printer width", or the number of print positions per line on printer output.

"pd" is the "printer depth", or the number of lines per page on printer output.

"tt" is the "terminal type" code. This designates the terminal driver to use and is vital to set prior to invoking any process which attempts to perform any kind of cursor control, such as UP, FlashBASIC, or PROC.

The terminal driver definitions are located in the "dm,devices," file. To obtain a listing of the available types, enter: "sort only dm,devices,".

Terminal types in D3 use descriptive names, rather than just a single alphabetic character.

Syntax

term tw,td,ls,ld,fd,bs,pw,pd,tt {(option)}

term tt {(option)}

Options

c Recompiles the terminal definition from the source.

h Sets high-intensity.

k Set the input/output terminal translation table. The name of table is defined in the fourth value of the attribute 1 of the 'devices' item. The name of the translation table is the item-id of an item in the 'keyboards' file.

o Creates a FlashBASIC cursor-control module when used with the "c" option. This FlashBASIC module is used by code generated with the FlashBASIC compiler. This option will not work with non terminal device definitions (printers).

r Redisplays resulting term parameters after changing.

Example

```
term 79,24
Change the terminal width and depth only.
term ,24
Change the terminal depth only.
term ,,,,,,132,62 (r
terminal name: wy-50
product name: WYSE 50
terminal width: 132   printer width: 132
depth: 82            depth: 62
lineskip:    0
lf delay:    1
ff delay:    1
back space:  8
Change the printer width and depth only, and display the resulting
changes.
term wy-50
Change the terminal type only.
term wy-50 (c
Change the terminal type and compile the terminal definition from the
"devices" file.
term ibm3151 (k
Set the terminal type to 'ibm3151' and activate the keyboard
translation associated to this terminal.
```

term-type

reads the terminal and printer characteristics from a file and automatically sets them, or sets them and updates the permanent settings for the current line.

In D3, the "pibs" file contains the parameters used by "term-type". The Update processor may be used to update the characteristics for each pib (port).

Under most D3 Unix implementations, pibs "float". This means that the same port is not always necessarily attached to the same Unix "pid". For this reason, the "choose.term" program is provided with D3. This should be invoked from each user's logon macro. See the "users" file.

In D3 Unix implementations, when "term-type" is used without arguments, it does the usual "term" setting for the current port by checking the port/control item in the "dm,pibs," file.

The item-id of the port/control item is the port.number for the current line. Attribute 2 of the port control item may contain a terminal name, as well as the appropriate "term" characteristic parameters. If the item is found, and it contains a terminal name, it is assigned as the D3 term type. If there is no terminal name, the terminal name is obtained from the Unix environment variable, "TERM". An item with the same name must exist in the "dm,pibs," file. If attribute 2 contains terminal characteristics (such as the screen size, ...), but no terminal name, the terminal name is obtained from Unix and the new characteristics are applied.

If attribute 2 of the pibs file is empty, and either, it is not an D3 Unix implementation, or the value of the TERM environment variable is not found in the devices file, then the terminal type of the previous logon to that pib is used. The terminal width is set to 79 and the depth to 24 regardless of what they were set to during the prior logon. If there were no prior logon or someone has done a reset-user on that pib then the term-type is determined by the first device compiled into the dict devices ccb item. Here also the default values for terminal width (79) and depth (24) are used.

Options

k Sets the keyboard input translation.

s Suppresses all output.

term.font

downloads fonts to a wyse 60 terminal or a VGA/EGA console (the latter only for D3/SCO assuming the SCO utility "vidi" is present).

It uses a special file which has 8x16 matrixes for every character in hexadecimal format.

The default font file name is "MOSCOW,FONTS.TERM,"

Syntax

term.font {font.file.name} {font.id}

Options

O Overwrites Unix font file.

R Requests permission to start loading.

S Shows the characters at loading time.

termp

interactive form of the "term" command.

It prompts for the terminal and printer characteristics for the line on which the verb is invoked.

See the "term" command for a complete explanation of all the arguments requested by this process.

Syntax

termp {(option)}

Options

r Redisplays characteristics after changing.

time

displays the system time and date. If no time.parameter exists, the current time and date will be displayed.

Syntax

time {time.parameter} {(option)}

Options

p Directs output to system printer, via the Spooler.

time-date

invokes the "timedate" verb to toggle or display the status of the time and date display.

Syntax

time-date {(options)}

Options

f Toggles OFF. Suppresses display of time/date at TCL.

n Toggles ON. Displays the time/date at TCL. The default is "on"

s Suppresses output of status message.

time-date-off

invokes the "time-date" command with an "f" option, turning off the automatic display of the time and date at the TCL prompt.

time-date-on

turns on the automatic display of the time and date at the TCL prompt.

Example

```
time-date-on
[1306] The time-date display is on.
```

timedate

sets or displays the status of the time and date display.

Syntax

timedate {(options)}

Options

f Toggles OFF. Suppresses display of time/date at TCL.

n Toggles ON. Displays the time/date at TCL. The default is "on"

s Suppresses output of status message.

Example

```
timedate (f
[1307] The time-date display is off.
timedate (n
[1306] The time-date display is on.
```

timedate-off

invokes the "time-date" command with an "f" option, to turn off the automatic display of the time and date at the TCL prompt.

timedate-on

invokes the "time-date" command with an "n" option, turning on the automatic display of the time and date at the TCL prompt.

Example

```
timedate-on
[1306] The time-date display is on.
```

timeout

sets the time (in seconds) to automatic logoff on inactivity.

If no timeout.value is provided, then the current value is displayed.

A timeout value of 0 disables the timeout. This state is the default.

If the line sits at an input for the timeout period, a logoff will take place.

Syntax

timeout {timeout.value}

tlog-restore

restores data from a transaction log magnetic media.

This is an alternative to restoring at the end of a full restore or account restore from a "save-class" media.

Syntax

tlog-restore {(options}

Options

p Directs output to printer, via the Spooler.

h Hot backup mode, do not prompt for more reels, assume true.

s Suppresses output. This outputs file headers only.

y Accept any reel number

to

see "logto".

touch

reads and re-writes all items, without changing them, in either specified files or in all files in specified accounts.

The main usage of this command is to force the transaction logger to enqueue for logging all items in files or in accounts, to synchronize data bases across a network.

itemlist A list of elements to process. By default, this is a list of files. Complete path names are processed. With the (A) option, the specified elements are account names, in which case all files in the specified accounts are processed.

If an explicit list of elements is not provided, a select list must be used.

The name of each file is displayed as it is processed. At the end, the number of files and items processed are displayed.

Syntax

touch {itemlist} {(options}

Options

F The list of elements is a list of files. This is the default option.

A The list of elements is a list of accounts. Synonym accounts are not processed.

L Log only. This option bypasses the update mechanism, but places all items into the transaction log queue. This makes the "touch" much faster and less abusive so long as a true update is not required.

Q Quiet option. Suppresses all messages.

Example

```
select mds # 'dm'
touch (a
Touch all accounts on the systems, except 'dm'.
touch customers inventory
Touch all items in the files 'customers' and 'inventory'.
```

transaction

system administration utility for transaction processing.

The "transaction" utility provides several different functions depending on the "keyword" provided. Current "keyword" values include:

CACHE ON/OFF Changes or queries the default cache condition for the process. The global default can be examined or set by adding the "g" option. See the BASIC "transaction cache" statement for more information on this setting. Note that the setting displayed by this command reflects the TCL default setting. If the default has been modified by a lower-level BASIC program, then these changes are not reflected. However, if the cache setting is then changed by the "transaction" TCL command, then the changes WILL be reflected in all lower-level BASIC programs.

FLUSH ON/OFF Changes or queries the default flush condition for the process. The global default can be examined or set by adding the "g" option. See the BASIC "transaction flush" statement for more information on this setting. Note that the setting displayed by this command reflects the TCL default setting. If the default has been modified by a lower-level BASIC program, then these changes are not reflected. However, if the cache setting is then changed by the "transaction" TCL command, then the changes WILL be reflected in all lower-level BASIC programs.

STATUS Displays the status of a transaction for the current line. The following information is displayed.

- Pib Process ID.

- Status Transaction status. This is "inactive," "active," or "commit." Normally, processes in the "inactive" state are not displayed unless the (z option is used.

- Size Number of operations in the current transaction.

- ID# Transaction id number.

- Name Transaction name if the "name" parameter was specified in the "begin work" statement.

Another process, or range of processes may be queried by specifying that process or process range. If the "g" option is used, then the information is displayed for the whole machine.

RECOVER Recovers transactions after a system halt. This operation should only be executed during the SYSTEM-COLDSTART macro while the system is in single-user mode. The

"recover" option attempts to roll-forward each committed transaction. If this fails, then the transaction is rolled-back. If the transaction cannot be completely rolled-back, then an error is printed and the transaction is aborted.

Syntax

transaction {keyword} {num{-num}} {(options}

Options

h Suppress heading.

g Apply operation to whole machine.

s Suppress messages.

z Examine all processes even if there is no active transaction.

trap

allows a TCL command (signal handler) to be executed upon receiving a specific signal.

The "trap" command traps a system-generated signal generated by the D3 monitor or by the Unix kernel and executes a TCL statement. Without any argument, lists the currently used signals.

"?" displays "help" and the list of user definable signals.

"*" specifies all user definable signals.

If the command is not specified, the signal handler is set to null, thus ignoring the signal. If a question mark (?) is used as a command, the signal handler, if any, of the specified signal is displayed.

If "command" is the keyword "default", the signal is reset to the default listed in the table in example 1. Otherwise, it may be any TCL command, including spaces and options.

Signal handlers are set for the whole virtual machine. When it is desirable to differentiate the signal handlers depending on the user or account, a FlashBASIC program, a PROC or an environ script must be provided to execute commands selectively. The signal handler is executed by pushing a level. Therefore, the user might see some effect on the terminal (Update processor screen being redisplayed, for instance). If fifteen TCL levels are pushed when the signal occurs, it will not be processed. Signals are always pre-emptive, even upon their own signal handler. For a signal handler to be processed correctly, the process receiving the signal must be logged on.

signame is one of the valid user definable signal names:

"alrm" Stands for "alarm". When a Unix SIGALRM is generated, either by a Unix program called from the D3 process, or by the FlashBASIC program calling the "%alarm()" C function, a signal is sent to the process. This signal can be used to log the process off, for instance, after a given time. The TCL "alarm" command generates this signal. The default action is to ignore the signal, therefore, an explicit "trap alrm" command must be used.

"dcd" Stands for Data Carrier Detect. If the DCD OFF protocol is active on one port, a HANGUP signal is generated by the terminal Unix driver when the DCD signal is dropped by the modem. The default handler logs the process off. On a network, a more complex program might be necessary to terminate the connection properly and must finish with a 'disc' command. See the section "hangup" for more details about the precautions to take when writing a signal handler for the loss of carrier.

"pwr" Stands for PoWeR off. When an imminent power failure is detected by Unix, the Kernel sends a SIGPWR signal to all active processes on the system. This signal is relayed to the D3 processes. Care must be taken to synchronize the default system handler (defined in the inittab) with the D3 handler. This signal might not be generated, depending on the hardware configuration.

"tdet" Stand for Tape DETach. When a "t-det (u)" command is issued, the process owner of the tape device receives this signal, so that the device can be closed properly. The handler for this signal is just a comment. It MUST NOT issue any tape command, including a "t-det".

"tdmof" Stands for TanDeM OFf. When the master process terminates, this displays "TANDEM TANDEM ON port.number". The slave terminal TERMINATED receives this signal.

"tdmon" Stands for TanDeM ON. When a process activates TANDEM, this displays "TANDEM STARTED ON port.number", both the master and slave terminals receive this signal.

Syntax

trap { [*|signame] { [?|default|command] } }

Example

```
trap
Sig   Description      Handler
----- ---------------- -------------------------
tdmof Stop tandem      display TANDEM TERMINATED
tdmon Start tandem     display TANDEM STARTED
dcd   Modem hangup     off
mirof Stop mirror      display MIRROR TERMINATED
alrm  Alarm            run dm,bp, useralarm
tdet  t-det (U         display Tape detached with (U)
pwr   Power off        off
 List the current settings
trap * default
 Resets all signals to their default.
trap tdmon
Disables the signal associated with TANDEM.
trap tdmof display @(-13)TANDEM ENTERED@(-14)
Sets the TANDEM ON trap to display a message in reverse video.
```

truncate-ovf

Truncate the safe overflow table to 1 block from high-water to maxfid and kill the b-tree table.

Syntax

truncate-ovf

txlog

invokes a menu for administration of the transaction logging subsystem.

When active, transaction logging logs all updates to files which have been marked for participation in logging. This involves changing the "d-pointer" to the file to "dl". This can also be indicated when creating the file by using the "l" option. Logging may go directly to magnetic media, or held in a file until the media can be attached.

Logging, or enqueing, occurs constantly in files that are marked as participants in transaction logging by having a "dl" designated in the "d-pointer" (attribute 1). "dequeing" occurs when transactions are being dumped to magnetic media.

txlog-off

turns off transaction log enqueuing for the current process.

No matter what file is updated by this user, the items will not be placed in the queue for later dequeuing to the tape or hot backup machine.

Options

a turns off transaction log enqueuing for all processes.

txlog-on

turns on transaction log enqueuing for the current process.

Options

a turns on transaction log enqueuing for all processes.

txlog-status

retrieves the last status of the transaction logging subsystem.

When active, transaction logging logs all updates to files which have been marked for participation in logging. This involves changing the "d-pointer" in the file to "dl". This can also be indicated when creating the file by using the "l" option. Logging may go directly to magnetic media, or held in a file until the media can be attached.

Logging, or enqueing, occurs constantly in files that are marked as participants in transaction logging by having a "dl" designated in the "d-pointer" (attribute 1). "dequeing" occurs when transactions are being dumped to magnetic media. See the "txlog" command.

Example

```
txlog-status
[609] Transaction logger not started.
```

type-ahead

enables or disables the system-wide type ahead feature, or displays the current status if no options are provided.

"Type ahead" is the process of buffering characters entered at a keyboard to allow for faster typing.

When type-ahead is "on", the system allows typing in characters "ahead" of their being displayed.

For example, after entering a "select" command at TCL, a "save-list" command can be entered into the type-ahead buffer. When the "select" finishes, the "save-list" command will be immediately entered provided a <return> was issued at the end of the command.

The command does not display on the screen until the previous process returns to TCL. There is no practical limit to the number of commands that can be "typed ahead", short of one's confidence in their abilities to type in the dark.

Syntax

type-ahead {(options}

Options

f Turns off type-ahead.

n Turns on type-ahead. This is the default.

s Suppresses output of status message.

Example

```
type-ahead
[1312] Type-ahead is on.
type-ahead (f
[1313] Type-ahead is off.
type-ahead (n
[1312] Type-ahead is on.
```

type-ahead-off

invokes the "type-ahead" command with an "f" option, disabling the system-wide type ahead facility.

type-ahead-on

invokes the "type-ahead" command with an "n" option, enabling the system-wide type ahead facility.

u

Invokes the "update" verb.

ud

invokes the Update processor to edit attribute-defining items or file-defining items in the dictionary of the specified file.

The program determines which type of entry is being edited and presents a different set of attribute names depending on whether the item is a "d-pointer" or a standard attribute-defining item (an "a", "s" or "x" type item).

Syntax

ud file.reference {item.list}

um

Displays the conversion units set by the set-units, set-dozens, set-decimal or set-thousands verbs.

Syntax

um

Example

```
:um
Decimals conversion set.
```

unlink-pibdev

unlinks a process from a port.

This is typically used to disconnect a process from its associated terminal I/O so it can be controlled by the calling process. Controlling processes like "tandem", "mirror" or "converse" may not be unlinked.

If the "process" is omitted, the current process or PIB is unlinked.

The specified "port.number" must be previously linked to the "process" or the current "process" if the "process" argument is ommitted.

Syntax

unlink-pibdev {process,} port.number

Example

```
unlink-pibdev 3
Unlinks the current process from port 3.
unlink-pibdev 2,3
Unlinks process 2 from port 3.
```

unlock-file

clears file locks which have not been cleared through normal system operation.

File locks are set by FlashBASIC "filelock" statements and other mechanisms.

File locks provide a means of assuring data reliability by preventing one process from updating a file while another process examines a batch of records from that same file.

File locks are displayed by "list-locks" as special item locks with an item id of "*" and a hash of "0".

Note that a "clear-locks (i" or a "clear-locks" will release all file locks.

Syntax

unlock-file file.reference {(options}

Options

i Suppresses "item 'x' released" message.

unlock-frame

clears the memory corelock previously 'corelocked' with a "lock-frame" command.

"fid" is the (decimal) frame number to unlock. A leading "." indicates that the frame number is in hexadecimal.

Syntax

unlock-frame {.}fid

Example

```
unlock-frame 7891
unlock-frame .1ed3
```

unlock-group

clears group locks.

This is capable of clearing all types of group locks or specific types of locks, such as: update group locks, retrieval group locks, or locks on file control blocks.

Group locks provide a means of assuring data reliability by preventing two processes from attempting to update the same group at the same time.

Many processes within the system set group locks. For instance, the "save" processor locks a group while it is being saved.

The "read" statements (in FlashBASIC) that have a "locking" form set "item" locks only while working with an item. This means that two processes can access items in the same group, provided they do not attempt to update the same item. The "locked" clause in the "read" statements allow a graceful means of dealing with the situation of attempting to access a locked item. The program can take a special path and indicate to the operator that they should "try again later."

Syntax

unlock-group {.}fid{-{.}end.fid} {(options)}

Options

fid{-end.fid} Used only when the fid(s) is specified in options string. The fid is implied as a decimal number unless it is preceded by a period, in which case it is considered to be a hexadecimal reference.

f Releases group locks placed on file control blocks only.

i Suppresses the "Group 'x' released" message.

o Releases read-only locks only.

r Releases retrieval group locks only.

u Releases update group locks only.

Example

```
unlock-group 15000
Unlocks the group at (decimal) fid 15000
unlock-group .3A98
Unlocks the group at (hexadecimal) fid 3A98 (15000 in decimal)
unlock-group (f
Unlocks FCB's only.
unlock-group 15010-15020
Unlocks groups between (and including) (decimal) fids 15010 through
15020.
```

unlock-item

clears item locks that have not cleared through usual system operation.

Item locks are set by FlashBASIC "read" statements and other mechanisms.

Item locks provide a means of assuring data reliability by preventing two processes from attempting to update the same item at the same time.

The "read" statements (in FlashBASIC) that have a "locking" form set "item" locks only while working with an item. This means that two processes can access items in the same group, provided they do not attempt to update the same item.

A "locked" clause in "read" statements allows a graceful means of dealing with the situation of attempting to access a locked item. The program can take a special case and indicate to the operator that they should "try again later."

When a process attempts to access an item that is locked, the screen simply freezes and starts "beeping". Usually, there is only a brief wait until the item is unlocked, at which time the terminal stops beeping and is granted access to the item.

Other times, however, the locking system does not unlock the item for whatever reason (aborts, programming errors, logoffs, bad karma, etc.). It is in cases such as this, and only in these cases, that this verb should be used.

Syntax

unlock-item file.reference {item.list*} {(options}

Options

start.port.number{-end.port.number} Used only when fid(s) specified in options string. The fid is implied as a decimal number unless it is preceeded by a period, where it is then considered to be a hexadecimal reference.

i Suppresses "item 'x' released" message.

Example

```
list-locks (i
Item Locks      PIB# Lvl Hash  Item-id    Filename
29198  (00720E)  33  2   2D52  33109      cx
29003  (00714B)   7  1   2C8E  32914      cx
234034 (039232)  86  0   1923  record2    doc
179222 (02BC16)  25  0   248A  74845480   entity
234034 (039232)  89  0   404B  accounts   doc
169528 (029638)  35  1   6432  110462     entity
71998  (01193E)  12  0   7250  8462       action
79366  (013606)  79  0   04A4  kr         tcl-stack
This displays the current item locks.
unlock-item action 8462
[454] Item '8462' unlocked.
This unlocks the item shown.
```

unset

deletes a D3 user shell variable.

Syntax

unset variable.name

up

Invokes the "update" command.

update

invokes the "Update processor" for adding, changing, deleting, or simply "cruising" through data in a given file.

The "attr.list" functions in the same way as the "outlist" in a "list" or "sort" command. It indicates the names of the attributes that are to be affected by the command. Specific attributes can be updated, without using the default macro defined on the file's "d-pointer".

"u" is a synonym of "update".

Syntax

update file.reference {item.list*} {attr.list} {(options)}

Options

i Includes item-id in UP workspace. Each item prompts for an item-id. (see "id-prompt")

s Toggles off spelling checker. (see "speller-off")

l "Look" only. The item cannot be filed or deleted.

c Clears the screen before bringing each item into UP workspace.

r "Raw" option. When using this option, UP will show non-displayable characters as periods, and the escape character as a "[". In this mode, the escape character can be entered directly. The raw option also causes the Update processor not to trim trailing attributes.

Example

```
update entity
In this form, the default list of attributes is obtained from the
"macro" attribute in the "entity" file-defining item located in the
dictionary of the entity file.
update invoices customer.name invoice.date invoice.amount
This explicitly requests the three fields shown, regardless of the
"default" macro.
update entity (cil
Like the first example, but uses "c" to clear the screen between each
item, "i" to include the item-id of each item, and "l" for "look
only."
update entity id-prompt (cl
This is exactly the same as the previous example, but uses the
"id-prompt" modifier rather than the "i" option.
```

update-abs-stamp

invoked by the "abs.fid" command and is the verb that produces the actual output.

Syntax

update-abs-stamp file.reference

update-account

Invokes "update-md".

update-accounts

updates verb definitions in a given account md, or in a list of mds. Any conflicts found are logged in the "CLASHES" file, with a new data section added for each account.

The account is checked in the following manner:

If an item is in "newac" but not the md, then it is copied to the md.

If an item is in both "newac" and the md, and is a file-defining item in the md, the md item is copied into the "md,clashes,<account.name>" file and retained in the md.

If an item is in both "newac" and the md and is not a file-defining item, the newac item is written over the item in the master dictionary. In addition, if the newac item is of a different type from the md item, the md item is written into the "dm,clashes,<account.name>" file.

If an item is a verb in the md and there is no corresponding item in "newac" nor in the "dm" master dictionary, the md item is written into the "dm,clashes,<account.name>" file, and deleted from the md.

If an item is a PROC in the md and there is no corresponding item in "newac", the PROC is retained.

If an item is a cataloged FlashBASIC program in the md and there is no corresponding item in "newac" nor in the "dm" master dictionary, the format of the item is updated to the D^3 format.

All other items are retained.

Syntax

update-accounts account.name {account.name...}

update-logging

toggles or displays the status of the transaction logger subsystem.

With no options, the current status is displayed.

Syntax

update-logging {(options)}

Options

f Toggles OFF. Only updates to "dl" files are logged.

n Toggles ON. All updates to files that are not "dx" or "dy" are logged. (See the "txlog" command for logging just the "dl" type files).

s Suppresses output of messages.

Example

```
update-logging
[1325] Update-logging is off.
update-logging (n
[1324] Update-logging is on.
```

update-md

updates the verbs and related items in a given account md, using the "newac" file on the "dm" account as the base standard for determining what should be there. It must be run from the "dm" account.

"update-md" checks for illegal verb definitions and converts "old" catalog pointer items to D3 format.

An "active list" (consisting of account names) may be used prior to running "update-md" to update several accounts at once.

If no account name is specified, and no list is active, the system requests the account name.

The process prompts with the following messages:

Create user id from each account name (y, <n>)?

This prompt allows the automatic creation of user-ids corresponding to any account updated. Valid responses are "y", "n", or "<return>", which defaults to "n".

Enter md file name:

This is where the "new" account name is entered. Entering a <return> at this prompt terminates the process.

"update-account" and "update-accounts" are synonyms of "update-md".

Syntax

update-md {account.name}

Example

```
update-md
Create user id from each account name (y, <n>)?n
Enter md file name:production
```

update-prot

toggles or displays the status of the file update protection scheme. When enabled, this feature globally protects all item updates so that a power-off condition will not cause a GFE.

By default the update protection is on.

If the user turns update-protection off, and later turns it back on, the user MUST use a "FLUSH" command afterwards to be sure that all previous updates are written to disk.

If update protection is desired for selected files only, the user may disable the global update protection flag (with "UPDATE-PROT-OFF"), and enable the protection on the selected files by changing attribute 1 of the D-pointers of those files to "DU". When all desired files have been changed to a "DU" type, the user MUST use a "FLUSH" command to write all previous updates to the disk.

If update protection is desired for most of the system, but the user wishes to disable it on certain files, the user can change attribute 1 of the desired D-pointers to "DN". These files will not be protected irrespective of the global update-protection status.

Syntax

update-prot {(option}

Options

f Toggles update protection off.

n Toggles update protection on.

update-prot-off

toggles off the global file update protection scheme.

See "update-prot" for an explanation of the protection scheme.

update-prot-on

toggles on the file update protection scheme.

See "update-prot" for an explanation of the protection scheme.

user-coldstart

performs the last step of the coldstart process and is the place where users can customize their own sequence of activities to perform each time the system is powered on or rebooted.

This avoids having to alter the "system-coldstart" command each time an upgrade to a new release is performed.

The commands executed by this macro vary from platform to platform, so its exact behavior can only be determined by looking at the macro, which usually exists in the "md" of the "dm" account.

Some of the commands typically executed by "user-coldstart" are attached as related subjects.

"user-coldstart" is called after "system-coldstart" completes. See "system-coldstart" for more information.

Example

```
n
startspooler (c
startptr 0,(10,12,17),1,p0 (s)
assignfq 10,hp.lzr2p.80
assignfq 12,hp.lzr2p.96
assignfq 17,hp.lzr2p.132
```

user-shutdown

called by the "shutdown" verb as the first procedure in shutting down the machine.

User specific shutdown procedures can be inserted in this macro, like stopping printers, networks or the transaction logger.

user.coldstart

invoked by the "coldstart" to call the "user-coldstart" routine.

useralarm

is an alarm handler used by the 'trap' command when a task is scheduled to be executed at a given time. When the alarm goes off, it scans the item 'dm,alarm.file, username' to find what to do next. This command is not normally executed at TCL.

UVDUMP

dumps a uniVerse account onto tape as a series of "t-dumps" which can be later read by the "generic-restore" D3 utility, or to a device (Unix pipe or serial line) to be transferred to a D3 system using the 'uvget' utility.

This BASIC program is designed to be transported to a uniVerse machine where it must be compiled and cataloged in the accounts which will be transferred to D3.

Once set up, the UVDUMP program provides a menu of options.

Option number 1 allows the user to set various destination parameters including the destination frame size, the desired capacity of the files when they are restored on D3, the number of items to sample to derive the expected modulo, and default create-file options. The defaults are set properly for most uses.

Option number 2 creates a control file for dumping the account. This must be done prior to actually making the tape. This option scans for relevant files, and determines the modulo of those files once they are moved to D3.

Option number 3 allows the user to edit the control item. This control item has a line of information for each file section. The format is as follows:

DICT/DATA {file.name} {modulo} {create-file options}

Lines may be added, modified, or deleted from this file to change the way the account is later dumped.

Option number 4 actually dumps the account based on the information in the control item. Each file section is dumped as a small "t-dump" of a header block followed by a large "t-dump" of the actual data in the section. Note that this option does NOT rewind the tape so that multiple accounts may be placed on the same tape.

The tape created by this utility must be read by the D3 utility "generic-restore".

Option number 5 transfers the account to a device based on the information in the control item. This option asks for the device name (default '/dev/pipe'). The device can be a pipe (created on Unix by 'mknod devicename p') or a sequential device (tape, serial line, Unix file,...). Before starting this option, the other 'end' of the device (if a pipe or a serial device) must be read by the D3 utility 'uvget' (see REF entry for uvget).

Syntax

UVDUMP

uvget

loads an account from a pipe, serial line or tape, created by the UVDUMP utility, to transfer uniVerse accounts to an D3 system on Unix.

'device' is the device name specified in the option '5' of the UVDUMP utility. If 'device' is a pipe or tape, the option (B8192) must be used.

This command is an alternative to the generic-restore TCL utility.

Syntax

uvget device {(option}

Options

Bn IO block size. 'n' is the block size in bytes. This option is almost always required, due to the way the uniVerse WRITEBLK statement works.

V Verbose. Display information abouf the files being loaded.

Example

```
uvget /dev/pipe (vb8192
```

verb.type

is called from the attribute-defining items "v/cat", "v/mode", "v/desc", and "v/flags" and attempts to decipher the type of item in the md (i.e. verb, macro, menu, PROC, cataloged FlashBASIC program, or "unknown".

Example

```
list md verb.type
```

verify-abs

Invokes "verify-system".

verify-index

verifies the integrity of one or all b-tree indices in a given filename.

"a.code" specifies the "a (algebraic)" processing code to be used in forming the keys to the index. The processing code must include an attribute number.

The "*" (asterisk) designates all indices.

A "?" displays on-line help for this command.

With no options specified, the process examines each item in the file and verifies each of the indicies specified. A counter displays on the screen for monitoring the progress of the "verify-index".

Syntax

verify-index file.reference a.code {(options)}

verify-index file.reference * {(options)}

verify-index ?

Options

c Compares the file to the index only.

e Creates an error log. When this option is specified, the command will prompt for the error log file and item.

f Displays link and node-id's during verification.

i Displays item-id's.

k Verifies links and nodes.

l Locks each data item as it is verified.

m Suppresses all messages.

o Reports missing item-id's and locates items indexed in the wrong place. While using this option makes the process run slower, it increases the reliability and effectiveness of the process.

p Directs report to system printer, via the Spooler.

r Inhibits index recreation on link error.

s Suppresses display of item counter.

v Verifies that the data exists for the given indices. This is used in the case where the index contains pointers to item-id's which have been deleted. This is not normal.

x Suppresses automatic fix of bad indices. This option produces a report only.

z Stops processing after link and node verification.

Example

```
verify-index entity a1
Verifies the index on attribute 1 (one) of the "entity" file.
verify-index entity * (o
Validates all indexes in the "entity" file.
```

verify-system

verifies the integrity of the D3 system software by comparing computed checksums for the executable abs frames with corresponding checksums stored in the data section of the "file.reference" file.

If "file.reference" is not specified the boot abs "dm,abs," is verified.

"verify-system" works with an active list.

In D3, "verify-system" can be ended early by entering a "x" while it is running.

Syntax

verify-system {file.reference} {(option}

Options

o Generates a new "checksum" item in the specified abs file.

Example

```
:verify-system
verifying ABS
System Verifies!
```

video.demo

demonstrates video display functions.

view

displays a single item in a scrollable fullscreen window.

This can be useful when a user wishes to display an item without editing it.

Syntax

view filename itemid

Example

```
view dm,bp, :scroll
This will display the view program.  Press the "H" key for keyboard
help.
```

w

see "where".

what

displays the system configuration and the status of system locks, specified lines, Spooler, and printers.

The following system configuration information is displayed:

"core"

‘Memory" is the amount of memory allocated to the D3 virtual machine, in Kilobytes.

"pibs"

is the number of process identification blocks. This includes communication lines for terminals and serial printers, Spooler, phantom processes, and scheduler.

"pcb0"

is the frame-id (fid) for the primary control block (pcb) for line zero.

"sysbase/mod"

is the base fid and modulo of the "mds" (system) file.

"maxfid"

is the last addressable frame-id (fid) on the system.

"available"

is the number of frames available in overflow.

"dfsize"

is the number of bytes available per data frame. The actual frame size is a bit larger. A dfsize of 1000 indicates an actual frame size of 1024. A dfsize of 2000 indicates an actual frame size of 2048.

In addition, "what" uses the "where" verb to display the current status of each line, the "list-locks" verb to display the system lock table, and the "sp-status" verb to display the Spooler and printer status.

Syntax

what {port.number{-port.number}} {'user-id'} {(options)}

Options

'user-id' Outputs "where" status only for users of the specified user-id.

{port.number{-port.number}} Outputs the status for the given port, or a range of ports.

l Suppresses display of lock status.

n No pause. Suppresses the pause at the end of the page on the terminal.

p Directs output to the system printer, via the Spooler.

s Suppresses the display of "sp-status".

w Suppresses the display of "where" output.

z Displays "where" status for all lines, including "inactive" lines (those which are not currently logged on).

Example

```
what
08:29:47 29 Feb 1992
memory  pibs   pcb0  sysbase/mod   maxfid  available  dfsize
18000k   138   1536     2199   7   358398      53442    2000
System Locks
0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
0 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
1 ### ### ### ### ### ### ### ###
Basic Locks
0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
0 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
1 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
2 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
3 ### ### ### ### ### ### ### ### ### ### ### ### ### ### ### ###
Spooler Locks
mq  iq  fq peq
0 ### ### ### ###
Group Locks        Update-Lock   Retrieval  Read-Only
Owner         Locks     Locks
151996 (0251BC)       38             0         0
155300 (025EA4)        -             1         0
166106 (0288DA)        -             1         0
131836 (0202FC)       82             0         0
131879 (020327)        -             1         0
282213 (044E65)        5             0         0
Item Locks          PIB#  Lvl  Hash  Item-id      Filename
109547 (01ABEB)     56   0   2379    tcl.what   ap.doc
151996 (0251BC)     38   0   289E    127910     journal
131836 (0202FC)     82   0   6C2C    monitor    status
011 00060B BF10 000018      sp.serialsleep:080
012 00060C FF10 000018      au.start:36C     tcl1:0E8
013 00060D BF10 000018      sp.serialsleep:080
014 00060E BF10 000018      sp.serialsleep:080
015 00060F BF10 000018      sp.serialsleep:080
019 000613 BF10 000018      sp.serialsleep:080
128 000680 BF10 000018   P  sp.sleep:040     sp.spoolout:0F0
130 04AA25 FF10 000018   P  br.unix.bix:000
131 044E0F DF10 000018   P  bt.searchx:17C   bt.read:028
137 039A0B BF10 000018   P  pp.sched:038
The spooler is inactive.
Printer #  0 is serial, inactive, and on line.
The printer is running on line  13.
Assigned output queues: 0 .
The number of inter-job pages to eject is 1.
Printer #  1 is serial, inactive, and on line.
The printer is running on line  11.
Assigned output queues: 3 , 4 , 5 .
The number of inter-job pages to eject is 1.
what 'dm'
Outputs all items indicated, but limits "where" output only to those
on the "dm" account.
what 13-22
Outputs all items indicated, but limits "where" output only to lines
13 through 22.
what (lsw
08:40:52 29 Feb 1992
memory  pibs   pcb0  sysbase/mod   maxfid  available  dfsize
18000k   138   1536     2199   7   358398      53418    2000
Outputs system configuration only.
```

where

displays the current execution status of all processes currently logged on to system, or for selected port.numbers.

The information displayed by "where" includes:

"Ln:"

is the port.number. An "*" (asterisk) indicates the port that issued the command.

"PCB FID:"

is the frame-id ("fid") of the primary control block (pcb) for each port.

"PIB STAT:"

is the pib status of each port. See the "pibstat" command for an explanation of the bit settings.

"ABS base:"

is the beginning fid of the abs currently being executed by each port.

"Stat:"

is the current status of the port:

n: (hexadecimal) current TCL level. If blank, the port is at the primary level (level 1).

d: in system debugger.

p: phantom process.

t: tape attached.

When the (U) option is used, the "stat" fields contains the 'user tally' code. Possible values for the user tally are:

0 Port is logged off.

1 Port is pushed a level.

2 Port is in process of logging on.

3 Port is in "break/end" sequence.

5 Normal setting.

-1 Spooler controlled process.

-2 Scheduler process.

"R1 & Return stack contents:"

displays the abs mode currently executing on the port. The first entry is the current location. Subsequent entries (which are separated by spaces) are the return stack mode addresses. If the display says "corrupted workspace", the most likely cause is the workspace is not properly setup or the process is in the middle of "push level" or "pop level".

"w" is a synonym of "where".

Syntax

where {port.number{-port.number}} {user-id} {(options)}

Options

user-id Outputs status for a specific user-id only. Quotes are optional.

'account' Outputs status for processes using the specified account name.

port.number{-port.number} Outputs the status for the given port, or a range of ports.

h Suppresses the heading.

l Displays the status for each "pushed" level.

n No pause. Suppresses the pause at the end of the page on the terminal.

p Directs output to the system printer, via the Spooler.

u Returns the user tally (under the "Stat" column heading).

z Displays "where" status for all ports, including "inactive" ports (those which are not currently logged on).

Example

```
where 'dm'
Displays the status of those ports which are currently logged into
the "dm" user-id.
where 13-22
Outputs the status of port.numbers 13 through 22.
where (z
Outputs the status of all port.numbers.
where
 Ln  PCB    PIB   ABS  Stat R1 & Return stack contents
 011 00060B BF10 000018    sp.serialsleep:080
 012 00060C FF10 000018    au.tcl.upd:084   au.start:36C
 013 00060D BF10 000018    sp.serialsleep:080
 014 00060E BF10 000018    sp.serialsleep:080
 015 00060F BF10 000018    sp.serialsleep:080
 019 000613 BF10 000018    sp.serialsleep:080
 045 0469F9 F310 000018  3 au.input:000     au.get.cmnd:0D8
*056 046AA6 FF30 000018  2 ws.where1:000    ws.whatwhere:354
 069 0457DD F310 000018  1 me.pause:018     me.display:598
 070 04514D F310 000018  1 me.pause:018     me.display:598
 074 00064A BF10 000018    sp.serialsleep:080
 082 000652 BF10 000018    sleepsub:0F4     br.rqm:18C
 128 000680 BF10 000018 P  sp.sleep:040     sp.spoolout:0F0
 130 04AA25 FF10 000018 P  br.unix.bix:000
 137 039A0B BF10 000018 P  pp.sched:038
```

whereall

invokes the "wheres" verb. See "wheres" for more information.

Syntax

whereall {options}

Options

* see "wheres".

wherebt

invokes the "wheres" verb with the "bt" parameter. See "wheres" for more information.

The "bt" parameter indicates that only the lines currently in FlashBASIC runtime are displayed.

Syntax

wherebt {options}

whered

invokes the "wheres" verb with the "db" parameter.

The "db" parameter indicates that only the lines currently in the system debugger are displayed.

Syntax
whered {options}

whereindx

invokes the "wheres" verb with the "in" parameter. See "wheres" for more information.

The "in" parameter indicates that only the lines accessing b-tree indices are displayed.

Syntax
whereindx {options}

wherelk

invokes the "wheres" verb with the "lk" parameter. See "wheres" for more information.

The "lk" parameter indicates that only the lines with locks set are displayed.

Syntax
wherelk {options}

whereovf

invokes the "wheres" verb with the "ovf" parameter. See "wheres" for more information.

The "ovf" parameter indicates that only the lines accessing the overflow handler are displayed.

Syntax
whereovf {options}

wherepu

invokes the "wheres" verb with the "pu" parameter. See "wheres" for more information.

The "pu" parameter indicates that only lines which are pushed one or more levels are displayed.

Syntax
wherepu {options}

wheres

performs a variety of functions depending on the options or parameters provided.

"wheres" displays a more readable form of the "where" command, including the user-id, the account name, and the last command entered.

This command combines the effect of the following TCL verbs: "who", "sort pibs", "listabs", "term", "listptr", "where", "sp-status", "set-port" and "xonoff".

"whos" is a synonym of "wheres".

A number of variants of this command are available and all begin with "who" or "where".

The following special parameters are provided:

br Displays ports which are in running a FlashBASIC program.

bt Displays ports which are accessing b-tree indices.

db Displays ports which are in the system debugger.

in Displays ports which are accessing indices.

lk Displays ports which have locks in place.

op Displays ports which are in the Output processor.

ov Displays ports which are accessing the overflow handler.

pu Displays ports which are pushed one or more levels.

sp Displays ports which are accessing the Spooler.

tp Displays the port with the tape attached.

Syntax

wheres {options}

Options

In this command, options do not have to be enclosed in parentheses.

? Displays "help" on the command.

'account name' Displays status only for users of the specified account name. The quotes around the account name are required.

[user-id Displays status of requested user-id only. The "[" character is required. A "]" character is not allowed.

{port.number{-port.number}} Outputs status for the given port, or a range of ports.

c Displays port communications parameters (baud rate, parity, etc.).

l Displays the status for pushed levels, if any.

n No pause. Suppresses the pause at the end of the page on the terminal.

p Directs output to the system printer, via the Spooler.

z Displays "where" status for all ports, including "inactive" ports (those which are not currently logged on).

wheresp

invokes the "wheres" verb with the "sp" parameter. See "wheres" for more information.

The "sp" parameter indicates that only the lines currently accessing the spooler are displayed.

Syntax

wheresp {options}

wheret

invokes the "wheres" verb with the "tp" parameter. See "wheres" for more information.

The "tp" parameter indicates that only the line with the tape attached is displayed.

Syntax

wheret {options}

which

displays the implementation being used, the release level, the boot monitor date, the boot abs date, the boot abs data file date, the system data files date, and the system serial #.

This verb displays the version of the elements used to create the D3 system. The display includes:

"Virgin Install Monitor"

is the date on the monitor diskette most recently used to do a virgin install.

"Boot Monitor"

is the date on the monitor diskette most recently used to boot the system.

"Executable ABS"

is the date on the boot ABS diskette most recently used to do an ABS restore.

"ABS Data File"

is the date on the ABS data file on the most recent F-level restore file save diskette. This date should match the date of the executable ABS. If the dates do not match, a message displays. "verify-system", "where", and the system debugger do not work correctly if they don't match.

"System Data Files"

is the date the system data files were originally created.

"System Serial Number"

is the number assigned to this copy of the D3 system software.

In D3 Unix systems, the following fields have a different meaning than the standard D3 "which" command, and are presented in the format:

D3 Unix: system;filename:name;rel;ver;hw

Where:

"system"

is the system name. This field may contain spaces.

"filename"

is the configuration file name.

"name"

is the content of the 'name' statement in the configuration file.

"rel"

is the Unix system release.

"ver"

is the Unix system version.

"hw"

is the machine hardware name (or serial number).

Virgin Install Monitor is not used.

Boot Monitor is the Monitor generation date.

All other fields are standard.

With the "z" option, only the Release and Boot Monitor are displayed.

Syntax

which {(option}

Options

a Displays highest abs patch level loaded on the system.

p Output to printer

s Displays the system id number.

z Displays release implementation and date only.

Example

```
which
                  System Release Information
                  ==========================
   D3 Release Version  7.0.1.SCO16

   Most recent mload into boot abs performed at 11:37:46 on 12 Jun
1996.

   Implementation. . . . . 6386
   System Serial Number. . . 00001010
   System ID Number  . . . .
   Release . . . . . D3/UNIX:  386
   Unix Information. . . . . SCO_SV;pick0:SCO;3.2;2;0285FE12

   Boot Monitor. . . . . 7.0.1.M2
   Boot ABS. . . . . 7.0.1.A4
   Boot ABS Data File. . . . 7.0.1.A4
   System Data Files . . . . 7.0.1.D5
   Flash Revision. . . . . 7.0.1.F2
   SQL Revision. . . . . 7.0.1.s3
```

which-line

puts the current port number into the PROC primary input buffer. This was used by the old PROC-version of "power-off", which has subsequently been replaced by a FlashBASIC program.

Example

```
who.substitute
001 pq
002 oYou are using port #
003 hwhich-line
004 p
005 d
```

who

displays the port.number, user-id, and account name for the current port, a specified port, or all ports.

"port.number" is the port number whose current account is to be displayed. If not specified, the account for the current port is displayed.

An "*" as the "port.number", displays the account names for all ports that are logged on.

Syntax

who {port.number} {(options}

who {*} {(options}

who {user-id} {(options}

Options

a Converts output format to D3-style, showing the user-id followed by the account name. This is the default setting and is toggled with the "c" option.

c Converts output format to R83-style, showing the account name followed by the user-id.

n No pause. Suppresses the pause at the end of the page on the terminal.

z Displays status of all ports, including those not logged on. The account name is displayed as "unknown" for ports not logged on.

Example

```
:who
56 bob ref
:who 0
0 unknown
:who (c
[1320] R83 output for WHO display specified.
:who
56 ref bob
:who (a
[1321] D3 output for WHO display specified.
:who
56 bob ref
```

who-info

Enables or disables "who" output at TCL.

When the TCL who information is on, the pib, user, and account are displayed along with the time/date information when the user enters a blank line at TCL.

Options

f Turns off TCL who information

n Turns on TCL who information

who-info-off

turns off TCL who information

who-info-on

turns on TCL who information

whoall

invokes the "wheres" verb. See "wheres" for more information.

Syntax

whoall {options}

whobt

invokes the "wheres" verb with the "bt" parameter. See "wheres" for more information.

The "bt" parameter indicates that only the lines currently in FlashBASIC runtime are displayed.

Syntax

whobt {options}

whod

invokes the "wheres" verb with the "db" parameter. See "wheres" for more information.

The "db" parameter indicates that only the lines currently in the system debugger are displayed.

Syntax

whod {options}

whoindx

invokes the "wheres" verb with the "in" parameter.

The "in" parameter indicates that only the lines accessing b-tree indices are displayed.

Syntax
whoindx {options}

wholk

invokes the "wheres" verb with the "lk" parameter.

The "lk" parameter indicates that only the lines with locks set are displayed.

Syntax
wholk {options}

whopu

invokes the "wheres" verb with the "pu" parameter. See "wheres" for more information.

The "pu" parameter indicates that only lines which are pushed one or more levels are displayed.

Syntax
whopu {options}

whos

Invokes program "wheres".

whosp

invokes the "wheres" verb with the "sp" parameter. See "wheres" for more information.

The "sp" parameter indicates that only the lines currently accessing the spooler are displayed.

Syntax
whosp {options}

whot

displays the "where" information for the line with the tape attached.

Syntax
whot {options}

whovf

invokes the "wheres" verb with the "ovf" parameter. See "wheres" for more information.

The "ovf" parameter indicates that only the lines accessing the overflow handler are displayed.

Syntax
whovf {options}

wlist

is a FlashBASIC simulation of the "list" verb that additionally exploits b-tree indicies.

For instance:

list entity by name

In this example, only one attribute is specified. If there happens to be an index defined on the attribute referenced by "name", the response is practically immediate, since it can fetch the pre-

sorted b-tree for the requested list. If the sentence were to contain a more complex expression, such as in the case:

list entity by zip by name

Again, assuming that indices are defined on both the "zip" and "name" field, AQL processes the sentence as though the indices were not even there.

In some rare and peculiar cases, "wlist" may be faster than the "list" verb.

In addition, AQL does NOT use the indices if any wildcards ("^") or string-searching characters ("[" or "]") are present in the sentence.

wselect

a FlashBASIC simulation of the "select" verb that additionally exploits b-tree indices.

For instance:

sselect entity by name

In this example, only one attribute is specified. If there happens to be an index defined on the attribute referenced by "name", the response is practically immediate, since it can fetch the pre-sorted b-tree for the requested list. If the sentence were to contain a more complex expression, such as in the case:

sselect entity by zip by name

Again, assuming that indices are defined on both the "zip" and "name" field, AQL processes the sentence as though the indices were not even there.

In addition, AQL does NOT use the indices if any wildcards ("^") or string-searching characters ("[" or "]") are present in the sentence.

In some rare and peculiar cases, "wselect" may be faster than the "select" verb.

wsort

a FlashBASIC simulation of the "sort" verb that additionally exploits b-tree indicies.

For instance:

sort entity by name

In this example, only one attribute is specified. If there happens to be an index defined on the attribute referenced by "name", the response is practically immediate, since it can fetch the pre-sorted b-tree for the requested list. If the sentence were to contain a more complex expression, such as in the case:

sort entity by zip by name

Again, assuming that indices are defined on both the "zip" and "name" field, AQL processes the sentence as though the indices were not even there.

In some rare and peculiar cases, "wsort" may be faster than the "sort" verb.

In addition, AQL does NOT use the indices if any wildcards ("^") or string-searching characters ("[" or "]") are present in the sentence.

Example

```
wsort entity by zip by name with name "a]"
```

wsselect

is a FlashBASIC simulation of the "select" verb that additionally exploits b-tree indicies.

For instance:

sselect entity by name

In this example, only one attribute is specified. If there happens to be an index defined on the attribute referenced by "name", the response is practically immediate, since it can fetch the pre-sorted b-tree for the requested list. If the sentence were to contain a more complex expression, such as in the case:

sselect entity by zip by name

Again, assuming that indices are defined on both the "zip" and "name" field, AQL processes the sentence as though the indices were not even there.

In some rare and peculiar cases, "wsselect" may be faster than the "sselect" verb.

In addition, AQL does NOT use the indices if any wildcards ("^") or string-searching characters ("[" or "]") are present in the sentence.

xcs

enables or disables the extended character set for the current port or a specific port, or displays the status when no options are provided.

When xcs is on, the high order bit is not stripped from input that is transmitted to the system. This allows characters in the range 128 through 255 to be used as unique characters.

When xcs is off, the high order bit is stripped from input. Characters from 128 through 255 are not available.

Two macros, xcs-on and xcs-off, have been provided to toggle this function.

Syntax

xcs {port.number} {(options)}

Options

f Toggles xcs OFF.

n Toggles xcs ON.

s Suppresses the output of status messages.

t Toggles xcs. If "on", this turns it off, and vice-versa.

xcs-off

disables the extended character set function on a given port, or the current port if omitted.

Syntax

xcs-off {port.number}

xcs-on

enables the extended character set function on a given port, or the current port if omitted.

Syntax

xcs-on {port.number}

xonoff

disables or enables x-on/x-off on a specified port, or the current port.

When the xon/xoff capability is "on", "<ctrl>+s" stops the terminal from displaying output; "<ctrl>+q" is used to resume the display. If no options are specified, the status is displayed.

xon and xoff are used for "flow control" to printers and other devices.

Syntax

xonoff {(options}

Options

port.number (integer number) Indicates the port number.

f Toggles xon/xoff OFF.

n Toggles xon/xoff ON. This is the default setting.

s Suppresses the output of the status message.

Example

```
:xonoff
[1314] xon/off is enabled.
:xonoff (f
[1315] xon/off is disabled.
:xonoff (n
[1314] xon/off is enabled.
```

xtd

converts a given hexadecimal number into its equivalent decimal (integer) format.

Syntax

xtd hexnum

Example

```
xtd 3AF2
15090
```

Update processor (UP)

UP is a dictionary-driven, full-screen editor and data processor. Used with the Output processor (OP), it operates as a document processor.

UP can be used to view, create, and manipulate file items. UP facilitates editing of text, dictionaries and programs, as well as data files (remember, everything in D^3 is an item in a file). When used as a data processor, attribute-defining items in the file's dictionary are employed to verify and translate data as it is entered and retrieved.

FlashBASIC subroutines can be called from dictionaries allowing absolute control of all data entry using the Update processor and dictionaries. Without any attribute-defining items specified, UP functions as a full-screen editor.

When invoked, UP copies the specified item to a workspace and displays as much of the item on the screen as it can. UP places the cursor at the upper left corner of the item (top) on the first character of the first value of the first attribute and waits for your input.

Once the item has been loaded, any character you type will overwrite whatever is on the screen. Control keys are used to perform editing functions, but as long as the control key is not down, all characters are entered as typed. An automatic word-wrap feature moves whole words to the next line as they are typed when you overrun the end of the current line. An optional spelling checker beeps and rejects your input when you type a character that does not lead to a legal word.

Text is normally entered in the insert mode (<Ctrl>+r is used to toggle between the insert and overtype mode). While in the input mode, the return/enter key (<Ctrl>+m) creates a new attribute/paragraph. When in the overtype mode, that same key does not create a new attribute but instead goes to the next attribute. The default mode is overtype.

UP commands begin with a control character formed by simultaneously pressing the <Ctrl> key and one of the alpha keys. Some UP commands require more command characters, while others require text input.

Until the item is saved, changes made using UP have no effect on the actual item in the file -- only UP workspace is changed. The item can be exited at any time without filing, by typing <Ctrl>+xe>, and the item in the file will not change.

Using UP as a Text Editor

In the absence of any attribute list, UP functions as a text editor. An attribute list generates a data entry screen with whatever audits and transformations are defined in the corresponding dictionary. The text edit form is to specify just the file name and item list (no attribute list) and if no macro is present on the file-defining item, the raw item will be presented. Refer to the entries update, up, and u for more niformation about the update commands.

update file.reference {item.list} { (options) }

up

u

Cursor Movement

The cursor is the underline or block which moves around the screen when characters are typed. Cursors can have various forms depending on the terminal: highlighted, blinking, reverse video, etc. In all cases, the cursor indicates where the next character typed is displayed on the screen.

Cursor movement commands are used to position the cursor anywhere in the item without altering the text. The cursor actually moves on the CRT screen. And, since the screen is only a window into the item, the cursor may be moved off the screen forcing a screen redisplay (scrolling).

<Ctrl> +

k Moves the cursor forward (right) one character.

j Moves the cursor back (left) one character.

n Moves the cursor down one line.

b Moves the cursor up one line.

i Moves the cursor forward to the next tab stop in the line.

u Moves the cursor forward (right) one word.

y Moves the cursor back (left) one word.

f Moves the cursor forward to the next sentence.

d Moves the cursor back to the beginning of the sentence.

g Moves the cursor to the end of the current paragraph.

m Moves the cursor to the beginning of the next paragraph(attribute) if in overtype mode. Creates a new attribute if in insert mode.

t Moves the cursor to the top of the current item.

z n Moves the cursor to the top of the next screen.

z y Moves the cursor to the top of the previous screen.

z h Moves the cursor to the first line of the next screen, then moves the display up 12 lines.

z q Moves the cursor to the first line of the next screen, then moves display up one quarter of a screen.

z e Move the cursor to the end of the current item.

z a Redisplay the screen with the current line at the top of the screen.

z b Redisplays the screen with the current line at the bottom of the screen.

z p Redisplays the screen.

z numeric Goes to the specified line number.

Overtype and Insert Modes

When UP is first entered, it is in overtype mode. This means that characters typed overwrite the characters on the screen. Insert mode is used to insert new text just in front of the text beginning at the cursor. When text is entered in the insert mode, the following text is "opened" and shown on the next line until insert mode is exited or a cursor movement command is typed.

The letter "O" is displayed at the top right corner of the screen when in overtype mode; the letter "I" is displayed when in insert mode. As text is entered in either mode, UP word-wraps the text as necessary. That is, if a character string is longer than the remaining space on the current line, the entire string is moved to the next line. This means you can type continuously within a paragraph (attribute) without pressing the <Ctrl>+m.

<Ctrl>+r Toggle between overtype and insert modes.

Paragraphs

To create a new paragraph in the middle of existing text, enter the insert mode and type <Ctrl>+m. This terminates the current paragraph and prompts with the next attribute number. If the cursor is at the end of an item, <Ctrl>+m creates a new paragraph whether or not in input mode. In the absence of a dictionary definition, this is how UP is used as a word processor; each attribute is a paragraph.

<Ctrl>+m Same as <Return>. Always moves the cursor to the beginning of the next attribute. If in insert mode or at the end of an item, <Ctrl>+m creates a new paragraph. If UP is being controlled by a dictionary, <Ctrl>+m does not create a new attribute mark, rather the cursor moves to the next attribute.

Inputting Text Using the Spelling Checker

UP uses a spelling checker to verify text as it is entered. The spelling checker looks at each character to see if those typed so far make a legal word or may become a legal word. The words used by the spelling checker are kept in the words file in the dm account.

In order to activate the spelling checker, a q-pointer must be present on the current account and an index must be generated on attribute 0 of the words file. If an error is detected, the terminal beeps and does not display the last character typed. If an incorrect letter is typed, type the correct one and continue entering data.

To view the words immediately adjacent to the misspelled word in the words file, use <Ctrl>+u to get the next legal word or <Ctrl>+y to get the previous word. Otherwise, press one of the following:

<Ctrl> +

a Override spell checker and accept the characters.

a a Insert the word into the words file and continue.

a a a Turn off spelling checker.

z s Toggle the spelling checker on and off from any point in the item. If the spelling checker is on, <Ctrl>+zs toggles it off; if the spelling checker is off, <Ctrl>+zs toggles it on.

Deleting and Undeleting Text

As text is deleted, the remaining characters in the paragraph are moved to the left. The Cut and Paste commands also delete text.

<Ctrl >+

l Deletes the character under the cursor and moves the text left.

o Deletes characters from the cursor position to the end of the word including the character under the cursor.

e Deletes from the character under the cursor to the end of the sentence.

z o Deletes the characters from the cursor to the end of the paragraph (attribute).

z d Deletes the characters from the cursor to the end of the item.

z z Undo the last delete or data change if the change was made by using one of the control delete keys.

Exiting Items

The <Ctrl>+x commands offer different methods for exiting an item. <Ctrl>+x signals to exit this item and perform one of a series of options. For example, an item can be exited and the information filed (saved), or the item can be exited without the information being filed.

If an item is exited without being filed, the changes made during the editing session are not saved. To save the changes, the item must be filed. Several of the options that exit an item do not file it. If changes have been made to an item during the current editing session and an option is selected that exits the item without filing it, the system double-checks with the user to make sure that the changed information is not to be saved.

All UP exit commands are made up of a minimum of two characters. When the <Ctrl>+x is pressed, the current size of the item, file name, and item-id are displayed on the top line of the CRT. UP waits for an additional command.

<Ctrl>+x+

b Exits the item and goes back to the previous item in the item list. If there is no previous item, the terminal beeps and remains in the current item.

c Files the current item then compiles and catalogs it. This option is used with FlashBASIC programs.

e Exits the item without filing it.

f Files the item, then exit it.

i {(file.reference}{item.id}

Renames and files the new item. This command prompts for a standard file item specification where the item is copied. After the file item specification is entered, the system asks if the old item from which the new item was created, is to be deleted. Enter <Ctrl>+y to remove the old item; otherwise, the item is unchanged.

k Exits the item and returns to calling process. If editing with a select list, the select list is terminated.

n Exits current item and creates a new null item. If a name is not entered when the new item is filed, it can be named using the <Ctrl>+xi> command. If there is no id processing code, UP uses the current date concatenated with a system-wide sequence number as the item-id.

o Exits and deletes the item.

p Files the item and prints it using the Output processor (OP).

r Files the item, then compiles and runs it if compilation was successful.

s Files (save) the item and remains in UP with the current item.

x Exits the item without filing it.

Searching and Replacing Text

The search commands in UP offer the capability to search for text, search for text and replace the text, and search for text and, once the text is found, determine whether or not to replace it.

<Ctrl >+

a string <Ctrl>+m

Search for the first occurrence of the specified string. If a match is found, the cursor is displayed on the first character of the matched string; otherwise, the cursor remains at the current location. When <Ctrl>+a is pressed, UP prompts for the search string at the top of the screen.

The search string is not case sensitive. To search for letters that are case specific, enclose the string in single- or double-quotation marks. If the string itself contains quotation marks, enclose those quotes in quotes.

a Search again for the string specified by the last search command. If a search string is not specified, the terminal beeps and displays a message.

a string1 <Ctrl>+r string2 <Ctrl>+n

Search for and replace every occurrence of string1 with the string2 from the current cursor position to the end of the item. When <Ctrl>+r is pressed, UP prompts for the replacing string. After entering the new text press <Ctrl>+n to indicate all occurrences.

The text is replaced as entered. If the replacement is at the beginning of a sentence, the initial letter is replaced as an upper-case letter. To specify an exact replacement of upper- and lower-case letters, enclose the text in single or double quotation marks.

a string1 <Ctrl>+r string2 <Ctrl>+m

Search for the first occurrence of string1. If a match is found, the cursor is positioned on the first character of the match and UP waits for one of the following commands:

Ctrl +

a Do not replace string1. Search again for next occurrence.

n Replace string1 with string2. Replace all further matches.

r Replace string1 with string2. Search again for string1.

x Stop search. Return to edit mode.

Cutting and Pasting Text

Cut commands are used to move or copy data from one place in an item to another place in an item. Text can be cut and deleted or cut with the original left in place (copied). After the text is cut, it is placed in the cut buffer replacing the previous contents of the cut buffer. The cut text in the cut buffer can be pasted in the current document or it can be pasted in a specified item in a file. Text can also be pasted from a specified item and file.

<Ctrl >+

c d Mark the beginning location of the text to be cut and deleted.

c l Mark the beginning of text to cut and leave (i.e., does not remove the cut from the current location in the text and makes a copy for the cut buffer).

c c Mark end of text to cut or copy from the beginning cut and puts the text in the cut buffer.

c p Paste contents of cut buffer at current cursor position. Once text has been put in the cut buffer it remains there for the duration of the current update session in the UP workspace.

c r {(file.reference}item.id {n{-m}} <Ctrl>+m

Read contents of specified item and paste at current cursor location. The optional n parameter is an integer that specifies the first attribute to copy; the optional m parameter is an integer that specifies the last attribute to copy. If a range is not specified, all attributes from beginning to the end of the item are copied.

c w {(file.reference }item.id <Ctrl>+m

Write the contents of the current item cut buffer to the item specified by item.id.

Prestore Commands

Prestore commands are used to store and execute a sequence of UP commands. These commands can be used during the current update session or saved as an item in a file and recalled for use in subsequent update sessions. Prestore commands can contain both UP commands and text. To specify a UP command, enter the letter portion of the editing command without pressing the <Ctrl> key. To specify text, enter the text enclosed within single/double quotation marks.

Prestores can be created in the prestore buffer or they can be created as an item in a file. Prestore items can exist in any file and must have a p in the first attribute. The following attribute values will contain prestore commands and text.

<Ctrl > +

p Execute the current command in the prestore buffer. If the prestore buffer is empty, displays the next screen of the item being edited (default).

z l Create or edit a prestore command in the prestore buffer. Most UP commands are available to input and edit the prestore buffer.

z r {(file.reference}item.id <Ctrl>+m

Load specified item into the prestore buffer but do not execute it. The first attribute must be the character p.

z r {(file.reference}item.id <Ctrl>+p

Load the specified item into the prestore buffer and execute it. The first attribute in the item must be a p. The default file is the master dictionary.

z w {(file.reference}item.id <Ctrl>+m

Write the prestore buffer into the specified item. The character p is written as the first attribute of the item. The default file is the master dictionary.

Using UP as a Data Processor

update file.reference item.id attribute.list (options

up

u

When UP is called with an attribute list it functions as a data processor. Each attribute name in the UP statement becomes a data entry location on the screen identified by the named attribute. The attribute names are listed along the left of the screen in half-intensity. Dependent attribute names are displayed horizontally across the screen with the controlling attribute. Each attribute-defining item creates a cell on the screen through which data may be accessed and modified

under the control of dictionary processing codes. Since a FlashBASIC program may be called using a processing code, complete application flexibility is provided.

The cursor can be moved, using the previously defined cursor movement commands, from attribute name to attribute name. (Some of the commands function in a different manner. These are defined below.) Data entered or changed next to an attribute name is filed in that attribute when the edited item is saved.

Stored data is subject to output conversions defined in attribute-defining items in the file dictionary prior to display. Data entered using UP is subject to input conversions defined prior to storage on disk. By invoking a single UP command, an entire data access and update screen can be defined. When that command is saved as a macro, it can be recalled any time for file access.UP can be used to scan data using file indexes. When attributes are indexed using the create-index verb, UP can be used to cruise on the index.

The key to cruising is understanding the Balanced Tree (b-tree) indexing feature of D^3. A b-tree is a structure used to keep an ordered list, such as names, and provides the ability to search through the list very quickly with a minimal number of disk I/Os. Any attribute in a file can contain an index. Indexes are defined using a-correlatives. Once the b-tree has been defined, the system automatically maintains the index.

b-trees allow access to data in ways not before possible. The new access capabilities are cruising (using the file index), cruising (using the file related index), zooming, and double clutching.

Cruising using the file index (local file) is the ability to move from item to item in the current file. You can move forward (with <Ctrl>+f or backward (with <Ctrl>+d through the items on any indexed attribute.

Cruising using the related file index (remote file) provides the ability to view valid data from another file. For example, a sales order file could have an attribute called name which points to the entity file. From the orders file one could view the valid names in the entity file and select the one desired.

Double-clutching means the combined abilities of searching the remote file for valid values and also searching the local file for specific items. For example, in an order file the name attribute could have an index pointing to the entity file allowing for searching of valid people. A local index on the name attribute would allow for searching for all of the orders already on file for that person.

Zooming refers to moving through a defined door to another file. Once in the other file, the user has the option to cruise from item to item in the new file, return to the original file or zoom through a door into yet another file. <Ctrl>+g is the command to zoom to another file.

When the cursor is positioned in a cell belonging to an indexed attribute, the commands <Ctrl>+d and <Ctrl>+f can be used to display the data associated with the item containing an adjacent value in the indexed attribute. The command <Ctrl>+f causes the data associated with the next value in the index to be displayed. The command <Ctrl>+d causes the data associated with the previous value in the index to be displayed.

For example, if an item containing customer name and order number indexed attributes is being displayed on the screen and the cursor is at the order number prompt, the <Ctrl>+f command will display the data associated with the next order number in the order number index. If the cursor is residing adjacent to customer name, <Ctrl>+f causes the data associated with the next

customer name in the index to be displayed and customer name is the key. All of the data in the new item associated with the attribute names displayed on the screen is displayed.

When the input conversion attribute in an attribute-defining item specifies an index pointing to another file, that file can be accessed (zoomed to) by using the <Ctrl>+g command. This command pushes the current process one level and accesses the item-id in the pointed to file that matches the current value in the cell where the cursor resides. Attributes to be displayed with that item-id must be specified in attribute 15 of the original attribute-defining item. Several logical views may be defined as macros in Attribute 15 of the d-pointer.

A FlashBASIC program may be called to specify the actual logical view to use for the current data item. It does this by assigning the logical view number into the pseudo-variable "AQL (18)." If no attributes are specified, the entire item is displayed. After viewing the new item the original item can be returned to the screen by a file exit command such as <Ctrl>+xe>.

When the input conversion attribute in an attribute-defining item specifies a second index as well as one pointing to another file, the commands <Ctrl>+u and <Ctrl>+y allow the user to cruise on indexes in the pointed to file. The attribute pointed to in the second file must have been previously indexed by the create-index verb. <Ctrl>+u shows the next attribute value in the pointed to file. Only this value is updated and it is shown in the cell where the cursor is residing. What is displayed is subject to the conversions specified in the dictionary of the first file. The command <Ctrl>+y shows the preceding item-id in the pointed to file. In order for UP to function in this manner, the attribute-defining items must contain the proper index processing codes. The input conversion attribute (att 14) of the attribute-defining item in the file dictionary is where they are specified.

Overloaded UP Commands for Data Files

Most UP commands work the same whether or not dictionary attributes are in control. Certain commands function differently when certain conditions are met. These commands provide the functionality for browsing on indexes. The cruising commands can be used in a double-clutching fashion by specifying the index for the current file and a second index for another file. <Ctrl>+u and <Ctrl>+y cruise on legal instances in the pointed to file staying in the same item. <Ctrl>+f and <Ctrl>+d cruise on other items in the same file using the index for the attribute as a guide.

<Ctrl> +

g When in an attribute which indexes to another file or has a translate code in the input processing code, this command pushes the current process one level and calls UP again with the item pointed to in the new file as the argument. If the macro attribute (attribute 15) in the primary file attribute definition item is used, this list is appended to the update statement. This is known as "zooming".

f On an indexed attribute, the sentence forward command goes to the next sequential index and gets the corresponding item into the UP workspace.

d On an attribute with an index, the sentence backward command goes to the previous sequential index and gets the corresponding item into the UP workspace.

u If the attribute indexes to another file, the word forward command gets the next sequential item from the pointed to file as the value for this attribute.

y If the attribute indexes to another file, the word backward command gets the previous sequential item from the pointed to file as the value for this attribute.

correlative

refers to processing codes that can be defined on the "correlative" dictionary of the file-defining item which affect the item when it is filed.

correlative processing codes (fdi)

defines processing which takes place during the item selection process and during data input and output.

Codes in the "correlative" attribute of the file-defining item are processed at filetime. Available codes include subroutines calls, bridge correlatives, id assignments, update stamps, and indices.

Codes in the "correlative" attribute in the attribute defining items are processed during input and output processing.

Many processing codes can be called on the "output-conversion" attribute of the attribute-defining item, including FlashBASIC subroutines. See the related subjects for a complete list.

When an item is filed using the Update processor, all processing codes defined in the "correlative" attribute of the file-defining item are processed. Items which are not filed using the Update processor use the bridge, index, and callx processing codes only.

cruising

ability to scan forwards and backwards through items in a file.

Cruising on a local file using the file index provides the ability to move from item to item in the current file. In the Update processor, "<ctrl>+f" moves "forward" to the next item in the local index and "<ctrl>+d" moves "backward" to the previous item. The terminal "beeps" when reaching the beginning or end of the index.

A "remote" index provides the ability to logically link a "remote" file to the current file with the capabilities of performing the following:

"table lookup"

To perform a "table"-like lookup of items in the remote index using either a full or partial key, enter the key followed by a "<ctrl>+u" to call up the first match. If no match is found, the next item in the remote list is called up.

"table forward"

To move "forward" to the next item in the remote index, enter a "<ctrl>+u". The terminal "beeps" when reaching the end of the remote index.

"table backward"

To move "backward" to the next item in the remote index or file, enter a "<ctrl>+y". The terminal "beeps" when reaching the beginning of the remote index.

"zooming"

To enter or jump into the remote file using the currently displayed reference point, enter a "<ctrl>+g".

"item binding"

To file or store any "link" modifications to the current item, including changes in its "remote" references that might have occurred using one of the above commands, enter "<ctrl>+xf". (See also "bridges", for maintaining automatic two-way referential integrity.)

A classic situation for using remote indexes, for example, would occur while entering an order item; the order file would likely be linked to an inventory file, and the operator could both "lookup" and/or "table" through the data in the inventory file until the desired item is located. If need be the operator could "zoom" over to the item in the inventory file and examine or modify it, and then return to the current order item. When finished with the order, the operator merely files the item and all its line-item references would be maintained with the order.

Note: There is no restriction from the "remote index" being used on the same file that the current item is also residing in. This would be typical in files where an explicit hierarchy exists between items, such as a "parts.master" file which has a bill of materials, an "entity" file which has both companies and employees, or a "projects" file containing sub-tasks, etc.

"Double-clutching" is a term given when a field contains both a local and remote index. An example of this is using the order example mentioned in the previous paragraph. In addition to the "remote" index to the inventory file, a "local" index could also exist to find the next item in the order file that had ordered a particular inventory item.

Syntax

<ctrl>+f

<ctrl>+d

<ctrl>+u

<ctrl>+y

cursor movement

within the Update processor, the cursor is moved using cursor control command sequences each beginning with the control key being depressed.

The cursor is the underline or block which moves around the screen when characters are typed. Cursors have various forms depending on the terminal: highlighted, blinking, reverse video, etc. In all cases, the cursor indicates where the next character typed is displayed on the screen. Cursor movement commands are used to position the cursor anywhere in the item without altering the text. The cursor actually moves on the screen of the CRT, but since the screen is only a window into the item, the cursor may be moved off the screen forcing a screen redisplay (scrolling).

The following list is a summary of the cursor movement commands, see each token for more information.

<ctrl>+b Moves the cursor back a line

<ctrl>+n Moves the cursor down a line

<ctrl>+u Moves forward a word, also used for cruising

<crtl>+y Moves backward a word, also used for cruising

cut paste

move or copy data from one place in an item to another place in an item.

Text can be cut and deleted or cut with the original left in place (copied). After the text is cut, it is stored in the cut buffer, replacing the previous contents of the cut buffer.

Text in the cut buffer can be pasted into the current item or it can be pasted into a specified item in another file.

Text can also be pasted from a specified item and file.

See the (UP) "c" command for a more detailed explanation.

It is possible to cut and paste when editing the TCL command stack item.

delete text

lists various ways to delete items, attributes, words, characters, and other data elements.

double-clutching

means the combined abilities of searching a remote file index for valid values and also searching the local (current) file index for specific items.

Double-clutching provides a unique feature when the local index is specified before the remote index in the input-conversion of an attribute definition. On a "new" screen, if a new value is entered at the attribute specified for double-clutching, and an item already exists on file containing this same value, that item will be automatically retrieved by the Update processor. For example, consider an Update processor entity screen where the "phone" attribute appears first. If this attribute is set up as described above, and a user enters a phone number that is already on file, that item will be retrieved. In this example, this feature prevents the entry of the same phone number for two different people.

This feature is only activated on new, unchanged items.

Note that specifying the remote index before the local index disables this feature.

exit item

lists several ways to exit an item in the Update processor, all of which begin with "<ctrl>+x".

hotkeys

allows calling FlashBASIC subroutines from dictionaries using the UP command "<ctrl>+xnumber", where "number" is a number between "0" through "9".

There are 11 dictionary attributes in both the file-defining and attribute-defining items related to "hotkeys". They are defined as "hotkey.all", "hotkey0", "hotkey1", etc., through "hotkey9".

Hotkey subroutine calls are available in addition to subroutine calls from "output-conversion", "correlative" or "input-conversion".

To call a FlashBASIC subroutine while in the Update processor on a particular attribute, type "<ctrl>+xnumber".

Only the program(s) on one dictionary will be executed by each command. There may be more than one subroutine called on an attribute. (Use "<ctrl>+v" to multi-value call statements.)

The subroutine that gets executed is dependent on the following set of priorities:

If there is a subroutine called on "hotkey.all" (attribute 20) in the current attribute-defining item, it will be executed, regardless of the value of "number".

If there is not a subroutine called on "hotkey.all", the subroutine called on attribute<20> + "number" of the current attribute-defining item will be executed . Note: If "number" = "0", the subroutine on "hotkey.0" (attribute 30), if present, will be executed.

If there are no subroutines called in the attribute-defining item, the file-defining item is used.

If there is a subroutine called on "hotkey.all" (attribute 20) in the file-defining item, it will be executed, regardless of the value of "number".

If there is not a subroutine called on "hotkey.all", the subroutine called on attribute<20> + "number" of the current file-defining item will be executed . Note: If "number" = "0", the subroutine on "hotkey.0" (attribute 30), if present, will be executed.

The following table illustrates the rules by which hotkey references are interpreted.

Att# Dict.name Subroutine call UP command

<20> hotkey.all call default <ctrl>+x<0-9>

<21> hotkey1 call one <ctrl>+x1

<22> hotkey2 call two <ctrl>+x2

<23> hotkey3 call three <ctrl>+x3

<24> hotkey4 call four <ctrl>+x4

<25> hotkey5 call five <ctrl>+x5

<26> hotkey6 call six <ctrl>+x6

<27> hotkey7 call seven <ctrl>+x7

<28> hotkey8 call eight <ctrl>+x8

<29> hotkey9 call nine <ctrl>+x9

<30> hotkey0 call zero <ctrl>+x0

Example

```
:ud filename test
dictionary-code    a
attribute-count    5
substitute-header
structure
output-conversion
correlative
attribute-type     l
column-width       20
input-conversion
macro
output-macro
description
hotkey.all
hotkey1            call list.states
                   call list.country
hotkey2
hotkey3
hotkey4
hotkey5
hotkey6
hotkey7
hotkey8
hotkey9
hotkey0

In this example, when the operator enters "<ctrl>+x1" from the attribute
"test", the subroutines "list.states" and "list.country" are executed. After
"list.country" completes, control returns to UP and remains in the item
"test".
```

macro

Identifies the default attribute-defining items used for input and output processing of the item.

Output from access verbs use the the "output-macro" attribute-defining item to identify the default attribute-defining items to be used for output display. If the "output-macro" is null, then the "macro" attribute-defining item is used.

In an attribute-defining item, with a remote index to another file, the attribute-defining items listed in the "macro" attribute are automatically passed to UP when "zooming" to the remote file. If no list of attribute-defining items exists in the "macro" attribute, the "default" macro in the target "remote" file is automatically used. In order to 'zoom' (<ctrl>+g) to another file, the "input" attribute-defining item must be defined with either an index or a translate to the remote file.

Example

```
The following example of an attribute-defining item shows the default
attributes to be used by the Update processor when 'zooming' to another file.

ud filename vendor
dictionary-code       a
modulo                1
structure
retrieval-lock
update-lock
output-conversion
correlative
attribute-type        l
input-conversion      ivendor;a1
macro                 vendor.name vendor.address vendor.zip (i
output-macro
output-conversion
description

When in data entry mode on the "vendor" attribute of the specified filename,
when <ctrl>+g is typed, UP 'zooms' to the vendor file and displays the
attibutes 'vendor.name', 'vendor.address', and 'vendor.zip'.

The 'i' option displays the item-id of the file along with the specified
attributes.

If no attribute-defining items are defined, when <ctrl>+g is
typed, UP will use the attributes defined on the macro attribute in the file-
defining item of the 'vendor' file.
```

options: Update Processor

several are available for use with the "update" commands which invoke the Update processor: "u", "up" and "update".

These options affect the data entry environment.

Syntax

u{pdate} file.reference {attr.name(s)} {options}

Options

i Includes the item-id of the file being updated in the UP workspace.

c Clears the screen between each item.

l "look only". The "l" allows data to be viewed only. Item(s) can not be filed.

s Turns the system wide spelling checker off.

r Edits raw text. This causes control characters to display as periods, escape characters to print as "[", and subvalue marks to print as "\".

processing codes

describes data base links and data manipulation rules

The D3 dictionary defines file layouts and data structures. Processing codes are stored in dictionaries to further describe data base links and data manipulation rules. These processing codes are used by various D3 System processors such as UP, FlashBASIC and the list processor.

Processing codes are stored in attribute-defining and file-defining items in file dictionaries. They are found in:

output-conversion - attribute 7

correlative - attribute 8

input-conversion - attribute 14

Note: Processing codes were previously known as "conversions" and "correlatives".

spelling checker

provides an optional spelling checker facility to verify the spelling of text as it is entered.

During text entry using the Update processor, the spelling checker looks at each character to see if those typed so far make a legal word or may become a legal word.

The spelling checker beeps to indicate the word was not found and does not interrupt the entry process.

The words used by the spelling checker are kept in the "dm,words," file.

The spelling checker uses the "speller", "speller-off", and "speller-on" verbs for checking or changing its current state. A "q-pointer" must be present in the current account md to the "dm,words," file and an index must be present on attribute 0 of the "dm,words," file. This q-pointer is usually provided to each new account automatically and can be found in the "dm,newac," file.

Update processor

a dictionary-driven, full-screen editor and data processor.

UP can be used to view, create and manipulate items in files. UP facilitates editing of text, dictionaries and programs, as well as data files (Remember: everything in D3 is an item in a file.)

When used in conjunction with the Output processor (OP), UP operates as a document processor.

When used as a data processor, attribute-defining items in the file's dictionary are employed to verify and translate data as it is entered and retrieved.

B-tree indexes permit cruising from item to item on an attribute value sequence and from value to value in a remote file. See "index, remote" and "index, local".

FlashBASIC subroutines can be called from attribute-defining items and file-defining items allowing absolute control of all data entry using the Update processor and dictionaries.

When evoked, UP copies the specified item to a workspace and displays as much of the item on the screen as it can. (In most applications, each input "screen" fits within a real screen.)

Any unprintable characters embedded within the item are displayed as a dot "." on.

Control keys are used to perform editing functions, but as long as the control key is not pressed, all characters are entered as typed.

An optional spelling checker beeps and rejects input if a character is typed that does not lead to a legal word. See "spelling checker".

UP commands begin with a control character formed by simultaneously pressing the "<ctrl>" ("control") key and one or more alpha keys. Some UP commands require more command characters, while others require text input. The second UP command can be entered as straight alpha or with "<ctrl>" depressed. On some terminals a "<ctrl>+s" leaves the terminal locked. To unlock the terminal, type "<ctrl>+q". See "commands, up".

Until the item is saved, changes made using UP have no effect on the actual item in the file -- only UP workspace is changed. The item can be exited at any time without filing, by typing "<ctrl>+xe", and the item in the file is not changed.

Without any attribute-defining items specified, UP functions as a full-screen editor.

Value marks are indicated by line breaks with no attribute label.

Options

i Include item-id in UP workspace.

l Look only. The item cannot be filed or deleted.

r Raw option. Display non-printable characters as periods. The escape character is displayed as a "[". The subvalue mark is displayed as a "\" character.

s Toggle off the spelling checker.

zoom

see zooming.

zooming

moving through a defined "doorway" to another file.

Once in the other file, the user has the option to "cruise" from item to item in the new file, return to the original file or zoom through a door into yet another file.

"<ctrl>+g" is the command to zoom to another file.

Zooming is only available when done through the Update processor and involves the use of the "i (index, remote)" processing code.

c

clears the screen between each item.

After an item is filed, the screen is cleared prior to the display of the next item.

Syntax

u file.reference {attr.name(s)} (c)

i

includes the item-id of the file being updated in the UP data entry screen.

The item-id appears as the first attribute in the screen, followed by all other defined attributes.

If the item-id is not explicitly requested, it does not appear in the update screen.

The "i" option can be used on the 'macro' attribute of the file-defining item. See 'macro'.

If the "i" option is used on the "output-macro" attribute of the file-defining item, the item is suppressed on output listings.

Syntax

u file.reference {attr.name(s)} (i)

Example

```
From TCL the item-id will appear in the update screen:
u entity name address zip (i

From the output-macro on file-defining item the item-id will NOT appear on
output reports.
output-macro definition  = name address zip (i
list entity
Produces a listing of name address zip with NO item-ids.
```

l

allows data to be viewed only. The item(s) can not be filed.

This option prevents items from being updated or deleted.

Syntax

u file.reference {attr.name(s)} (l)

s

turns the spelling checker off.

For the duration of the Update processor session, the spelling checker is inhibited.

Syntax

u{pdate} file.reference {attr.name(s)} (s)

a

search for a specified text string and replace it with another string if necessary.

The search and replace capabilities allow you to do the following:

- Search for text

- Search for text and replace the text, and

- Search for text and, once the text is found, determine whether or not to replace it.

See "spelling checker" for other uses of "<ctrl>+a". All of the possible command sequences are listed below.

Searching for a string: To search for a string, enter <ctrl>+a as follows:

<ctrl>+a

search for:string<return>

When "<ctrl>+a" is issued, UP prompts with "search for:" at the top of the screen. Enter the "string" followed by a <return>.

If the "string" is found, the cursor is displayed on the first character of the matched "string"; otherwise, the terminal beeps and the cursor remains at its current location and the message "...string not found ..." displays on the terminal.

The search "string" is not case-sensitive. To search for letters that are case-specific, enclose the "string" in single or double quotation marks.

To search for a string containing quotes, the string must be included in alternate quotes. For example, the string "" would find one double-quote, if present in the item.

Repeating the last search: If the previous operation was a simple search and the cursor is still positioned on the most recently found "string", you can enter the <ctrl>+a command to invoke the search again and find the next occurrence of the string. If the cursor has been moved off of the most recently executed search string, the "search for:" prompt reappears.

Searching and replacing all occurrences of a string: If you want to search for a string and replace all occurrences with a replacement string, use <ctrl>+a as follows:

<ctrl>+a

search for:string<ctrl>+r

replace with:replace.string<ctrl>+n

When "<ctrl>+r" is issued, UP prompts with "replace with:". "<ctrl>+n" searches for and replaces every occurence of the "string" with the "replace.string" from the current position of the cursor to the end of the item with no further prompting.

The "string" is replaced as entered; however, if the replacement is at the beginning of a sentence, the initial letter is replaced as an upper-case letter. To specify an exact replacement of upper-case and lower-case letters, enclose the text in single or double quotation marks.

Selective search and replace: To search for a string and selectively replace with a replacement string, use <ctrl>+a as follows:

<ctrl>+a

search for:string<ctrl>+r

replace with:replace.string<return>

In this form, UP searches for the next occurence of the "string". If a match is found, the cursor is displayed on the first character of the matched "string" and UP waits for one of the following four commands:

1) <ctrl>+a

Does not replace the current "string" and locates the next occurence of "string".

2) <ctrl>+n

Replaces current and all further occurences of "string" with "replace.string"

3) <ctrl>+r

Replaces current "string" with "replace.string" and searches for the next occurence of "string".

4) <ctrl>+x

Terminates the search and returns to edit mode.

If none of the above commands are provided at this prompt, the process displays a message on the top line of the terminal indicating the valid responses it will accept.

Searching and deleting strings: To remove all occurrences of "string" without replacement, the following sequence of commands may be used:

<ctrl>+a

search for:string<ctrl>+r

replace with:<ctrl>+n

Syntax

<ctrl>+a

Example

```
Search for "joe's bar", then repeat search.

<ctrl>+a
search for:"joe's bar"<return>
<ctrl>+a

Search and replace all occurrences of "joe's bar" with "sam's pub".

<ctrl>+a
search for:"joe's bar"<ctrl>+r
replace with:"sam's pub"<ctrl>+n

Search for and delete all occurrences of "joe's bar" from current positon of
the cursor to the end of the item.

<ctrl>+a
search for:"joe's bar"<ctrl>+r
replace with:<ctrl>+n
```

a

when the spelling checker is active, accepts the current word as shown in the Update processor.

Syntax

<ctrl>+a

aa

accepts the current word as shown and adds it to the "words" file.

The spelling checker must be active.

Syntax

<ctrl>+a<ctrl>+a

aaa

turns off the spelling checker.

Syntax

<ctrl>+a<ctrl>+a<ctrl>+a

arm

searches and replaces text in an item.

Step one is "<ctrl>+a" which prompts for "search.string". Step two is "<ctrl>+r" which prompts for the "replace.string".

If the "search.string" is found, the cursor is displayed on the first character of the matched string and UP waits for one of the following commands:

"<ctrl>+a"

Does not replace "search.string", searches again for next occurrence.

"<ctrl>+n"

Replaces "search.string" with "replace.string"; replaces all further matches.

"<ctrl>+r"

Replaces "search.string" with "replace.string"; searches again for "search.string".

"<ctrl>+x"

Stops search, return to edit mode.

Syntax
<ctrl>+a search.string <ctrl>+r replace.string <ctrl>+m

arn

searches for and replaces every occurrence of the "search.string" with the contents of the "replace.string" from the current position of the cursor to the end of the item.

When "<ctrl>+r" is typed, UP prompts for the "replace.string".

After entering the "replace.string", enter "<ctrl>+n" to replace all occurrences of the "search.string".

The text is replaced as entered; however, if the replacement is at the beginning of a sentence, the initial letter is replaced as an upper-case letter.

To specify an exact replacement of uppercase and lowercase letters, enclose the "replace.string" in single or double quotation marks.

Syntax
<ctrl>+a search.string <ctrl>+r replace.string <ctrl>+n

b

moves the cursor up one line from the current cursor position.

Syntax
<ctrl>+b

c

moves or copies data from one place to another. The data may reside in the current item being edited, or another item in the current file, or an item in a different file.

Text can be cut and deleted or cut with the original left in place (copied). After the text is cut, it is placed in the "cut buffer", replacing the previous contents of the cut buffer. The text in the cut buffer can be pasted into the current item or it can be pasted into a specified item in another file.

Marking the beginning cut location:

There are two ways to mark the beginning position of text to cut. In both commands, the beginning cut position is determined by the position of the cursor when the command is issued.

1) "<ctrl>+cl" marks the beginning position of text to cut and specifies that the text is to be left in place (copied, not moved).

2) "<ctrl>+cd" marks the beginning position of text to cut and specifies that the text is to be deleted (moved) from its current location after it is cut.

UP marks the beginning of the cut with a backslash ("\") on the terminal.

"<ctrl>+zz" cancels the beginning cut mark.

Marking the ending cut location:

"<ctrl>+cc" marks the ending position of the cut.

The ending cut position is determined by the position of the cursor when the command "<ctrl>+cc" is issued.

Once the ending position has been marked (UP does not display any character for the end mark.), the marked text is placed in the cut buffer.

The entire contents of a value can be saved into the cut buffer with one command. To store the internal contents (the value stored in the item), <ctrl>+ci can be used. To store the external contents (the value displayed on the screen), use <ctrl>+co.

Saving the cut buffer:

The "<ctrl>+cw" command writes the contents of the cut buffer to the item specified.

"<ctrl>+cw" causes the Update processor to prompt with:

cut save-id:

This prompt appears at the top left corner of the terminal. Enter either an "item-id" or a "file.reference" and "item-id".

If only an "item-id" is entered, the cut buffer contents are saved in the current file. If the file already has an item with the specified "item-id", the following prompt displays at the top left corner of the screen :

'cut.id' item exists on file, press "y" to overwrite:

If any key other than "y" is typed, the cut text is NOT saved. If "y" is typed, the item that was on file is overwritten with the cut item.

If the response to the prompt for cut save-id is a "file.reference" and "item-id", the item is written to the designated file with the specified "item-id". A left parenthesis character is required immediately before the "file.reference" and indicates that a different file is being specified.

Pasting from an existing item:

The "<ctrl>+cr" command reads the contents of the specified item and pastes the contents at the current cursor location.

After "<ctrl>+cr" is entered, the terminal prompts with:

paste from:

Any valid "file.reference" and "item-id" are allowed in response to this prompt.

When using UP as a text editor, a range of attribute numbers (separated by a space) may be specified after the "item-id". If the beginning and ending attribute numbers are omitted, the entire item is copied into the item being updated at the current cursor location.

When in data entry mode, with dictionaries defining the attributes, "<ctrl>+cr" has two actions, depending on the status of the item being updated. In both cases, the range is omitted.

1) If the item being updated is new and if the cursor is on the first attribute in the item, "<ctrl>+cr" merges the entire item from the source item specified.

2) In all other cases, when "<ctrl>+cr" is entered, the corresponding (same attribute position) attribute is merged into the current item from the source item specified.

Pasting cut at cursor location:

"<ctrl>+cp" inserts the contents of the cut buffer, beginning at the current cursor position.

Syntax

<ctrl>+c

cc

The "<ctrl>+cc" (UP) command marks the end of a block. If the block was initiated by a "<ctrl>+cd", the text will be deleted from the screen and placed into the cut buffer. If it was initiated by a "<ctrl>+cl" command, the text will simply be placed into the cut buffer, so it can be copied as many times as needed.

The "<ctrl>+cp" command inserts the current contents of the cut buffer at the current screen coordinate. The cut buffer remains intact until the Update processor is exited or another cut is performed.

See the (UP) "c" command for a more complete description.

Syntax
<ctrl>+cc

cd

marks the beginning of a block which will be placed into the cut buffer and deleted from the screen. The next "<ctrl>+cc" command marks the end of the cut and places the "cut" string into the cut buffer.

The cut buffer remains intact until the Update processor is exited or another cut is performed.

See the (UP) "c" command for a more complete description.

The beginning cut position is determined by the position of the cursor when the command is issued.

UP marks the beginning of the cut with a backslash ("\"). To cancel this command use the "<ctrl>+zz" command.

Syntax
<ctrl>+cd

ci

copies the internal contents of the current value into the cut buffer. This is mostly used in attributes with conversions. For example, if the cursor is at a value displaying "Pick Systems", and this is a translating attribute where the internal value is "ps", the string "ps" will be copied into the cut buffer. To copy the external contents ("Pick Systems"), see the "<ctrl>+co" command.

The "<ctrl>+cp" command inserts the current contents of the cut buffer at the current screen coordinate. The cut buffer remains intact until the Update processor is exited or another cut is performed.

Syntax
<ctrl>+ci

cl

marks the beginning of a block which will be copied ("left"). The next "<ctrl>+cc" command marks the end of the cut and places the "cut" string into the cut buffer.

The cut buffer remains intact until the Update processor is exited or another cut is performed.

See the (UP) "c" command for a more complete description.

The beginning cut position is determined by the position of the cursor when the command is issued.

UP marks the beginning of the cut with a backslash ("\"). Once the ending position has been marked (UP does not display any character for the end mark), the marked text is placed in the cut buffer.

Syntax
<ctrl>+cl

co

copies the entire contents of the current value into the cut buffer. This is mostly used in attributes with conversions.

The "<ctrl>+cp" command inserts the current contents of the cut buffer at the current screen coordinate. The cut buffer remains intact until the Update processor is exited or another cut is performed.

Syntax
<ctrl>+co

correlative processing codes (adi)

lists the processing codes available.

Note: called Pick/BASIC subroutines may be Flash'ed to improve performance.

cp

inserts the current contents of the cut buffer at the current screen coordinate.

The contents of the cut buffer remain present until the Update processor is exited or another cut is performed.

See the (UP) "c" command for a more complete description.

Syntax
<ctrl>+cp

cr

reads from an item and pastes the contents at the current cursor location.

See the (UP) "c" command for a more complete description.

When using UP as a text editor, a range of attribute numbers (separated by a period ".") may be specified after the item-id. If the beginning and ending attribute numbers are omitted, the entire item is copied into the item being updated at the current cursor location.

When in data entry mode, with dictionaries defining the attributes, "<ctrl>+cr" has two actions, depending on the status of the item being updated. In both cases, the range is omitted.

1) If the item being updated is new and if the cursor is on the first attribute in the item, "<ctrl>+cr" merges the entire item from the source item specified.

2) In all other cases, when "<ctrl>+cr" is entered, the corresponding (same attribute position) attribute is merged into the current item from the source item specified.

Syntax
<ctrl>+cr{(file.reference} item-id {start.attribute}{.end.attribute}<return>

cw

writes the current contents of the cut buffer.

The contents of the cut buffer remain present until the Update processor is exited or another cut is performed.

"<ctrl>+cw" causes the Update processor to prompt with

cut save-id:

at the top left corner of the terminal. Enter either an item-id or a file.reference and item-id.

If only an item-id is entered, the cut buffer contents are saved in the current file. If the file already has an item with the specified item-id, the following prompt displays at the top left corner of the screen :

'cut.id' item exists on file, press "y" to overwrite:

If any key other than "y" is typed, the cut text is NOT saved. If "y" is typed, the item that was on file is overwritten with the cut item.

If the response to the prompt for cut save-id is a file.reference and item-id, the item is written to the designated file with the specified item-id.

Syntax
<ctrl>+cw

d

moves the cursor back one sentence when in text entry mode.

In an indexed attribute, "<ctrl>+d" retrieves the previous item in the index.

At TCL, "<ctrl>+d", when typed at the beginning of the TCL command, retrieves the previous command in the user's tcl-stack.

Syntax
<ctrl>+d

data-entry

allows repetitive entry of new items.

Using this connective precludes the need to file an item using "<ctrl>+xf". After the last value on the screen has been entered, "return" files the item and brings up a new item.

Example
```
u entity data-entry
```

e

deletes text from the current cursor position to the end of the sentence.

A sentence is defined as a series of words terminated by a period.

If OP commands are contained in a sentence, the system assumes that the period at the beginning of the OP command is the end of the sentence.

As text is deleted, the remaining characters in the paragraph are moved to the left.

An alternate method of deleting one or more sentence involves the use of the cut and paste features. See the (UP) "c" command.

Syntax

<ctrl>+e

f

moves the cursor forward one sentence in text entry mode.

In an indexed attribute, "<ctrl>+f" retrieves the next item from the index.

From TCL, "<ctrl>+f" retrieves the next command in the user's tcl-stack.

Syntax
<ctrl>+f

g

positions to the end of an attribute, or zooms to a linked file.

Case 1: positions the cursor at the end of the current attribute value (paragraph).

Case 2: if the attribute is set-up for zooming and the cursor is positioned at the end of that attribute value (which is also the case on a null value), <ctrl>+g pushes a level and uses the zoomed-from value as the item-id in the zoomed-to file effectively executing a 'u file.name item-id'.

The macro field in the current attribute-defining item may be used to override the default attribute list defined in the zoomed-to file's dictionary.

Syntax
<ctrl>+g

h

backspaces the cursor.

A backspace moves the cursor back (left) one character and replaces the character with a blank. On ASCII terminals the backspace key actually generates a <ctrl>+h. If the cursor is at the end of an attribute, the character is deleted but not replaced with a blank.

Syntax
<ctrl>+h

i

moves the cursor to the next tab position.

The "<ctrl>+zt" command is used to define the tab positions.

Syntax
<ctrl>+i

id-prompt

causes UP to include the item-id attribute in the data entry screen.

The item-id prompt appears as the first attribute in the screen, followed by all other defined attributes. If the item-id is not explicitly requested, it does not appear in the update screen. The item-id prompt can be used on the 'macro' attribute of the file-defining item. See 'macro'.

The "i" AQL Option is a synonym for id-supp.

Syntax
update file.reference {attr.name} {attr.name...} id-prompt

Example

```
From TCL

u entity name address zip id-prompt

From 'macro' attribute in file-defining item

name address zip (i

From 'macro' attribute in file-defining item

name address zip id-prompt
```

input-conversion

lists processing codes available from the input-conversion.

Any of the processing codes listed under "see.also" may be used as an input conversion.

input-conversion

contains codes used to perform pre-processing functions such as inserting default data.

The subroutine is executed after the item is retrieved from the file, but before the item is displayed on the screen.

See the "call" processing code for specific information about using a FlashBASIC program in an input-conversion.

See the FlashBASIC "access" and "system" functions for information about features useful from within a "call" processing code.

insert mode

toggles "insert/replace" mode and is used to insert new text before the character at the current cursor position.

When text is entered in the insert mode, the following text is "opened" and moved to the next line until insert mode is toggled, or a cursor movement command is typed.

The letter "I" is displayed at the top right corner of the screen when in "insert" mode. An "O" is displayed in "replace" (overtype) mode, which is the default state.

As text is entered in either mode, UP word-wraps the text as necessary. That is, if a character string is longer than the remaining space on the current line, the entire string is moved to the next line. This allows typing continuously within a paragraph (attribute) without pressing the "<return>" key.

A new attribute (paragraph), may be created by pressing "<return>" while in insert mode.

When UP is first entered, it is automatically in "overtype" (replace) mode. Any characters entered will be typed "over" the characters on the screen.

Syntax

<ctrl>+r

j

positions the cursor left one character, but does not replace or erase the character.

The "<ctrl>+h", or backspace key, by contrast, moves the cursor left and erases the character.

Syntax

<ctrl>+j

k

moves the cursor one character position to the right.

Syntax
<ctrl>+k

l

deletes one character from the current cursor position.

As text is deleted, the remaining characters in the paragraph are moved to the left. If the cursor is on the last character of a paragraph, the "<ctrl>+l" command deletes the attribute mark.

Syntax
<ctrl>+l

m

terminates the current paragraph and prompts with the next attribute number or moves to the next attribute.

When UP is in insert mode, but not being controlled by a dictionary, "<ctrl>+m" starts a new attribute. When UP is in insert mode, and being controlled by a dictionary, "<ctrl>+m" moves to the next attribute. When UP is in overwrite mode, "<ctrl>+m" moves to the next attribute.

If the cursor is on the last line of an item, "<ctrl>+m" creates a new paragraph, in either overtype or insert modes.

Syntax
<ctrl>+m

n

moves the cursor down one line.

Syntax
<ctrl>+n

o

deletes a word beginning with the current cursor position.

A word is defined as a string of characters terminated by a blank or a period.

As text is deleted, the remaining characters in the paragraph are moved to the left.

Syntax
<ctrl>+o

only

permits editing the "raw" item(s), without using any of the inherent attribute-defining items.

When a file does not contain default attribute-defining items, the "only" mode is automatically implied.

Syntax
u{pdate} only file.reference

Example

```
u only entity *
```

output-conversion

codes defined in the "output-conversion" attribute, are processed during input and output processing.

overtype mode

means that characters typed overwrite the characters on the screen.

When UP is first invoked, it is in overtype mode.

The letter "O" is displayed at the top right corner of the screen when in overtype mode. The letter "I" is displayed when in insert mode.

UP places the cursor at the upper left corner of the item (top) on the first character of the first value of the first attribute and waits for input. Once the item has been loaded, any character typed overwrites what is on the screen.

As text is entered in either mode, UP word-wraps the text as necessary. That is, if a character string is longer than the remaining space on the current line, the entire string is moved to the next line. This allows continuous typing within a paragraph (attribute) without pressing the <return> key.

"<ctrl>+r" toggles between overtype and insert modes.

To create a new attribute (paragraph), press <return> while in insert mode. To return to overtype mode, press "<ctrl>+r" again.

Syntax

<ctrl>+r

p

executes the command stored in the prestore buffer.

If there is no command in the prestore buffer, "<ctrl>+p" advances the cursor to the next page. This is equivalent to the "<ctrl>+zn" command.

When working on an "active list", if "<ctrl>+p" is the last command in a prestore command, the entire prestore command is repeated until the list is exhausted.

See the (UP) "zl" command for defining prestored commands.

Syntax

<ctrl>+p

Example

```
The example below shows how to create a prestore command which will be used to
search and replace text in a list of items.

get-list items

retrieve previously saved list

u document

invoke UP on document file

<ctrl>+zl
```

```
command to define prestore

a'old'r'new'nxfp<return>

actual prestore command

<ctrl>+p

execute the prestore command

The prestore command used in the example does the following:

a   UP command to search
'old'  text to search for
r      UP command to replace
'new'  text to replace with
n      UP command to replace all occurrences
xf     UP command to file an item
p      UP command to repeat prestore command
```

prestore command

defines a series of Update processor commands which can be saved and recalled for later use.

Once defined and saved, prestore commands can be used during the current update session or recalled and used in subsequent update sessions.

There is one active prestore buffer available for each update session. This buffer can be re-used as many times as necessary. Prestore commands can be loaded into the buffer and modified using UP cursor movement commands.

The default prestore buffer contains the command "zn", which advances the cursor to the next page. Note: the default buffer contents, do not display when editing the prestore buffer.

There is no limit to the number of prestore commands that are available, because prestore commands are saved as items in a file. The items remain in the file until they are deleted or overwritten.

"md" is the default file used for saving prestore commands, but any legal file.reference may be used.

Prestore commands may contain both UP commands and literal text. When literal text is included as part of the command, the text must be enclosed in single or double quote marks.

When a UP command is included as part of the prestore command, the UP command is typed in WITHOUT depressing the control key. The Update processor interprets anything in the command that is NOT in quote marks as a UP command.

r

toggles the Update processor mode between "overtype" and "insert" modes.

Upon initial entry, UP is is "overtype" mode.

Syntax

<ctrl+>r

search replace

provides search, search and replace globally, or search and selectively replace capabilities.

See 'arm' and 'arn' for more details.

Syntax
See the (UP) "a" command.

t

positions the cursor at the top of the item or document in UP.

From TCL, "<ctrl>+t" positions the cursor at the beginning of the line.

Syntax
<ctrl>+t

u

moves the cursor forward one word in text entry mode.

In data entry mode on an indexed attribute, "<ctrl>+u" retrieves the next item from the remote index.

Syntax
<ctrl>+u

UP Commands

provides a link to all subjects related to the Update processor.

v

inserts a new value.

The value mark causes any characters that follow it in that attribute to be moved to the next line.

A value mark has an ASCII decimal value of 253 and is not displayable in UP.

The "v" processing code is used to limit the number of values that are allowed on an attribute. When the limit is reached, the terminal beeps and no more values are allowed. See "(v value limit)".

It is possible to enter multi-values as stacked data or multiple commands at TCL.

Syntax
<ctrl>+v

w

inserts a single space

This command is useful when inserting a single character (or even a few characters). It is sometimes faster than changing from overtype to insert mode.

Syntax
<ctrl>+w

x

the beginning of the command sequence used to exit items or invoke "hotkeys".

Items can be exited from UP with or without filing.

"<ctrl>+xnumber" (control<x> + a number between "0" and "9") invokes the corresponding "hotkey" command, which calls a FlashBASIC subroutine. See "hotkeys'.

A list of all UP exit commands follows:

<ctrl>+xb

If the current update session is using an "active list", this command exits the current item and returns to the previous item in the list.

If there is no previous item, the terminal beeps and remains in the current item.

<ctrl>+xc

Used when updating FlashBASIC programs, this command files, compiles and catalogs the program.

<ctrl>+xe

This command exits an item without filing any changes. This is used when viewing an item without making any changes.

If changes were made during an update session and "<ctrl>+xe" is used to exit the item, the terminal prompts with:

item has changed, press "y" to exit or "f" to file:

<ctrl>+xf

This command files an item and then exits the item. Any changes that were made during the current update session are saved and the previous copy of the item (if present) is overwritten.

Items can be filed automatically using the "data-entry" connective. See "data-entry".

<ctrl>+xi

This command allows the item being updated to be renamed as it is filed, and optionally deletes the original item.

After entering this command, the terminal prompts with:

Rename item; new{(filename} item-id:

After the item-id and optional filename is entered, the terminal prompts with:

delete current item (y/<cr>=n)?

To delete the current item, answer "y". "n" or <return> leaves the current item on file and no changes are saved from the current update session.

<ctrl>+xk

If the current update session is using an "active list", this command exits the current item and returns to the calling process. The "active list" is abandoned.

<ctrl>+xl

This command files and compiles a FlashBASIC program.

<ctrl>+xn

This command exits or files the current item and creates a new null item.

If the item has been changed since being entered, the following message appears:

'item has changed, press "y" to exit or "f" to file'

The new item can be assigned an item-id by either the 'id' processing code, or by filing the item with the "<ctrl>+xi" command.

If neither of these methods of assigning an item-id are used, the system assigns the item-id based on the current date concatenated with a system-wide sequence number.

<ctrl>+xo

This command deletes the item being updated.

If any changes were made to the item during the update session, the terminal prompts with:

item changed, delete it (y/n=cr)?.

See "recover-item".

<ctrl>+xp

This command files the current item and prints the item to the currently assigned form queue, in Output processor (OP) format.

<ctrl>+xr

This command files (and compiles, if the item is a FlashBASIC program) and executes the item being updated.

This command can be used to run macros, PROCs, menus, and FlashBASIC programs.

<ctrl>+xs

This command files the current item and remains in the item.

<ctrl>+xx

This command exits the item without filing any changes. This is equivalent to the "<ctrl>+xe" command.

Syntax

<ctrl>+x

x0

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey0'.

If a subroutine has been defined on 'hotkey0', the subroutine is executed when the UP command "<ctrl>+x0" is executed from within an item in the Update processor.

Syntax

<ctrl>+x0

x1

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey1'.

If a subroutine has been defined on 'hotkey1', the subroutine is executed when the UP command "<ctrl>+x1" is executed from within an item in the Update processor.

Syntax

<ctrl>+x1

x2

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey2'.

If a subroutine has been defined on 'hotkey2', the subroutine is executed when the UP command "<ctrl>+x2" is executed from within an item in the Update processor.

Syntax

<ctrl>+x2

x3

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey3'.

If a subroutine has been defined on 'hotkey3', the subroutine is executed when the UP command "<ctrl>+x3" is executed from within an item in the Update processor.

Syntax
<ctrl>+x3

x4

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey4'.

If a subroutine has been defined on 'hotkey4', the subroutine is executed when the UP command "<ctrl>+x4" is executed from within an item in the Update processor.

Syntax
<ctrl>+x4

x5

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey5'.

If a subroutine has been defined on 'hotkey5', the subroutine is executed when the UP command "<ctrl>+x5" is executed from within an item in the Update processor.

Syntax
<ctrl>+x5

x6

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey6'.

If a subroutine has been defined on 'hotkey6', the subroutine is executed when the UP command "<ctrl>+x6" is executed from within an item in the Update processor.

Syntax
<ctrl>+x6

x7

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey7'.

If a subroutine has been defined on 'hotkey7', the subroutine is executed when the UP command "<ctrl>+x7" is executed from within an item in the Update processor.

Syntax
<ctrl>+x7

x8

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey8'.

If a subroutine has been defined on 'hotkey8', the subroutine is executed when the UP command "<ctrl>+x8" is executed from within an item in the Update processor.

Syntax
<ctrl>+x8

x9

invokes a FlashBASIC subroutine called from the dictionary attribute 'hotkey9'.

If a subroutine has been defined on 'hotkey9', the subroutine is executed when the UP command "<ctrl>+x9" is executed from within an item in the Update processor.

Syntax
<ctrl>+x9

xb

exits the current item and returns to the previous item in the list.

If there is no previous item, or there is no active list, the terminal beeps and remains in the current item.

Syntax
<ctrl>+xb

xc

files, compiles and catalogs the FlashBASIC source item.

Syntax
<ctrl>+xc

xe

exits the current item without filing any changes.

This is used when viewing an item without making any changes.

If changes were made during an update session and "<ctrl>+xe" is used to exit the item, the terminal prompts with:

item has changed, press "y" to exit or "f" to file:

If neither a "y" nor an "f" are typed, no action takes place and control returns to the last cursor position in the current item.

Syntax
<ctrl>+xe

xf

files and exits the item.

Items can be filed automatically using the "data-entry" modifier.

Syntax
<ctrl>+xf

xi

allows the item being updated to be renamed as it is filed, and optionally deletes the original item.

After entering this command, the terminal prompts with:

Rename item; new{(filename} item-id:

After the item-id and optional filename is entered, the terminal prompts with:

delete current item (y/<cr>=n)?

To delete the current item, answer "y". "n" or <return> leaves the current item on file and no changes are saved from the current update session.

Syntax
<ctrl>+xi

xk

exits the current item and returns to the calling process.

If editing with an active list, the select list is terminated.

Syntax
<ctrl>+xk

xl

files and compiles the FlashBASIC source item.

Also see the "xc" and "xr" commands.

Syntax
<ctrl>+xl

xn

files or exits the current item and creates a new item.

If the item has been changed since being entered, the following message appears:

'item has changed, press "y" to exit or "f" to file'

The new item can be assigned an item-id by either the 'id' processing code, or by filing the item with the "<ctrl>+xi" command.

If neither of these methods of assigning an item-id are used, the system assigns the item-id based on the current date concatenated with a system-wide sequence number.

Syntax
<ctrl>+xn

xo

deletes the item being edited.

If any changes were made to the item during the update session, the terminal prompts with:

item changed, delete it (y/n=cr)?.

Syntax
<ctrl>+xo

xp

files and prints an item.

Using current formqueue and assignment options.

Syntax
<ctrl>+xp

xr

files (and compiles, if the item is a FlashBASIC program) and executes the item being edited.

This command can be used to run macros, PROCs, menus, and FlashBASIC programs.

Syntax
<ctrl>+xr

xs

files and does not exit the item being edited.

Using the "<ctrl>+xs" command, followed by the "<ctrl>+xe" command would produce identical results to the "<ctrl>+xf" command.

This is like the (old) Editor "fs" command.

Syntax
<ctrl>+xs

xx

exits the current item without filing any changes.

This is used when viewing an item without making any changes.

If changes were made during an update session and "<ctrl>+xx" is used to exit the item, the terminal prompts with:

item has changed, press "y" to exit or "f" to file:

If neither a "y" nor an "f" are typed, no action takes place and control returns to the last cursor position in the current item.

This is the same as the (UP) "<ctrl>+xe" command.

Syntax
<ctrl>+xx

y

moves the cursor left one word in data-entry mode or moves to the previous entry in a remote file.

In text-entry mode, "<ctrl>+y" moves the cursor back one word.

On an indexed attribute, <ctrl>+y goes to the previous sequential index in a remote file and gets the corresponding item into the UP workspace. See 'cruising'.

In order to use a remote index, the index must first be created using the "create-index" verb. See "create-index".

To use a remote index, an index correlative must be present on the input-conversion dictionary. See "i (index, remote)".

Syntax
<ctrl>+y

z

beginning character for several miscellaneous commands.

These commands are listed below:

<ctrl>+za

In text-entry mode, this redisplays the item with the current line at the top of the terminal.

From TCL, "<ctrl>+za" searches forward in the "tcl-stack" file for a specified string.

<ctrl>+zb

Redisplays the item, with the current line at the bottom of the terminal.

<ctrl>+zc

Displays the terminal's columnar positions at the top of the terminal.

<ctrl>+zd

Deletes text from the cursor position to the end of the item.

<ctrl>+ze

Moves the cursor to the end of the item.

<ctrl>+zg

Moves the cursor to the end of the previous paragraph.

<ctrl>+zh

Moves the cursor to the first line of the next screen, then moves the display up "n" lines, where "n" is equal to the current terminal depth minus 12. (In other words, down 1/2 page.)

<ctrl>+zl

Creates or edits a prestore command in the prestore buffer. See "zl".

<ctrl>+zn

Advances the cursor to the beginning of the next page.

<ctrl>+zo

Deletes text from the cursor position to the end of the attribute or paragraph.

<ctrl>+zp

Redisplays (refreshes) the data on the screen.

<ctrl>+zq

Moves the cursor to the first line of the next screen, then moves the display up one quarter of a page.

<ctrl>+zr

Loads a previously saved prestore command into the prestore buffer and, optionally, executes the prestore command. See "zr".

<ctrl>+zs

Toggles the spelling checker on/off. See "zs".

<ctrl>+zt

Sets the location of tab stops. It prompts with the message: "Enter Tab stops:". Tab stop positions may be entered, with each numeric parameter being separated by a space.

<ctrl>+zw

Writes the contents of the prestore buffer to a file for later use. See "zw".

<ctrl>+zy

Moves the cursor back one page.

<ctrl>+zz

Undoes the last delete or data change if the change was made by using one of the control delete keys.

This is also used to cancel a "cut" command in UP. See "cut paste".

<ctrl>+znumber

Moves the cursor to the attribute or line number specified. For example, to go to line# 5:

<ctrl>+z5<return>

The given number may be preceded by a "+" (plus sign) to move forward (down) the specified number of attributes or lines, or it may be preceded by a "-" (minus sign) to move backward (up) the given number of attributes or lines. For example, to move forward three lines:

<ctrl>+z+3<return>

Syntax

<ctrl>+z

z number

moves the cursor to the attribute or line number specified.

The line number may be preceded by a "+" (plus sign) to move forward (down) the specified number of attributes or lines, or it may be preceded by a "-" (minus sign) to move backward (up) the given number of attributes or lines.

Syntax

<ctrl>+znumber

Example

```
<ctrl>+z5<return>
This moves to line five of the current item.

<ctrl>+z+3<return>
This moves forward three lines.
```

za

redisplays the item with the current line at the top of the terminal or searches forward in the "tcl-stack" for a specified string.

The (UP) "<ctrl>+za" command serves 2 purposes:

In text-entry mode, it redisplays the item with the current line at the top of the terminal.

At TCL, "<ctrl>+za" searches forward in the "tcl-stack" file for a specified string.

Syntax

<ctrl>+za

zb

redisplays the item, with the current line at the bottom of the terminal.

Syntax

<ctrl>+zb

zc

displays the terminal's columnar positions at the top of the terminal.

Syntax
<ctrl>+zc

zd

deletes all text between the current cursor position to the end of the item.

Syntax
<ctrl>+zd

ze

moves the cursor to the end of the item.

Syntax
<ctrl>+ze

zg

moves the cursor to the end of the previous paragraph.

Syntax
<ctrl>+zg

zh

moves the cursor down a about half page.

Specifically, "<ctrl>+zh" moves the cursor to the first line of the next screen, then moves the display up "n" lines, where "n" is equal to the current terminal depth minus 12.

Syntax
<ctrl>+zh

zl

edit a command in the prestore buffer.

After typing in the command "<ctrl>+zl", the Update processor displays the following prompt at the top left corner of the terminal screen:

P:

The prestore command (followed by "<return"> or "<enter>") is typed in immediately following this prompt.

"<ctrl>+p" is used to execute the prestore command.

Syntax
<ctrl>+zl

Example
```
<ctrl>+zl               command to define prestore
P:                      terminal prompt for command
r'inserting'r<return>   actual prestore command
```

zn

moves the cursor to the beginning of the next page.

This is the default command loaded into the default prestore (the "<ctrl>+p" command each time the Update processor is invoked.

Syntax

<ctrl>+zn

zo

deletes all text between the current cursor position and the end of the paragraph or attribute.

Syntax

<ctrl>+zo

zp

redisplays the data on the screen.

Syntax

<ctrl>+zp

zq

moves the cursor to the last line on the screen.

This is done by printing "n" lines of data, where "n" is the screen length-18, then repositioning the cursor to the first new line of data printed.

Syntax

<ctrl>+zq

zr

loads a previously saved prestore command into the prestore buffer and, optionally, executes the prestore command.

After the "<ctrl>+zr" command has been entered, the following prompt appears at the top left corner of the screen:

Prestore id:

The response to this prompt can either be an item-id (if the prestore command is stored in the current master dictionary) or a file.reference and item-id (if the prestore command is stored in a file other than the current master dictionary.)

The item-id is followed by either "<return>" or "<ctrl>+p".

When the "<ctrl>+zr" command is followed by a "<return>" or "<enter>", the contents of the prestore command are loaded in the prestore buffer but the command is not executed. When the "<ctrl>+zr" command is followed by a "<ctrl>+p", the prestore command is executed.

Once the prestore buffer has been loaded, it can be executed any time during the current update session using "<ctrl>+p".

Syntax

<ctrl>+zr{(file.reference} item-id <return>

<ctrl>+zr{(file.reference} item-id <ctrl>+p

Example

```
The following example shows how to load the prestore buffer with a saved
prestore command from the current master dictionary without executing the
prestore command.

<ctrl>+zr               command to load prestore buffer
Prestore id:            terminal prompt for item-id
id<return>              item-id from md
```

```
The next example shows how to load the prestore buffer with a saved prestore
command from the file 'commands' using the item-id of 'holdit'. The command is
immediately executed using <ctrl>+p.

<ctrl>+zr                command to load prestore buffer
Prestore id:             terminal prompt for item-id
(commands,holdit<ctrl>+p  file.reference,item-id, execute
```

zs

toggles the spelling checker on or off.

There are several other ways the spelling checker can be turned off, including from TCL or from a macro.

The spelling checker is only applicable to those items who have an attribute type of "w" in their attribute-defining item.

Syntax

<ctrl>+zs

zt

sets the location of tab stops.

It prompts with the message:

Enter Tab stops:

Tab stop positions may be entered with each numeric parameter being separated by a space.

Syntax

<ctrl>+zt

zw

writes the contents of the prestore buffer to a file for reuse.

"<ctrl>+zw" causes the Update processor to prompt:

Prestore id:

This displays at the top left corner of the terminal. At this time an item-id may be specified or a file.reference and item-id may be specified.

If only an item-id is entered, the prestore item is saved in the current master dictionary (md). If the md already has an item with the specified item-id, the following prompt displays at the top left corner of the screen :

'prestore.id' item exists on file, press "y" to overwrite:

If any key other than "y" is typed, the prestore item is NOT saved. If "y" is typed, the md item that was on file is overwritten with the prestore command.

If the response to the prompt for Prestore id is a file.reference and item-id, the item is written to the designated file with the specified item-id. The format of the saved item is :

attribute 1 = p

attribute 2 = prestore command

Syntax

<ctrl>+zw{(file.reference} item-id<return>

Example

```
The following example shows a prestore command composed of only a literal
string, with no UP commands. Since there is no file.reference specified in
response to the Prestore id prompt, the command is saved in the md.

:u document document.id
invoke UP to edit document
<ctrl>+zl
command to define prestore
P:
terminal prompt
'This is text'<return>
actual prestore command
<ctrl>+zw
command to save prestore
Prestore id:
terminal prompt for item-id
id<return>
user-defined item-id


The next example shows a prestore command consisting of UP commands and text.
This command does the following :

mm
issues two carriage returns
r
puts UP in insert mode
'This is text'm
Text to insert, plus a <ctrl>+m (return)
r
puts UP back in overtype mode

The command is saved in the file 'commands' with the id of 'holdit'.

u document document.id
invoke UP to edit document
<ctrl>+zl
command to define prestore
P:
terminal prompt
mmr'This is text'mr<return>
actual prestore command
<ctrl>+zw
command to save prestore
Prestore id:
terminal prompt for item-id
(commands holdit<return>
user-defined file and item-id

The file commands 'holdit' contains the following :

01 P
02 mmr'This is text'mr
```

zx

restores a value to its original contents, providing that the line has not been exited.

Syntax

<ctrl>+zx

zy

moves back a page

zz

undoes the last delete or data change.

This is also used to cancel a "cut" command in UP. See "cut paste".

Syntax

<ctrl>+zz

Unix

Boot Error Codes

describes the boot error codes, displayed in the message: Boot aborted D3 process terminated. Line 0. Code X (0xNNNNNNNN)

where:

X is a decimal value

0xNNNNNNNN is the hexadecimal value of X

Name/Value	Description/Note
>0 Signal number from 1 to 22.	
	Normally, all signals are caught by the process. However, uncaught signals terminate the process. Normally all Unix signals are caught by the monitor. If this error happens, it is probably because a user-written C function restored the signal handler routine to SIG_DFL.
-n (1< n < 1000)	Instruction diagnostic error. Optionally, the monitor instruction diagnostics are executed at boot time by the line 0. In case of error, the process is terminated with a negative error code. The absolute value is the instruction diagnostic number.
p_dskerr/-1000	Disk error. Unrecoverable. Check Configuration File. Common cause for this error is a disk size too small (third argument to the disk statement in the configuration file).
p_ioerr/-1001	Terminal I/O error. Using the following command, check read/write permissions to your terminal:
ls -l /dev/ttyXX	
p_maxfid/-1010	MAXFID has changed. The monitor required confirmation, but the user decided to abandon. A file restore is necessary. This error happens when choosing the option 'X' on a disk which has no file or has damaged files. Check the Configuration file to make sure the size of the disks or the number of disks has not been changed.
p_sysbase/-1011	SYSBASE has changed. The monitor required confirmation, but the user decided to abandon. This error occurs if the number of PIBS is changed beyond the number allowed by the license. A file restore will be necessary.
p_flushst/-1012	The flush process has terminated, upon receiving a 'logoff' signal.
p_halt/-1013	Monitor HALT. The halt code and the halt address are displayed. Note the code and address.
p_nopib/-1014	No PIB available on the virtual machine. The allowed maximum number of users on the system has been reached. No new user process can be started.
p_twoflsh/-1015	Flush process already active.

p_vabserr/-1016	ABS is invalid or not loaded. Do an ABS restore (option 'A') and retry boot.
p_defect/-1017	Defect table has not been initialized. Delete disk and reload abs and files.
p_defectr/-1018	Defect table can not be read. Disk needs to be reformatted and both abs and files reloaded.
p_bactkey/-1019	An error occurred while trying to activate the machine. Possible reasons are:
	Installer does not agree with terms.
	Invalid activation key entered.
	Attempt to activate more pibs than license allows.
p_badsksz/-1020	The size of the disk in the configuration file is invalid. Check the size and change the configuration file.
p_ra/-2001	Global MCB address error. The monitor is incompatible with the Unix version. A different monitor is required for Unix System V Release 4.

Halt Error Codes

describes the halt error codes, displayed in the message: "Monitor halt code XX at 0xNNNNNNNN"

where

XX is a decimal value

NNNNNNNN is the address where the HALT occurred.

Some errors are trapped in the Monitor Debugger, with a message:

<HLT> YY

(register dump)

H!

YY is the hexadecimal value of XX.

All HALT errors are logged in the D^3 Unix error log. See the documentation on the perrpt command.

Code (hex) Description

1 (01) Machine abort (bus error, segmentation violation, etc...). This error often indicates an ABS corruption. Run the TCL verify-system command. To exit the Monitor Debugger, type 'g' <return> to continue and enter the Virtual Debugger.

14 (0E) Process workspace corruption. This error occurs when a process workspace is so badly corrupted that it cannot even enter the Virtual Debugger. To recover from this error, use the TCL command reset-user port.number to clean up the bad process from another terminal.

16 (10) Old process. When the flusher which controls the virtual machine is killed by Unix (or by a 'kill -9' command), in case of low paging space, for example, some old processes may linger around. When the virtual machine is restarted, these processes will halt when trying to

access any resource. This error does no harm to the virtual machine and can be ignored. When in the Monitor Debugger type g<return> to continue.

Monitor Debugger

tool for recovery in case of a system crash, following, for instance, a power failure.

The Monitor Debugger allows:

- Display and change D3 virtual memory.

- Display and change 'real' memory (remember that 'real' memory is, in fact, Unix virtual memory).

- Force a flush of the D3 memory space back to disk.

- Display, change the status of the system semaphores, to remove a dead lock situation.

- Get access to a locked system.

- Terminate processes, including doing a shutdown.

- Trace modifications to a memory area.

- Put low level break points in the virtual code.

IMPORTANT

When the Monitor Debugger is entered unexpectedly, try first to type g<return> to see if the system restarts. If not, hit the <BREAK> key again to examine the problem. There are cases when the debugger is entered wrongfully. For instance, when some specially long, un-breakable, tape operations (like a rewind) are running, hitting the break key several times on line 0 may enter the Monitor Debugger with a 'tight loop' condition, which means the process was engaged in a 'long' operation, preventing it from servicing the <BREAK> key.

Entering the Debugger

The debugger can be entered:

- Voluntarily, by hitting the <BREAK> key on a D3 process which has been started with the -D option, to enable the Monitor Debugger.

- Voluntarily, by setting a monitor trace on real or virtual memory.

- Voluntarily, by setting a monitor break point in virtual memory.

- Following a system abort. When a serious system abort occurs, the debugger is entered. The user cannot continue from such a condition.

- Following a Monitor HALT. When a process cannot continue execution, a HALT is executed on this process. This normally does not affect the other processes. The faulty process enters the Monitor debugger and waits. Type 'x' to display the hardware registers, note them to transmit them to Technical Support and type 'g' to try restarting the process. If the process aborts again, try a logoff and/or reset-user from another terminal before trying 'g'. If it fails again, type 'q'.

- By hitting the <BREAK> key 5 times in less than 5 or 6 seconds on the line 0 when the system does not respond (system stuck in a tight loop, in a semaphore dead lock or line 0 comatized). When a semaphore is left hanging, or when the processor enters a short tight loop, due to an ABS corruption, for example, the process does not respond any more and is incapable of going to the Virtual Debugger. The fifth time the <BREAK> key is pressed, the signal handler checks to see if the first occurrence of the break was serviced normally. If it is not, the debugger is

entered. On a busy system, it might be necessary to try the <BREAK> sequence several times to get to the Monitor Debugger.

- By hitting the <BREAK> key on the line 0 when it waits for a system lock (overflow lock, spooler lock, etc...) for more than approximately 5 seconds.

The different causes of entry in the debugger are displayed by a message on entry and a special prompt, as defined in the table below:

CONDITION MESSAGE PROMPT

Break key on line

started with -D <BRK> B!

System abort <ABT> A!

Monitor trace <TRC> addr C!

Break point <BPT> bp# I!

Monitor HALT <HLT> code H!

Break key on line 0

on tight loop <TLP> T!

Break key on line 0

on virtual lock <VLK> V!

Referencing Data

Data can be referenced from the Monitor Debugger either in the virtual space or in the real memory space.

Data Specifications

Data location is defined by the following format:

address{;window}

The data is always displayed in hexadecimal.

Virtual Address Specification

The address of a virtual element can be represented by:

[r reg|{.}fid][.|,]disp

The base FID is either the content of the register reg or a FID number fid in decimal or in hexadecimal, prefixed by a dot. The displacement is either expressed in decimal, prefixed by a comma, or in hexadecimal, prefixed by a dot.

For example:

1.300 Offset x'300' in frame 1.

.12,16 Offset 16 in frame x'12'.

r3.100 Offset x'100' off the location pointer at by register 3.

Monitor Address Specification

The address of a Monitor element can be represented by either of the two following forms:

{ [l|g] }.hexaddress{ [+|-] {.}offset}

/symbol{ [+|-] {.}offset}

The l prefix is used for local data. The g is used for global data.

The second form requires the presence of the file sdb.sym on the current directory or on /usr/lib/pick. This file is normally not shipped with the system. It is reserved for development purpose.

The optional offset which is added to, or subtracted from, the n base address is either expressed in decimal, or in hexadecimal if prefixed by a dot.

For example:

.40000100 Absolute address.

l.0 First address in the local data space.

g.100+.10 Offset x'10' off the address x'100' in global data space.

/sys.time Address of symbol sys.time.

/tcb0+.100 Offset +x'100' off the symbol tcb0.

Window Specification

The window specifies the number of bytes to display. The window is expressed in decimal or in hexadecimal, prefixed by a dot. The default window size is 4. When using a symbolic name, the window is set automatically.

Changing Data

When a window of data is displayed, it is followed by an equal sign = . Hitting Ctrl-N will display the next window, if available, and Ctrl-P the previous one. New data can then be entered as follows:

'char Character Insertion. A character string is preceded by a single quote. The characters in the display window are replaced by those in the input string, beginning from the left.

.hex Hexadecimal string insertion. A hexadecimal string is preceded by a dot. It must contain only hexadecimal characters and an even number of nibbles. The characters in the display window are replaced by those in the input string, beginning from the left.

{+|-}int Integer. The display window is treated as a numeric element. The window must be 1, 2 or 4 byte long. The new integer replaces all data in the window.

Debugger Commands

The debugger prompts for a command with a one character code followed by an exclamation mark (!). Commands are terminated by a carriage return.

!shell Submit a Shell command. The command is submitted to shell.

? Display help information. The Unix file /usr/lib/pick/sdb.help is displayed using pg.

ba{.}fid[.|,]disp

bao{.}offset

ba+{.}offset

b{n} Add a break point. The effective address can be specified in different ways:

Form 1: The effective address is computed by adding the argument fid to the fid break point offset, and disp to the displacement break point offset defined by the bo command.

Form 2: The effective address is specified by the fid be defined by in break point offset and offset added to the break point offset displacement set by the bo command.

Form 3: The effective address is defined by the current value of R1 to which the offset is added.

Form 4: The effective address is defined by the current value of R1 to which the n * 4 is added. If n is not specified, 1 is assumed. This form is used for architectures which have a fixed 4 byte instruction size (RISC).

The address must be at a virtual address boundary. Break points are global for the whole virtual machine. Once the break point is set, any process which hits it will stop. Break points are removed once they are encountered. A break point should not be set into an ABS frame which has been write required (mloaded into). Up to three break points can be set simultaneously. Upon successful setting, a '+' is displayed and the break point is displayed. Break points remain in action until they are explicitly removed or until the virtual machine is shut down.

bd[*|n] Delete break points. If * is used, all breakpoints are removed. If n from 0 to 2 is used, the specified break point is deleted.

bl List break points. List the Monitor break points.

bo{.}fid[.|,]disp Break point offset. Define a fid and displacement which are used in computing the effective address of a break point in the ba command.

d? Show/Change default dump string. In case of system abort (bus error, segmentation violation, ...), the system automatically dumps some critical elements in the file "/usr/tmp/ap.core". This file can be examined with the apcrash utility. After the display, the string can be changed by typing the new dump string after the '=' sign. The dump string can have up to 15 characters, one-character codes and arguments. See the Monitor Debugger command d below for the description of each code. The dump string can be different for each process. See the Pick Systems Reference Manual documentation for the description of 'apcrash'.

dcommand.string Dump D^3 core memory to the Unix file /usr/tmp/ap.core. The result of the dump can be examined with the utility apcrash. This command can be used, after an incident, to dump selected elements of the current D^3 memory to be able to investigate the problem. See the Pick Systems Reference Manual documentation for the description of 'apcrash'. The Unix file can be dumped to tape using Unix utility like tar (eg. tar cv /usr/tmp/ap.core). The content of the dump is controlled by command.string which is composed of one character codes, some followed by arguments. Codes can be separated by commas for readability. The order of the codes in command.string is unimportant. This dump utility is automatically invoked in case of system abort (bus error, segmentation violation, etc. ...) before entering the Monitor debugger. What is dumped in this case is controlled by a default dump string (see the Monitor Debugger command d? above). Valid codes are:

a : Dump 'all'. This option is equivalent to the command string "l,g,b,p,r,c,0,f1". See the description of each code below.

0 : Dump the PCB of the current process. If the PCB is not attached at this point, this performs no operation.

b : Dump the buffer table.

c : Dump the current process context. All the frames currently attached to the process, their forward and backward links are dumped.

f{.}n : Dump the fid n. If the specified frame is not in memory, it is read from disk.

g : Dump the global space.

l : Dump the local (private) memory, not including the stack.

p : Dump the pibs.

r : Dump the hardware registers. The registers are dumped in the same order as described in the Monitor debugger 'x' command described later in this appendix.

s{.}start;[*|{.}size] : Dump the main shared memory segment. Start is the starting offset, expressed in bytes and size is the size expressed in KILOBYTES. If * is used instead of size, the entire shared memory segment, starting at the specified offset, is dumped.

v{.}n : Dump n virtual buffers.

e Toggle the debugger ON/OFF. When OFF, prevent the entry to the debugger with the <BREAK> key. On line 0, though, the debugger will be entered in some special cases (See section 'Entering the Debugger' above) even when the debugger is disabled.

f{!} : Flush memory. All frames modified in memory are written back to disk. If a disk error occurs, a minus sign is displayed. If the '!' option is specified, all frames in memory, even if they are not write required, are written to disk.

g {fid.disp} : Go. Without any argument, the process resumes execution. If fid.disp is specified, control is transferred to the specified mode. fid is expressed as a relative offset in the current abs.

gl{-} : This command displays or removes group locks. With no options, the monitor prints the status of the global group lock (with a "G+" or a "G-"), and scans memory for any frames which are marked as locked. For each locked frame, the monitor displays the fid, the address of the buffer table entry, and all group lock information held in the frame itself. To clear all group locks, type "gl-". To clear a specific group lock, type "gl-{fid}". Note that locks cleared from the monitor debugger may still display with the "list-locks" command. Such locks should be cleared with a "clear-locks" command when the virtual machine becomes accessable. In general, group locks should always be cleared by TCL commands only. The monitor debugger should only be used when system access is denied due to a lock set on a critical file (like the "mds,," file).

h fid : Hash Fid. This command displays internal information about the specified FID if it is in memory, or the message <NIM> if the fid is not in memory. The content of the buffer table can be altered. Input is terminated by:

carriage return : return to debugger.

^N : Next buffer table entry, following the age queue forward link.

^P : Previous buffer table entry, following the age queue backward link.

^F : Next buffer table entry, following the hash queue forward link.

k{w}[f| pib] : Kill. Terminate the process associated to the PIB pib or the flusher if used with the key f by sending a SIGTERM to it. The w key waits up to 10 seconds for the process to terminate. If it does not terminate, a SIGKILL is sent to it. Note that if the target process is in

the Monitor Debugger or stuck on a semaphore, the SIGTERM signal will have no effect until it leaves the Monitor debugger or the semaphore is released. Kill the flusher ('k{w}f') will unconditionally log all processes off and shut down the virtual machine.

l fid : Display/Modify Link fields. Displays in hexadecimal the link fields of the frame fid, in the following format:

nncf:frmn:frmp:npcf:clnk=

nncf: number of next contiguous frame(s)

frmn: forward link

frmp: backward link

npcf: number of prior contiguous frame(s)

clnk: core link

New values for the fields can then be entered, separated by commas, with an empty field to leave a field untouched.

m {*}monitor.address{;window} : Display/Change Real Memory. Display the specified window at the real address as specified (see the section about Monitor address specification above in this section). If an asterisk (*) is used, the address is considered as a pointer and its content is used as the monitor address. The length and window specification applies to the area pointed at by the pointer.

IMPORTANT: Access to an illegal address will cause a segmentation violation or a bus error sending control back to the Monitor Debugger with an abort condition, from which it is impossible to recover. It is strongly advised to avoid absolute addresses, since they vary from implementation to implementation.

p {pib}{[.|,]offset}{;window} Display/Change PIB. Display window bytes in the pib specified by pib (current pib if pib is omitted), at the optional offset.

q{!} : Quit. Quit Monitor debugger. Confirmation is asked. Leaving the Monitor Debugger terminates the D3 process. When asked to confirm, the user must type y (no return). The optional '!' by-passes the normal D3 termination, and terminates the process abruptly. This forms should be used only in extreme situations where even quitting from the Monitor Debugger aborts.

r reg{.disp}{;window} : Display data through register. Displays data pointed at by the register reg from 0 through 15. If specified, disp is added to the register displacement.

c [*|sem]{[?|+|-]} Display/Change semaphore status. Display or change the semaphore specified by sem, expressed from 1 through 3, or all semaphores if the key * is used. The key + sets (locks) the specified semaphore. The key - resets (unlock) the specified semaphore. The key ? displays the information as in the example below:

00: O pid=0985

01: O pib=0023 W

02:

03: O pib=001A

where semaphore 0 is owned by the process with the pid number of x'0985' - Only semaphore 0 is displayed with the Unix pid number instead of the D3 pib number; semaphore 1 is busy,

owned by the pib x'23' and has at least one process waiting on it (W); semaphore 2 is free; semaphore 3 is busy, owned by process x'1A' but has no process waiting on it. If the owner pib is not setup yet when the command was executed, the "pib=?" is displayed. Re-enter the command again to see to owner pib.

S{f}{h}{i}{m}{s}{w}{-} Scan buffer table bits. This command displays and/or clears buffer table bits depending upon certain criteria. The options are as follows:

f Referenced bit

h Hold bit

i iobusy bit (disk read)

m Temporary mlock bit

s Suppress detail output. Show only total.

w Write-required bit

- Clear instead of display

The user specifies which bits to search for using the above options. When a buffer is found, it is displayed in a manner similar to that of the "h" command. The user may go backwards or forwards in the selection list with the CTL-P and CTL-N commands. At the completion, the total count of items is indicated. Note that the count is only accurate if forward movement only is used.

t{[mmonitor.address|fid.disp]{;window}} Set/remove Monitor trace. Without any argument, any pending monitor trace is removed. A minus sign is displayed in acknowledgment of the removal. Else, set a trace on the specified area of memory starting at monitor.address or the area of memory associated to fid.disp with a length equal to window. If no window is specified, the default window or the size of the monitor element is used. The maximum window size on a monitor address is 32767. The maximum window on a virtual address is the frame size. A plus sign is displayed in acknowledgment of the setting. The memory is checked for any change at every virtual branch or call, and every frame fault. If the memory is changed, the Monitor debugger is entered. When setting a trace on a virtual address, the frame is locked in memory. Removing the trace unlocks the frame if it was not locked when the trace was set.

v{code} Enter the Virtual Debugger with a code 'code'. If 'code' is not specified, it just enter the debugger as if the <BREAK> key had been hit. 'code=14' will log off the process. This command will display 'ADDR' and fail if the Virtual Debugger PCB is not set up.

x Display hardware registers. Display on the first line the program counter, followed by a variable number of 32 bit registers. The information is implementation dependent:

AIX: Registers r3 through r31.

SCO: Registers edi esi ebp esp ebx edx ecx eax

HP-UX: Registers r2 through r30

SINIX (MIPS): Registers r1 through r30

SVS (ICL DRS6000): Registers %pc %npc %g1 %o %l %i

y{!} Toggle Lock By-Pass. When ON, the process under debugger will by-pass all locks in the system, monitor and virtual. This option should be used very carefully, since it can create extensive damage if used on a live system. To be used safely, all other processes should be either logged off or stopped by setting a semaphore (see section 'Usage Hints' below). Unless used with the key !, the user has to confirm the activation of this by-pass.

Usage Hints

This section shows how to use the Monitor Debugger to perform some unusual actions. Extreme care must be exercised when using the debugger to remove a lock or a semaphore. This may cause data loss. It is strongly recommended to contact Technical Support when a system gets locked.

- Running in single user. To prevent all users from running, except one terminal:

At shell, activate a D^3 process in the Debugger.

ap -D <return>

Once in the Monitor Debugger:

B! s1+ <return>

B! y <return> and confirm by typing 'y'

B! g <return>

All processes will now be locked, except the one running under the debugger and the flusher. This is useful when patching some critical virtual structures, like the Overflow table.

To restart the multiuser activity:

Break into the Monitor Debugger (either on the line which has the lock set, or on line 0 by hitting <BREAK> twice, since the line 0 should be locked by the semaphore).

This enters the Monitor Debugger.

B! y <return> to toggle the by-pass off

B! s1- <return>

B! g <return>

All processes will now be unlocked.

- Removing a dead lock. When a process has been killed by the system, it may have left a semaphore or a virtual lock behind. To remove the lock, make sure all users are inactive, and do the following:

On the line 0, hit <BREAK> up to six times, in less than 5 or 6 seconds. The process should drop into the Monitor Debugger, with the message <TLP> in case of a tight loop, or <VLK> in case of a virtual lock. Do the following, depending on the case:

T! f <return>

T! s*? <return>

Note which semaphore number is set and remove it by the command:

T! s semnum - <return>

T! g <return>

If the lock is a virtual lock, it may have to be cleared. This may have disastrous results if done without some inside information. It is strongly advised to contact technical support.

V! f <return>

V! r15;2 <return>

The system will display a number. If non zero, zero it. Else, there is another problem. Contact technical support.

V! r15;2 .001A= 0 <return>

V! g <return>

A virtual lock might also be one of the system wide locks. Do the following to identify it:

V! f <return>

V! 1.100;2 <return>

The system will display a number, normally 0. Type <ctrl> N 10 times or until a non zero value shows. If no null value shows, there may be another problem. Contact technical support.

V! 1.100 .0000= Ctrl N

V! 1.102 .001A= 0 <return>

V! g <return>

A virtual lock might also be an item lock. Do the following to identify it:

V! 1.15a;6 <return> <return>

The system should display a frame number. Now type the following:

V! .(frame number displayed before).0;4 <return>

The system displays an 8 digit hex number. The first 4 digits indicate the global lock, while the next 4 digits indicate the number of item locks. To zero both of these, type "0" at the "=" prompt followed by a <return>.

A virtual lock might also be a group lock. Do the following to identify it:

V! gl <return>

If the system prints anything other than "G-" then group locks are set. To clear them, type the following:

V! gl- <return>

V! gl <return>

The system should report "G-" after the last command. Note that the virtual machine may still report locks set if the "list-locks" command is used. To fix this, go back to tcl as follows:

V! g <return>

If the system does not go to a tcl prompt, then some other lock condition exists and the user should contact technical support. If a tcl prompt appears, type the following:

: clear-locks (g <return>

Allow some time for "clear-locks" to complete as it can be delayed by processes which have been terminated abnormally. After completion, the group locks should be cleared. If any further deadlocks are encountered, contact technical support.

If only the line 0 is stuck, it might be because it has been accidentally comatized. The WHERE command shows the first two characters of the status field as FE or 7E. A logoff from another port will un-comatize the line 0, or do the following:

T! p.0;1 <return>

T! p.0;1 .7E= .ff <return>

T! g <return>

- Enabling the debugger on line 0. If the line 0 has been started without the -D option, it is impossible to get in the Monitor debugger, unless there is a lock. To enable the debugger, proceed as follows, to set temporarily a virtual lock to be able to drop in the debugger, enable it and restart:

At TCL:

: debug <return>

! 1.102;2 <return>

! 1.102;2= -1 <return> This sets the overflow lock

! g <return>.

Line 0 (and the WHOLE system) is now locked. Wait 5 seconds and hit the <BREAK> key. This drops in the debugger.

V! 1.102;2 <return>

V! 1.102;2 .FFFF= 0 <return>

V! e <return>

V! g <return>

The Monitor Debugger is now enabled on line 0.

- Finding your line number. To determine the PIB number on which the debugger is running, do the following:

Break into the debugger.

B! p.18;2 <return>

This displays in hexadecimal the pib number +1

- Restarting the virtual machine after an abort early in the boot stage. If the virtual machine boot aborts very early (during or right after the 'Diagnostic' message), after having corrected the error when possible, the Boot can be restarted quickly by doing (the line 0 must have been started with the '-D' option):

! Hit the <BREAK> key to enter the Monitor debugger

B! g3.0 <return>

This should redisplay the message 'Diagnostic ...' and proceed.

Precautions

A process started with the -D option has some special privileges, which might lead to data destruction if used indiscriminately:

- Access to the virtual machine will always be granted, even if the Initialization lock is set. In particular, it would be possible to start a user process while the line 0 is in the process of initializing the virtual machine. Therefore, always make sure that the line 0 has reached, at least, the 'Diagnostics...' stage before starting a process.

- A process with debugger privilege may by-pass all locks, including virtual locks! When changing data structures, be sure that nobody is accessing the virtual machine, or, better, set a semaphore, as shown above, to prevent concurrent access. For the same reason, use debugger only on one line at a time.

- When the memory is full, the debugger can abort with the message 'MEM FULL'. Do a flush and retry until it succeeds.

- When the system runs in 'single user' (with a lock set by the command s1+) and with the lock by-pass activated, do not try to shut down the system with the TCL command shutdown. Instead, go to the monitor debugger, do a flush (f) and kill the flush process (kf). This terminates all active processes.

- When changing virtual memory with the debugger, it is a good idea to flush memory frequently, using the f command.

Performance Monitoring

see performance monitoring (Definitions)

Unix Signal Usage

examines Unix signals in relationship to the D3 Virtual Machine.

This discussion is intended for experienced Unix programmers to interface Unix applications with D3 or to use Unix process synchronization in a purely D3 environment.

For performance reasons, Unix signals are not used in regular D3 activity. They are used solely to handle exceptions. Diverting signal usage for an application would simply remove nonessential facilities, like the possibility to send a message or to logoff a process. This does not mean that signals can be used freely, though. The D3 Monitor provides a system call to process regular D3 signals inside a customized signal handler, thus combining system and application signal handling.

The following list, details the D3 signal system call and gives some examples of application signal handler.

Signal Usage

The following is a table of the signals currently used by D3. Future development may require using more signals.

D3 uses only Unix System V.3 signals, which are common to more recent Unix versions. Non Unix System V.3 signals are left to their default signal handler (normally SIG_IGN) as set by the system.

Numerical values may vary between Unix implementations. It is very important to use the proper include when writing in C or the include dm,bp,unix.h, signal.h when writing FlashBASIC programs using signals.

Signal Description

SIGHUP Hangup. When received, the process is logged off D3, but remains connected to the virtual machine. The signal is controlled by the hupcl command in the D3 configuration file. It is normally generated by the TTY device driver when data carrier is lost. This signal can be altered by the TCL command trap dcd command .

SIGINT Interrupt. Generated by the BREAK key and the alternate Virtual Machine. The break character is defined by brkchr in the configuration file or by the TCL command set-break . The process is sent to the FlashBASIC, or system debugger, if brk-debug is on or a level is pushed if brk-level is on.

SIGQUIT Quit. Generated by the ESC key or the alternate escape character as defined by the escchr in the configuration file or by the TCL command set-esc . A level is pushed if esc-level is on. If esc-data is on, a signal is still generated, but eventually ignored by the Virtual debugger. The character defined by set-esc will eventually be put in the process input buffer, but it must be emphasized that a noticeable delay can be experienced when inputting this character.

SIGILL Illegal instruction. Sends the process to the system debugger with an 'illegal opcode' abort. Used for assembly single stepping on some implementations.

SIGTRAP Trace trap. Sends the process to the system debugger with an 'illegal opcode' abort. Used for assembly single stepping on some implementations.

SIGIOT I/O trap instruction. Sends the process to the system debugger with an 'illegal opcode' abort.

SIGEMT Emulator trap. Sends the process to the system debugger with an 'illegal opcode' abort. Used for assembly single stepping on some implementations.

SIGFPE Floating point exception. Sends the process to the system debugger with an 'illegal opcode' abort.

SIGKILL Kill. This signal cannot be caught and will terminate the process immediately after doing a logoff. Killing the flusher of a virtual machine will log all processes off and shut down the virtual machine.

SIGBUS Bus error. Sends the process to the system debugger with an 'illegal opcode' abort. Used for assembly single stepping on some implementations.

SIGSEGV Segmentation violation. Sends the process to the system debugger with an 'illegal opcode' abort. Used for assembly single stepping on some implementations.

SIGSYS Bad argument to a system call. Sends the process to the system debugger with an 'illegal opcode' abort.

SIGPIPE Write to a pipe with no one to read it. The process is sent to the Virtual debugger. Pipes are sometimes used as 'tapes' to do file save. The other end of the pipe can be connected to a serial device, a communication server process... Therefore, when this signal is raised, the write system calls aborts. In an application environment, this signal should be caught and a message sent to the user saying 'Communication Server not Ready', for instance. The write system call

would then be interrupted, reported to the save process as a 'parity' error, and the write would be retried until the write finally succeeds.

SIGALRM Alarm clock. The default signal handler is a simple return. Therefore, the signal is active, but does nothing. It can be used in FlashBASIC, with the %alarm() function, do set time outs around critical sections. This signal can be altered by the TCL trap alrm command.

SIGTERM Software termination signal. This signals terminates the D3 process, leaving it logged on, but disconnected. This signal 'freezes' the D3 environment. If the process is restarted, execution will resume where it stopped. Normally, this signal should be preceded by a SIGHUP to log the process off.

SIGUSR1 User-defined signal 1. Not used. Can be used freely by the applications.

SIGUSR2 User-defined signal 2. Force the process to examine its current state, possibly changing it. This signal is used for D3 message, by the TCL logoff , TCL , END commands. Sending this signal out of context will cause no harm. This should be the way, for instance, to wake up a process waiting in a %pause() system call. This signal can be used by applications, provided they call the D3 signal handler inside their customized signal handler. See the section about the examples below.

SIGCLD Death of a child. Not used.

SIGPWR Power fail. When received, the process is logged off D3, but remains connected to the virtual machine. The behavior of the system depends on how the hardware generates this signal. If there is no battery back-up, it is unlikely that the virtual machine will be brought down gently. The system usually change its running state, which will control what happens. A ap - k command should be inserted in the appropriate shell script associated to the new run level. This signal can be altered by the TCL trap pwr command.

D3 Signal System Call

The D3 monitor provides a system call to inform the D3 Virtual environment of an incoming signal. This C function is normally called by the default signal system handlers, but it can be called by an application signal handler too.

This function is defined as follows:

#include "/usr/lib/pick/include/sigmon.h"

void sigmon(code)

int code;

code is the action code, as described in the following table. Note some functions do not return to the caller .

Code Value/Description

db_slcout 0/No operation. Simply forces the process to examine its current state. This function must be called by the SIGUSR2 signal handler.

db_privop 7/Privileged opcode. Sends the caller to the system debugger. This call does not return to the caller. This call can be used to abort a C program, even if not called from a signal handler.

db_break 10/Interrupt. Sends the process to the FlashBASIC, or system debugger. This function must be called by the SIGINT signal handler. Going to the system debugger is not immediate. Normally, the process will enter the debugger (or push a level) when the signal handler terminates.

db_esc 12/Quit. Pushes a level . This function must be called by the SIGQUIT signal handler. Normally, the process will push a level when the signal handler terminates.

db_logoff 14/Log off. This function is normally called by the SIGHUP handler. It can be called by other signal handlers as part of the application (like SIGALRM for instance, to log off a process after a given time).

db_pwrdown -1/Disconnect. The process is disconnected from the virtual machine, but remains logged on.

Examples

This sections shows some simple examples of using signals in a D3 application, or cooperating with Unix applications. All these examples require writing a C signal handler, which must be linked with the D3 monitor, as detailed in this document.

Time out

The purpose is to write a signal handler which will log the user process off if there is no activity for a given time. It is assumed the user is in a FlashBASIC application, waiting input.

- Write the following signal handler and function

setalrm.c:

```
#include <signal.h>
#include "/usr/lib/pick/include/sigmon.h"
myalrm(){
/* signal handler. When activated,
log the process off */
sigmon( db_logoff );
}
/* Set the new signal handler
Set the signal handler to
the application signal handler.
Return the address of the
previous signal handler, so
that the basic application
can remove the time out
*/
void (*setalrm())(){
return signal(SIGALRM, myalrm );
}
```

<

- Compile the previous program and incorporate it to the D3 monitor, by typing, in the dm account:

addbi setalrm signal

ar vru libgm.a setalrm.o

make -f /usr/lib/pick/Makefile

Note this incorporates the user-defined function setalrm() as well as the system call signal() not normally shipped with the Monitor. The latter will be extracted from the standard C library. The %setalrm function can then be called from FlashBASIC.

- Enter the following 'application' program

```
*
* Reprogram the alarm handler,
* keeping the previous system
* handler
* Note the function pointer
* returned by the function
* is treated as a pointer to a character.
include dm,bp,unix.h signal.h
old$alarm = (char*)%setalrm()
*
loop
* Set the alarm to 10 minutes
* From now on, if the user
* doesn't enter a command before
* 10 minutes, the process will
* be logged off.
%alarm(600)
* .. display application entry screen here
crt 'Command : ':
input command
while 1 do
begin case
case command = 'a'
* ... put command 'a' routine
case command = 'end'
print 'end of the application'
* disable the alarm and set the default alarm
```

* handler back to the default.

%alarm(0)

%signal(SIGALRM, (char*)old$alarm)

chain 'off'

end case

repeat

Note that, in the example above, if the alarm was not disabled and if the signal handler is not reset to the default, the alarm clock would have continued to run, even if the process went back to TCL, thus logging it off later. A small FlashBASIC program, just rearming the alarm can be written to incorporate the alarm in PROCs or menus to achieve the same time out capability in TCL.

Synchronizing Two D3 Processes

On conventional D3 implementations, synchronizing two D3 processes is done either through polling for the existence of a file or, a little better, by using a FlashBASIC LOCK instruction. On most implementations, however, the FlashBASIC LOCK instruction itself involves, internally, a polling, thus making the waiting process to consume cpu resource. By using signals, it is possible to have a process waiting for a signal to occur (a BREAK, logoff or a signal generated by another process, possibly a Unix process or a D3 process from another D3 virtual machine) without any cpu consumption.

The following example has one phantom task waiting for a signal sent by 'somebody' to go read a message in a pipe where the 'somebody' has deposited a message. This communication with message, probably, would be better achieved through Unix messages, but this is merely an example. Also, this is a one to one communication. Multiple partners in this scheme would require semaphores. The server process makes itself visible to the entire system (D3 and Unix) by writing a temporary file which contains its process id, which other processes use to send a signal to it.

The (simple) signal handler uses SIGUSR1 . The 'message' is in ASCII, containing a header and some data. The message has the following format:

```
+---+---+--------//-----------+
|   |NL |                      +
+---+---+--------//-----------+
|   |   |
|   |   +- Data (code dependent)
|   +------------ New line (discarded)
+---------------- Command code.
T : Terminate
M : Message
R : file receive

- Enter the following C signal handler:
setusr1.c:
#include <signal.h>
#include "/usr/lib/pick/include/sigmon.h"
myusr1(){
/* Just reset the signal for next time */
signal( SIGUSR1, myusr1 );
}
/* Set the new signal handler
```

```
Set the signal handler to the application signal handler.
Return the address of the previous signal handler
so that the basic application can remove the time out
*/
void (*setusr1())(){
return signal(SIGUSR1, myusr1 );
}
- Enter the following phantom task code in the item dm,bp, pserver :
pserver
001 !
002 * Example of a file server running as
003 * a phantom
004 * Commands are received on a named pipe.
005 *
006 include dm,bp,unix.h signal.h
007 include dm,bp,unix.h mode.h
008 include dm,bp,unix.h fcntl.h
009 *
010 * Defines:
011 equ NL to char(10) ;* Unix line terminator
012 equ AM to char(254)
013 equ PIPE to "/dev/mypipe"
014 equ BUFSIZE to 10000 ;* max message size
015 *
016 char buffer[BUFSIZE]
107 *
018 * Remove temporary file
019 execute '!rm -f /tmp/pserverid'
020 *
021 restart:*
022 *
023 * Open the communication pipe in read
024 * only, no wait on read
025 * If open error, stop, sending a
026 * message back to the originating
027 * process.
028 fd=%open(PIPE, O$RDONLY+O$NDELAY)
029 if fd<0 then
030 stop 1,"Cannot open '":PIPE:"'. Error ":system(0)
031 end
032 *
033 * Do not allow BREAK from now on
034 break off
035 *
036 * Create the temporary file which
037 * contains our pid, so that
038 * everybody on the system will
039 * know the server id
040 mypid=%pgetpid(-1)
041 execute "!echo ":mypid:" > /tmp/pserverid"
042 *
043 * Reprogram the alarm handler,
044 * keeping the previous system handler
045 * Note the function pointer returned
046 * by the function is treated as
047 * a pointer to a character.
048 old$usr1 = (char*)%setusr1()
049 *
050 * Go try read the pipe. If there is no
051 * message, the signal we received
052 * was not an application one. May be
053 * somebody tried to log us off
```

```
054 * so close the pipe, restore the
055 * signal and wait a bit. If we come
056 * come back, was a false alarm. Restart.
057 loop while 1 do
058 * Wait for a signal
059 %pause()
060 *
061 * Read the pipe
062 n=%read(fd, buffer, BUFSIZE )
063 *
064 begin case
065 case n=-1
066 * IO error
067 gosub terminate
068 stop 1,"PSERVER: IO error ":system(0)
069 case n=0
070 * Pipe is empty...
071 * The signals we got is not
072 * the one we expected. Enable
073 * break to allow for a
074 * possible
075 * logoff, and restart
076 gosub terminate
077 sleep 1
078 goto restart
079 case 1
080 * Got something (n is the
081 * number of byte)
082 * The 1st byte is a code,
083 * the second a new line
084 * (discarded)
085 code=buffer[1,1]
086 message=buffer[3,n-2]
087 * Go handle the message
088 gosub do$it
089 end case
090 repeat
091 *
092 do$it:*
093 begin case
094 case code='T'
095 * Terminate
096 gosub terminate
097 stop 1,"PSERVER: Terminated"
098 case code='M'
099 * Message.
100 execute "msg * PSERVER: ":message
101 case code='R'
102 * Receive a file. The message is
103 * full filename (NL) item (NL)
104 * text
105 convert NL to AM in message
106 filename=message<1>
107 item=message<2>
108 message=delete(message,1)
109 message=delete(message,1)
110 open filename then
111 write message on item
112 close
113 end else
114 execute "msg !0 PSERVER: '":filename:"' is not filename"
115 end
```

```
116 end case
117 *
118 return
119 *
120 * Clean everything
121 terminate:*
122 %close(fd)
123 %signal( SIGUSR1, (char*)old$usr1 )
124 execute '!rm -f /tmp/pserverid'
125 break on
126 return
127 *
128 end
```

- Create a communication pipe by doing the following from TCL:
su (must be superuser to do the following)
mknod /dev/mypipe p
chmod 0666 /dev/mypipe
exit (Get back to TCL)
- Compile and catalog the FlashBASIC program, compile the C program and add it to the built in function list by:
addbi setusr1
cc -c setusr1.c
ar vru libgmu.a setusr1.o
To be able to use the customized monitor with a phantom process, it is necessary to shutdown and to move the customized monitor to the common directory /usr/bin while being in single user mode.
shutdown
make -f /usr/lib/pick/Makefile
init s (wait for the system to effectively go to single user)
mv /usr/bin/ap /usr/bin/ap.save
mv ./ap /usr/bin
init 2 (go back to multiuser)
Restart the D3 virtual machine.
- Create the phantom task server:
z pserver
- The server is now waiting for a signal to read the message. Normally, this would be done by another process (D3 or Unix), but, just to test this, the following shell script allows sending messages, small Unix files or terminate the server:
snd:

```
# Send a command to pserver through a pipe
PIPE=/dev/mypipe # the communication pipe
SERVERID=/tmp/pserverid # where the server puts its PID
COMMAND=$1
# Make sure the D3 server is active
if [ ! -r $SERVERID ]
then
echo D3 server is not active
exit 1
fi
# Get the server pid
PSERVERID=`cat $SERVERID`
# Send it a signal 0 to test its existence
kill -0 $PSERVERID
if [ $? != 0 ]
then
echo D3 server is not active. PID is not valid.
exit 1
fi
# Parse command
case $COMMAND in
```

```
"r"|"R")
# Receive a file : snd r unix.file
# D3.file item
echo "R\n$3\n$4\n~`cat $2~`" > $PIPE
;;
"m"|"M")
# Send a message: snd m <text>
shift
echo "M\n$*" > $PIPE
;;
"t"|"T")
# Terminate
echo "T\n" > $PIPE
;;
*)
echo "usage: snd r unix.file D3.file item"
echo " snd m message"
echo " snd t"
;;
esac
# Wake up the server by sending SIGUSR1
# -30 for AIX; -16 for SCO, DG
kill -30 $PSERVERID
- Test the server by the following commands (from Shell):
snd m Hi there
This command should send a D3 message to all D3 users.
snd r /etc/inittab dm,pointer-file, inittab
This command should send the Unix file /etc/inittab into the D3 item
dm,pointer-file, inittab .
snd t
This command should terminate the server process.
```

VI Quick Reference

lists the most commonly used vi commands.

Please note: vi is a case sensitive editor. The command "A" is different from the command "a".

Invoking vi:

vi filename

Insert Mode

In insert mode, the keyboard behaves like a typewriter. Keystrokes appear as screen text after typing any of the following commands.

i	Insert text at the cursor
A	Append text at the end of the line
a	Append text after the cursor
O	Open a new line of text above the cursor
o	Open a new line of text below the cursor
<ESC>	Exit insert mode & invoke command mode

Command Mode

In command mode, keystrokes perform functions such as moving the cursor, searching for patterns, or quitting from the document. All commands are referenced from the current cursor position.

Cursor Movement	
Arrow Keys	Move one space in any direction
G	Go to the last line in the file
nG	Go to line "n"
w	Move forward to the next word
b	Move backwards to the previous word
$	Move to the end of the line
0	Move to the beginning of the line
<ctrl>+d	Scroll down 1/2 screen
<ctrl>+u	Scroll up 1/2 screen
<ctrl>+b	Scroll up Full
<ctrl>+f	Scroll down Full

Searching	
/string	Search for a "string" (pattern) of characters
n	Search for the next occurrence of the "string"
:%s/str1/str2/g	Replace all occurrences of str1 with str2
Deleting Text	
x	Delete a single character
dw	Delete a word
dd	Delete an entire line
ndd	Delete an "n" number of lines
d$ or	
D	Delete from the cursor to the end of the line

Copying Text	
yy	Copy (yank) a line to the buffer
nyy	Copy (yank) an "n" number of lines to the buffer
P	Paste text from the buffer

Changing Text	
r	Mark a single character for replacement
cw	Mark a word for changing

cc	Mark a line for changing

Miscellaneous Commands

j	Join a line with the one below it
!cmd	Execute a Unix command
:r file	Read a file into vi
.	Repeat the last command
u	Undo the last command

Saving and Exiting

u	Write (save) the file
:q!	Quit the file without saving changes
ZZ	Write (save) the file and quit vi

.profile

is a Unix script executed when a Unix user logs in.

addbi (D3 Unix)

extends the functionalities of the D3 Monitor by making new C functions available to FlashBASIC.

"addbi" adds the user-defined built-in functions defined in the given "function" to the Monitor.

If a list of functions is omitted from the command line, then a list is presumed active. If no list is active, the User Generic Monitor library "libgmu.a" is simply rebuilt to match the list of built-in functions.

Up to 128 user-defined functions can be added to the Monitor.

Built-in functions are defined in the item "user.builtin" in the "dm,messages," file. "addbi" updates this item and creates, in the current directory, the C module "px_user.c" which is compiled and linked with the default Generic Monitor libraries to create a customized Monitor. Added functions can be called directly from BASIC.

User-defined built-in functions may be removed using the "rmbi" command.

Syntax

addbi {function {function} ...}

ap

starts the D3 virtual machine or a D3 user process.

-0

Starts the virtual machine. This process has the responsibility to initialize the virtual machine. All other processes will wait for the coldstart to complete its initialization before actually starting. Once the virtual machine is started, the line 0 can be disconnected, by typing 'exit' or

'disc', on any other line. Issuing 'ap -0' again will simply reconnect to the virtual machine if it is booted.

-a bootarg

Starts a virtual machine automatically. This process has the responsibility to initialize the virtual machine. All other processes will wait for the coldstart to complete its initialization before actually starting. This option is similar to the "0" option, with the difference that the system does not prompt for a boot option. Instead, it takes one-character commands from "bootarg" among the boot options: "x", "f" or "a". When encountering the "x" command, the system polls the keyboard for a period of "bootsleep" seconds (the default is 3 seconds) for a user intervention. If any key is pressed during this short period, the system defaults to a manual boot. "bootsleep" can be redefined in the D3 configuration file, by adding the statement 'bootsleep n' or by using the TCL command "config options". "bootarg" is a string of commands as if typed by the operator for a manual boot. When the command involves a tape, the tape is assumed to be ready. Therefore, the "c" for continue should not be included in the string.

- port.number

Starts a user process. Expressed in decimal from 1 to the maximum allowed number of users, this starts a D3 process on a given port when it is necessary to control the port on which process is running. If the port is not given the system allocates the first available port.

-q

Query. This command can be executed by any process to get information about the specified virtual machine. If the virtual machine is started, then the command returns exit code "0" to the shell, otherwise a value of "1" is returned. This allows testing the existence of the virtual machine from the shell.

-k

This command kills all processes attached to the specified virtual machine. First, a D3 logoff is attempted, followed by a terminate signal, which should send the process back to Unix. If the terminate signal has no effect, a kill is attempted which removes the process. This command should be used only in extreme situations. This is not a normal way to stop a virtual machine.

-n configfile

This specifies the name of the configuration file. If not given, the default file name is "pick0" in the current directory or alternately in the "/usr/lib/pick" directory.

-t tty

This option specifies which port is to become the terminal for the process. The device is assumed to be on the "/dev" special files directory. If not specified, the terminal is "stdout/stdin", unless it has been redirected, (the usual case on AIX systems). This option is normally intended to be used only in the "/etc/inittab" file. When this field is present, the system assumes the user process is started automatically, and, behaves a bit differently when starting: it waits for line 0 to start, if it is not started already.

-y sttyarg

This option allows changing the default port setting for the process. "sttyarg" is any "stty" argument. If more than one element is changed, they must be separated by spaces and the whole

argument enclosed between double quotes (see examples below). When the process terminates, the port characteristics are not reset to their original values.

-d dataarg

This option allows stacking data for the process once it is activated. "dataarg" is any string which contains displayable characters and commands prefixed by a back slash (\). Note that the string is subjected to the normal shell parsing, thus, back slashes must be "escaped", or the entire string must be enclosed in single quotes. "dataarg" can be contained in a Unix file by using the shell command substitution mechanism (e.g. using "back quoting" mechanism: -d ~` cat /usr/lib/pick/logon~`). All line feeds in the input string are converted to carriage returns.

The 'dataarg' commands are:

\r Insert a carriage return.

\f Turn echo off. The stacked data will not be displayed.

\n Turn echo on. The stacked data will be displayed.

\m Wait until the D3 virtual machine enters multiuser mode. The "maxusers" TCL command should be included in the "user-coldstart" macro after all system and application initialization are completed.

\\ Insert a back slash.

When activating a process for the first time, the system reads one character, then empties the type ahead buffer. Therefore, stacked data should always start by the sequence \r, followed by the real logon sequence.

-l

Retains login Unix user-id. This option logs the process as the same Unix user as the one used to log in to Unix. This option overrides the user definition contained in the configuration file. A D3 process started with this option does not have access to the D3 spooler, nor to the message facility (or at least very restricted). The user is in some way isolated from the other D3 users. This option should be used only for users who want their own Unix environment underneath the D3 process or to do system configuration which requires 'root' access.

-D

Enables Monitor Debugger. On entry, the process enters the Monitor Debugger. This option is ignored if the process started is a phantom or a printer. Type g<return> to actually start the process.

-s

Enables "silent" mode. If used along with the "-0" command, the D3 machine will boot, and return directly to Unix. If used on a normal line, output of logon and logoff messages and user macros is suppressed and any attempt to logoff will return directly to Unix. Because output is suppressed, the "-s" flag must generally be used along with the "-d" flag followed by a string containing the user name, user password, MD, and MD password when applicable so that the user is logged into D3 automatically. The "-s" flag is used by the "tcl" Unix shell script.

-i nice

Set the relative priority of the D3 process, compared to other processes, D3 or not, running on the system. Legal values of 'nice' are -20 to +19. -20 gives the highest priority, +19 the lowest.

-W

Wait for the device specified by the '-t' option to be created. This option instructs the D3 monitor that the device does not exist yet, thus avoiding the 'no such file' error. This is used in configurations where the '/dev/' entry is created dynamically by a Unix daemon (for example on HP-UX using a DTC). The process polls the specified device and waits until it is created as a pipe or block or character device. This option is ignored if the '-t' option is not specified as well.

-printer

This option specifies that the port is to be used as a printer. It suppresses the messages 'Connected to virtual machine',and 'Disconnected from virtual machine', but does not suppress the normal D3 processes messages, like the D3 logon message.

-pprinter

Parallel Printer. This option specifies that the port is to be used as a printer on a parallel device. This option must be used on Unix implementations where the device is a write-only device, such as in the case of AIX. If the parallel printer is a Read/Write device, this option is equivalent to "-printer".

-spooler

-scheduler

-phantom

Any of these three options specifies that the port is to be used as a phantom, or as a process not attached to a physical port.

-u /ttelnet.port,s

Starts D3 telnet on telnet port telnet.port, where s is the server host name. This option starts the D3 telnet server on the telnet port number specified by telnet.port and waits for connection from a client. The client makes the connection by using telnet on the server host and the same telnet.port.

Syntax

ap {{-[0|a bootarg|port.number|f|q|k] }{-n configfile} {-t tty} {-y sttyarg} {-d dataarg} {-l} {-i nice} {-[printer|pprinter]}} {-D} {-s} {-W} {-u /ttelnet.port,s}

Example

```
ap -0
Starts the virtual machine 'pick0' (default name).

ap -n mymachine -0
Starts the virtual machine 'mymachine'.

ap
Starts a User process on the virtual machine 'pick0' (default name ' '), on
the first available port. (default PIB ' ').

ap -5
Starts a User process on the virtual machine 'pick0' (default name '-'), on
port 5.

pick -n mymachine -7

Starts a User process on the virtual machine 'mymachine', on port 7.

ap -q
```

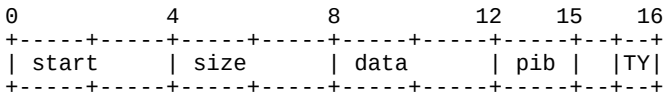
```
Displays information about the virtual machine 'pick0' (default name).

ap -0 -t tty2
Starts the virtual machine 0 on /dev/tty2. This statement is normally included
in the /etc/inittab file.

ap -3 -t tty6 -y "9600 parenb -parodd" -printer

Starts a user process on /dev/tty6 as a printer. It changes the baud rate to
9600 baud and sets the parity to odd. This statement is normally included in
the /etc/inittab file.

ap -a a3x
Automatically boots the virtual machine, then does an ABS restore from device
3, then issues an "x" option.

ap -d '\r\mdm\racct\rterm ibm3151\rmenu\r'
Starts a user process on the first available port, stacking commands to wait
until multiuser mode is entered, and then log it as "dm" on the account acct,
executing the "term ibm3151" and "menu" commands.
Note the first '\r' to make sure the process will be logged on properly the
very first time after a boot.

Shell script 'boot.ap':
001 # Test if VM is active. else boot it
002 ap -q > /dev/null
003 if [ $? ! -eq 0 ]
004 then
005    -a x
006 fi

This shell script uses the -q option to test whether the virtual machine is
booted or not. If the 'ap -q' command returns a null exit code (ok), then the
virtual machine is already booted and nothing is done. Else the virtual
machine is booted automatically.

$su
password:(enter 'root' password)
$ap -l

Enters the D3 virtual machine, retaining the current Unix user id
(root, because of 'su').

$ap -7 -u /t2007,serverhost &

Starts a background D3 telnet server process which connects to the default D3
virtual machine "pick0" on PIB 7 and waits for connection from client.  If the
server host name is serverhost, a Unix client makes the connection by using
the "telnet serverhost 2007" command in Unix shell.
```

ap.core

is created automatically on the directory "/usr/tmp" when a machine abort occurs (bus error, segmentation violation, ...), or upon request with the Monitor Debugger 'd' command.

This file is a regular Unix file which can be written on tape, using Unix utilities like 'tar' or 'cpio' to be sent to Technical Support for investigating a problem. The utility 'apcrash' can be used to examine the core file with commands similar to the D3 Virtual Debugger.

The core file is made of 'segments', each representing part of the memory at the time the dump occurred. The segments have a 16 byte header as follows:



start Start address.

size Size in bytes.

data Optional data filed.

pib PIB number of the process.

TY Segment type.

The actual size of the segment is rounded up to the nearest 4 byte boundary.

The following lists the segment types and the meaning of the other fields, where applicable:

"0" PCB.

"B" Buffer table.

"F" Virtual Frame. 'data' is the FID.

"G" Global segment. 'start' is the value of the GM register ra.

"L" Local segment. 'start' is the value of the GM register ral.

"M" Monitor Version. 'data' is the frame size, in bytes.

"P" PIBs table. 'data' is the pib size, in bytes.

"R" Hardware registers.

"S" Main shared memory. 'data' is the starting offset of the dump.

"V" Virtual Buffers.

ap.log

logs all errors and major events occurring on a D3 virtual machine.

This file is created automatically at install time in the directory /usr/tmp. It is updated by any D3 virtual machine.

It is a binary file which grows indefinitely, until cleared by the 'perrpt' Unix command.

The format of each entry is as follows:

struct {

int log_time; /* Unix time */

int log_key; /* Virtual machine key */

int log_res0; /* reserved */

short log_code; /* Error code */

short log_pib; /* D3 line number */

short log_res1; /* reserved */

short log_len; /* Optional data length */

char log_mon[16]; /* Monitor version */

char log_text[]; /* Optional data */

}

Each entry is aligned to an (int) boundary.

The last (int) of an entry contains the length of the entry, to be able to scan the file backward.

It can be disabled by issuing 'perrpt -s'.

apcrash

a utility running under Unix to examine D3 core dumps.

A D3 core dump is created either automatically when a machine abort occurs (bus error, segmentation violation, ..), or by requesting it with the Monitor Debugger 'd' command.

The core file contains a series of segments, each representing part of the D3 memory at the time the dump was done. Segments can contain both Monitor and Virtual data. The data in the core file cannot be changed.

"core.file" Name of the core file to examine. If not specified, the default '/usr/tmp/ap.core' is used. Note this file can be very large.

Display formats:

Data is displayed in a way similar to the Virtual Debugger. "apcrash" has a notion of a 'window' (number of bytes displayed). After each window is displayed, an equal (=) sign is displayed and the user is prompted for an action. A one character code, followed by <return> is expected:

n Next window.

p Previous window.

x Redisplay data in hexadecimal.

c Redisplay data in characters.

i Redisplay data in decimal.

"apcrash" displays a prompt '(crash)' and waits for commands on its standard input:

? Display Help.

!unix.command Execute the Unix shell command 'unix.command'.

{.}fid[,|.]dsp{;win} Display the specified Virtual Address. The core file must contain either a segment for the appropriate 'fid', or both the Buffer Table segment AND a the Virtual Buffer segment which contains the specified frame.

b{.}fid Display the buffer table entry associated to the specified FID. The Buffer Table segment must be present.

l{.}fid Display the Link fields of the specified frame. The core file must contain either a segment for the appropriate 'fid', or both the Buffer Table segment AND a the Virtual Buffer segment which contains the specified frame.

m{l|g}{.}addr{;win} Display 'real' memory. The prefix 'l' specifies a local address (mpcb); 'g' specifies a global address (mcb). Without prefix, 'addr' is an 'absolute' address.

q or <ctrl>+D Quit.

S{type} Display the core segments in the current core file. If 'type' is specified, only the segments of this type are specified. See the Pick Systems Reference Manual entry for 'ap.core'.

v Display the Monitor version.

x Display the hardware registers, PCB fid, ra and ral values and the PIB of the process which made the dump. The Register, PCB, es Global and Local segments must be in the core file.

Syntax

apcrash {core.file}

d3

starts the D3 virtual machine or a D3 user process. The D3 installation procedure creates an alias named ‘ap’ to the ‘d3’ command. Either command can be used. The ‘ap’ form will be obsoleted on a future D3 release.

-0	Starts the virtual machine. This process has the responsibility to initialize the virtual machine. All other processes will wait for the coldstart to complete its initialization before actually starting. Once the virtual machine is started, the line 0 can be disconnected, by typing 'exit' or 'disc', on any other line. Issuing 'd3 -0' again will simply reconnect to the virtual machine if is booted.
-a bootarg	Starts a virtual machine automatically. This process has the responsibility to initialize the virtual machine. All other processes will wait for the coldstart to complete its initialization before actually starting. This option is similar to the "0" option, with the difference that the system does not prompt for a boot option. Instead, it takes one-character commands from "bootarg" among the boot options: "x", "f" or "a". When encountering the "x" command, the system polls the keyboard for a period of "bootsleep" seconds (the default is 3 seconds) for a user intervention. If any key is pressed during this short period, the system defaults to a manual boot. "bootsleep" can be redefined in the D3 configuration file, by adding the statement 'bootsleep n' or by using the TCL command "config options". "bootarg" is a string of commands as if typed by the operator for a manual boot. When the command involves a tape, the tape is assumed to be ready. Therefore, the "c" for continue should not be included in the string.
- port.number	Starts a user process. Expressed in decimal from 1 to the maximum allowed number of users, this starts a D3 process on a given port when it is necessary to control the port on which process is running. If the port is not given the system allocates the first available port.
-q	Query. This command can be executed by any process to get information about the specified virtual machine. If the virtual machine is started, then the command returns exit code "0" to the shell, otherwise a value of "1" is returned. This allows testing the existence of the virtual machine from the shell.

-k	This command kills all processes attached to the specified virtual machine. First, a D3 logoff is attempted, followed by a terminate signal, which should send the process back to Unix. If the terminate signal has no effect, a kill is attempted which removes the process. This command should be used only in extreme situations. This is not a normal way to stop a virtual machine.
-n configfile	This specifies the name of the configuration file. If not given, the default file name is "pick0" in the current directory or alternately in the "/usr/lib/pick" directory.
-t tty	This option specifies which port is to become the terminal for the process. The device is assumed to be on the "/dev" special files directory. If not specified, the terminal is "stdout/stdin", unless it has been redirected, (the usual case on AIX systems). This option is normally intended to be used only in the "/etc/inittab" file. When this field is present, the system assumes the user process is started automatically, and, behaves a bit differently when starting: it waits for line 0 to start, if it is not started already.
-y sttyarg	This option allows changing the default port setting for the process. "sttyarg" is any "stty" argument. If more than one element is changed, they must be separated by spaces and the whole argument enclosed between double quotes (see examples below). When the process terminates, the port characteristics are not reset to their original values.
-d dataarg	This option allows stacking data for the process once it is activated. "dataarg" is any string which contains displayable characters and commands prefixed by a back slash (\). Note that the string is subjected to the normal shell parsing, thus, back slashes must be "escaped", or the entire string must be enclosed in single quotes. "dataarg" can be contained in a Unix file by using the shell command substitution mechanism (e.g. using "back quoting" mechanism: -d ~` cat /usr/lib/pick/logon~`). All line feeds in the input string are converted to carriage returns. The 'dataarg' commands are: \r Insert a carriage return. \f Turn echo off. The stacked data will not be displayed. \n Turn echo on. The stacked data will be displayed. \m Wait until the D3 virtual machine enters multiuser mode. The "maxusers" TCL command should be included in the "user-coldstart" macro after all system and application initialization are completed.

	\\ Insert a backslash. When activating a process for the first time, the system reads one character, then empties the type ahead buffer. Therefore, stacked data should always start by the sequence \r, followed by the real logon sequence.
-dcdon	Enables DCD protocol handling. This is similar to "dcd on" at TCL, with the exception that this option does not change any terminal characteristics. This option, along with the 'y' option will allow lines sitting at the login prompts to log off and/or restart after an imterruption such as a power failure.
-b	Disables the Abs protection mechanism for a particular process. This option is required when loading 3rd party applications into boot abs, or loading patches.
-f	Restarts the flusher process. If the flusher process is incorrectly killed, this option can be used to restart the flusher. Note that this option should never be used if the flusher is already running.
-m	Restarts the signal handler process. If the signal handler process is incorrectly killed, this option can be used to restart the signal handler process. Note that this option should never be used if the signal handler is already running.
-l	Retains login Unix user-id. This option logs the process as the same Unix user as the one used to log in to Unix. This option overrides the user definition contained in the configuration file. A D3 process started with this option does not have access to the D3 spooler, nor to the message facility (or at least very restricted). The user is in some way isolated from the other D3 users. This option should be used only for users who want their own Unix environment underneath the D3 process or to do system configuration which requires 'root' access.
-D	Enables Monitor Debugger. On entry, the process enters the Monitor Debugger. This option is ignored if the process started is a phantom or a printer. Type g<return> to actually start the process.
-s	Enables "silent" mode. If used along with the "-0" command, the D3 machine will boot, and return directly to Unix. If used on a normal line, output of logon and logoff messages and user macros is suppressed and any attempt to logoff will return directly to Unix. Because output is suppressed, the "-s" flag must generally be used along with the "-d" flag followed by a string containing the user name, user password, MD, and MD password when applicable so that the user is logged into D3 automatically. The "-s" flag is used by the "tcl" Unix shell script.

-i nice	Set the relative priority of the D3 process, compared to other processes, D3 or not, running on the system. Legal values of 'nice' are -20 to +19. -20 gives the highest priority, +19 the lowest.
-W	Wait for the device specified by the '-t' option to be created. This option instructs the D3 monitor that the device does not exist yet, thus avoiding the 'no such file' error. This is used in configurations where the '/dev/' entry is created dynamically by a Unix daemon (for example on HP-UX using a DTC). The process polls the specified device and waits until it is created as a pipe or block or character device. This option is ignored if the '-t' option is not specified as well.
-printer	This option specifies that the port is to be used as a printer. It suppresses the messages 'Connected to virtual machine', and 'Disconnected from virtual machine', but does not suppress the normal D3 processes messages, like the D3 logon message.
-pprinter	Parallel Printer. This option specifies that the port is to be used as a printer on a parallel device. This option must be used on Unix implementations where the device is a write-only device, such as in the case of AIX. If the parallel printer is a Read/Write device, this option is equivalent to "-printer".
-spooler -scheduler -phantom	Any of these three options specifies that the port is to be used as a phantom or as a process not attached to a physical port. Note that these are reserved for use by D3 utilities.
-u /ttelnet.port,server.hostname	Starts D3 telnet on telnet port telnet.port, where server.hostname is the server host name. This option starts the D3 telnet server on the telnet port number specified by telnet.port and waits for connection from a client. The client makes the connection by using telnet on the server host and the same telnet.port.

Syntax

d3 {{-[0|a bootarg|port.number|f|q|k] }{-n configfile} {-t tty} {-y sttyarg} {-d dataarg} {-l} {-i nice} {-[printer|pprinter]}} {-D} {-s} {-W} {-u /ttelnet.port,s}

Example

```
d3 -0
Starts the virtual machine 'pick0' (default name).

d3 -n mymachine -0
Starts the virtual machine 'mymachine'.

d3
Starts a User process on the virtual machine 'pick0' (default name ' '), on
the first available port. (default PIB ' ').

d3 -5
Starts a User process on the virtual machine 'pick0' (default name '-'), on
port 5.
```

```
d3 -n mymachine -7
Starts a User process on the virtual machine 'mymachine', on port 7.

d3 -q
Displays information about the virtual machine 'pick0' (default name).

d3 -0 -t tty2
Starts the virtual machine 0 on /dev/tty2. This statement is normally included
in the /etc/inittab file.

d3 -3 -t tty6 -y "9600 parenb -parodd" -printer
Starts a user process on /dev/tty6 as a printer. It changes the baud rate to
9600 baud and sets the parity to odd. This statement is normally included in
the /etc/inittab file.

d3 -a a3x
Automatically boots the virtual machine, then does an ABS restore from device
3, then issues an "x" option.

d3 -d '\r\mdm\racct\rterm ibm3151\rmenu\r'
Starts a user process on the first available port, stacking commands to wait
until multiuser mode is entered, and then log it as "dm" on the account acct,
executing the "term ibm3151" and "menu" commands.
Note the first '\r' to make sure the process will be logged on properly the
very first time after a boot.

Shell script 'boot.d3':
001 # Test if VM is active. else boot it
002 d3 -q > /dev/null
003 if [ $? ! -eq 0 ]
004 then
005 -a x
006 fi
This shell script uses the -q option to test whether the virtual machine is
booted or not. If the 'd3 -q' command returns a null exit code (ok), then the
virtual machine is already booted and nothing is done. Else the virtual
machine is booted automatically.

$su
password:(enter 'root' password)
$d3 -l
Enters the D3 virtual machine, retaining the current Unix user id (root,
because of 'su').

$d3 -7 -u /t2007,serverhost &
Starts a background D3 telnet server process which connects to the default D3
virtual machine "pick0" on PIB 7 and waits for connection from client.  If the
server host name is serverhost, a Unix client makes the connection by using
the "telnet serverhost 2007" command in Unix shell.
```

env

displays the Unix environment variables.

environ

manipulates Unix shell environment variables from TCL.

Commands may be passed from the TCL command line or from an item.

It is possible to set and test shell variables, as well as execute TCL commands (including Unix commands prefixed by "!") under control of the shell variables.

If no arguments are provided, the commands are read from the item ".profile" in the md of the current account.

A "command" is a sequence of non-blank words separated by spaces. If the command starts with an "*" (asterisk), it is treated as a comment and ignored. If the first word is not one of the key words recognized by "environ", the command is treated as a TCL command, which is then executed. Commands are read from either the TCL sentence or from an item. Multiple commands may be issued on the same line by separating each command with a ";" (semi-colon).

Command Substitution:

Text enclosed in "back quotes" (`text`) is considered a TCL command, which is executed, its output captured and the result inserted in the command in place of the "back quoted" command. Attribute marks in the output are converted to a space. Command substitution occurs before the command is parsed.

If the command is a Unix command, only "stdout" is captured. Use the Unix redirection to send "stderr" to "stdout" if necessary.

Parameter Substitution:

The "$" character is used to introduce parameters for substitution. If the parameter is a one-digit number, it is a positional parameter in the range 0 to 9.

"$0" is the concatenation of all positional parameters from "$1" through "$9" with one space inserted between each value.

Positional parameters are set by the "set" environ statement.

The parameter name can be one reserved keyword interpreted locally by "environ" or a Shell parameter.

Parameter substitution is performed before the command is parsed and before command substitution.

Shell parameter (or environment variables) may be assigned values by the command:

name=value

Variables set in this manner are automatically marked for export to the environment of any subsequently executed Unix commands.

The following Shell variables are available:

$? This contains the value of the return code of the last executed command. If the last command was a Unix command, this value is the exit code. If it was a TCL command, this value is the error message number returned.

$$ This returns the Unix ID (PID) of the current D3 process.

$! This returns the current D3 port.number.

$n This indicates the positional parameter "n" to substitute. Must be in the range 0 through 9

$USER This returns the current D3 user-id.

$ACCT This returns the current D3 account name.

$TIME This returns the current D3 internal time.

$DATE This returns the current D3 internal date.

${parameter}

The value of the Shell environment variable "parameter" is substituted. The "braces" are required only if the "parameter" is not followed by a space.

Control Structures:

if condition

then

{elif condition}

{then}

else

end

Where "condition" is evaluated and the commands are executed according to the result. Conditions may be expressed in either of two ways:

argument1 {!}operator argument2

Where "argument" is either a string, a number, or a parameter, and "operator" is either "=", "<", ">", "<=" or ">=".

{!}op argument1 {argument2{ ...}}

Where "argument" is either a string, a number, or a parameter and "op" is either:

-f file

The condition is true if the specified file exists and can be opened.

-r file item-id

The condition is true if the specified item-id exists and can be read.

-v file item-id value

The condition is true if the specified value exists and is not empty.

-n string

The condition is true if "string" has a non-zero length.

The "!" before the operator negates the condition.

Environ Commands:

chain {file.reference} item-id

This causes "environ" to begin executing the commands in the given item-id. If "file.reference" is omitted, md is assumed. Control does not return to the calling "environ" program.

exit

This causes "environ" to terminate.

read

This reads one line from the terminal and assigns each word to the positional parameters "$1", "$2", etc. Up to 32 words are moved into the positional parameters.

set {[+|-] flags} {arguments...}

This sets (-) or resets (+) the given "environ" flags. This setting overrides the flags set on the command line.

shift {n}

The positional parameters "$n+1" "$n+2" etc., are renamed "$1" "$2" etc. If "n" is omitted, "1" is assumed.

Invocation flags:

-c string

Reads commands from "string". If "string" contains more than one word, it must be enclosed in single quotes.

-f {file.reference} item-id

Reads commands from "item-id" in "file.reference". If "file.reference" is omitted, md is assumed.

-n Reads commands, but does not execute them. This should be used with the "-x" flag to investigate the effect of an environ program, for debugging purposes.

-v Prints commands and their arguments as they are read, whether they are executed or not.

-x Prints commands and their arguments as they are executed. Display is prefixed by a "+".

Syntax

environ {-x} {-n} {-c string|-f {file.reference} item-id}

Example

```
1) :environ -c 'term $TERM'

This sets the D3 terminal type definition to the same value as defined in
Unix.

2) :environ -f ${USER}.env

Evaluates the value of the variable $USER and executes the specified environ
script in the current dictionary. Note the usage of the braces around the
variable name, since it is not followed by a space.


3) :environ -c 'TERM=wyse50'

This sets the Unix terminal driver to a Wyse 50. The verb "env" verifies the
new environmantal parameter.

4) :environ -f myprofile

Executes the commands contained in the (md) item "myprofile", as defined
below, which tests for the existence of the Unix "TERM" variable, uses it as
an argument to the TCL "term" command if the corresponding item exists in the
"dm,devices," file. It also sets a Shell variable PICKLOGON" equal to the D3
date and time, and a Shell variable "PICKWHO" equal to the usual TCL "who"
command.
Finally, it tests for the existence of an item profile.username" and executes
it if it exists.

md myprofile
001 *
002 * Set D3 Term if the shell TERM variable is defined.
003 if $TERM != ""
004 then
005 if -r dm,devices, $TERM; then term $TERM; end
006 else
007   display TERM is not defined.
```

```
008 end
009 *
010 * Set a Shell variable equal to D3 logon time, using
011 * command substitution (back quotes around time).
012 PICKLOGON=`time`
013 *
014 * Set a shell variable equal to port number,user id and MD
015 PICKWHO=$! $USER $ACCT
016 *
017 * And go execute any private environment, if item exists
018 if -r md .profile.$USER; then chain .profile.$USER; end
```

export (Unix)

exports D3 items to Unix files.

Attributes marks are converted to new line (x'0a') unless the "c" option is used. Other D3 delimiters (value marks, etc.) are not translated.

"unix.file" is created with Read Owner and Write Owner rights only (mode 0600), unless the "g" and/or "w" options are used.

"file.reference" is the source D3 file name.

"item.list" is the list of the source items. If not specified, an active list must be present. (see "select", "sselect", "get-list").

"dir" is the directory path where the destination files are to be copied. The filenames are the same as the item-id(s) from the source list. The "l" option is used to convert Unix filenames to lower case.

"Unix.file" is the list of the destination Unix files. Complete pathnames must be used if the files are not to be exported to the current directory. The current directory may be changed with the "cd" command. If no destination list is provided, the source list is used as the destination list.

Syntax

export file.reference {item.list} {(options}

to :[dir | {Unix.file {unix.file} ... }]

Options

c Prevents attribute marks to new lines conversion.

g Gives the unix.file read/write permissions to the group. This flag may be modified by the value of umask(1) of the current process.

i Suppresses display of item-ids.

l Converts unix.file name to lower case before exporting.

w Gives the unix.file read/write permissions to the "rest of the world". This flag may be modified by the value of umask(1) of the current process.

listbi

lists the built-in C functions in the item "unix.builtin" (D3 Unix) and "user.builtin" located in the "dm,messages," file. These include any built-ins that have been added using the "addbi" verb.

Syntax

listbi {(option}

Options

p Directs output to system printer, via the Spooler.

s Lists system (predefined) functions only.

u Lists user-defined functions only.

Example

```
listbi
System Functions

Num Name          Num Name          Num Name

62 ALARM         78 CALLOC          0 CHDIR
 1 CHMOD          2 CHOWN           3 CLOSE
72 CPSTARTUP      4 CREAT           5 DUP
 7 FCLOSE         8 FDOPEN         11 FGETC
12 FGETS         14 FOPEN          15 FPRINTF
16 FPUTC         17 FPUTS           6 FREE
18 FREOPEN       51 FSIZE          67 GETENV
20 GETPGRP       19 GETPID         21 GETPPID
22 IOCTL         23 KILL           24 LSEEK
25 MALLOC        81 MEMCCPY        82 MEMCHR
80 MEMCPY        83 MEMXCPY        26 OPEN
27 PAUSE         65 PCLOSE         56 PGETPID
28 PIPE          76 PLDMEMALLOC    79 PLDMEMFREE
64 POPEN          9 PRINTF         73 PSTRALLOC
68 PUTENV        13 RDHEX          29 READ
30 SEMCTL        31 SEMGET         32 SEMOP
33 SHMCTL        58 SHMDT          34 SHMGET
10 SPRINTF       70 STRCAT         36 STRCPY
69 STRLEN        63 STTY           75 UNLOAD
49 USER0         50 USER1          37 WAIT
59 WHEX          38 WRITE

No user function currently defined
```

lppick

acts as a filter between the output of a D3 printer and the Unix spooler to implement shared printers.

The continuous data stream coming out of the D3 printer process is cut into separate Unix jobs. To do so, the data stream is scanned to find an 'End of job' string. Once a 'job' has been identified, a Unix command (typically 'lp') is invoked to process the data.

This command is normally called automatically by the TCL 'startshp' command.

"-d"

Debug. Some information is displayed while data is processed. This option should be used only to diagnose problems. The (V) option of 'startshp' sets this flag.

"-s string"

Specify the end of job sequence. The default is a one character string x'04'. It is strongly advised to specify an alternate end of job sequence. Non printable characters can be specified in octal, prefixed by a back slash '\'.

"-c command"

Specify the Unix command to use to spool data. This command must be able to accept data on its standard input. The default is "lp".

"-l file.name"

Create a Unix file name when the filter is activated. This file contains the PID of the filter, the strings "command" and "string", separated by attribute marks. The 'startshp' command uses this option so that the TCL command 'shp-status' can keep track of the shared printers.

"-t level"

Activate tracing on the printer activity. 'level' is an integer from 1 to 3 specifying the level of tracing. See the 'startshp' documentation to see the effect of this tracing. The -'t' option requires the '-l' option.

"-v"

Display the version of the 'lppick' utility, as follows:

lppick: Version X.Y

"-V"

Display the version of the 'lppick' utility in a short form, for use in shells or FlashBASIC utilities, as follows:

X.Y

Syntax

lppick {-d} {-t level} {-s string} {-c command} {-l file.name}

lppick -v

lppick -V

Example

```
lppick -s "End of Job\377" -c "lp -s -onobanner"
Start the filter, with an end of job sequence equal to 'End of Job_', where
'_' denotes a segment mark. The data will be passed to the Unix spooler
suppressing all messages (-s) and the banner (-onobanner).
```

perrpt

displays, clears, restarts or suppresses the D3 error log maintained on Unix. The system automatically logs every boot, shutdown and incidents or abnormal procedures related to a D3 virtual machine.

Options are specified by the following flags:

-e {code}

Displays errors only. Boots and shutdowns are not displayed. If 'code' is specified, only the log entries with this error code are displayed.

-h

Displays only the header of the log entries (short form). The following information is displayed:

Unix date and time

Virtual machine key

Monitor version

Error code,

Error description,

Port number (PIB).

-f file.name

Uses 'file.name' as the log file, instead of the default.

-p pib

Displays all entries logged by the D3 port 'pib', expressed in decimal.

-a

Displays logs for all virtual machines.

-k key

Displays error log for the virtual machine specified by 'key'. The virtual machine key is specified using standard C syntax: 0xNNNN for a hexadecimal number, NNNN for a decimal number and 0NNNN for an octal number.

-x key

Displays error log for all virtual machines except the one specified by 'key'. The virtual machine key is specified using standard C syntax: 0xNNNN for a hexadecimal number, NNNN for a decimal number and 0NNNN for an octal number.

-c

Clears the error log and restarts error logging. Requires 'root' privilege.

-s

Clears error log and stops the logging process. Requires 'root' privilege. After this command is issued, no error logging will take place. Error logging is restarted by using the '-c' flag. The trace is re-activated automatically every time D3 is (re)installed on the system.

The log is kept is the file '/usr/tmp/ap.log'. Each log entry has the Unix time and date of the incident, the key in hexadecimal of the D3 virtual machine, an error code, the process port (PIB) number of the process which LOGGED the error (not necessarily the process which had a problem) and optional information in form of a hexadecimal dump. The following explains briefly each error and the format of the additional data:

1 Boot

2 Shutdown

3 Line 0 or the flusher received a SIGTERM. When the flusher receives this signal, it shuts the virtual machine down. There should be two shut entries when the system is shut down properly (i.e., there is log entry for a shutdown). If not, the system was shut down abnormally ('ap -k' for example).

4 Process killed. When a D3 process does not respond, 'logoff' will escalate and send a signal SIGKILL to the D3 process. The additional information is (in hexadecimal):

<pib+1 of killed process> <pib+1 of killer>

5 Process abort. When a process receives a serious abort signal (SIGSEGV, SIGBUS, SIGTRAP, etc...) the machine registers are dumped in the additional data (in a format implementation dependent).

6 Security failure. Wrong activation code.

7 <reserved>

8 Tape error. The additional information contains:

<cmd> <devnum> <errno>

<cmd> 0 Read

1 Write

<devnum> Tape device number, in hex.

<errno> Unix error code, in hex.

9 Tape buffer allocation failed. The internal tape buffer is allocated dynamically. If this fails because Unix is temporary out of memory, the 'set-device' command may appear to hang.

10 Keyboard read error. A process attempted to read from 'stdin' (normally a keyboard), and got an error. The process will be disconnected. The additional information is (in hexadecimal):

<pib> <errno>

11 Terminal write error. A process attempted to write to 'stdout' (normally a terminal) and got an error. The additional information is (in hexadecimal):

<pib> <errno>

Syntax

perrpt {-e {code}} {-h} {-f file.name} {-p pib} [-a|-k key|-x key|-c|-s]

Example

```
perrpt -a | pg
Shows all errors log, stopping at each page.

perrpt -e -k 0x29733 | lp
Shows only the errors relative to the virtual machine with the key 0x29733
(hexadecimal), and pipes the result to the Unix spooler.

perrpt -c
Clears error log, but leaves it active, or restarts it if it was stopped.

perrpt -s
Stops error logging.

perrpt -e 4 -h
Shows all entries with an error code=4, in a short form.

mv  /usr/tmp/ap.log  /usr/tmp/ap.log.monday
perrpt -a -f /usr/tmp/ap.log.monday

Save the current error log in the file usr/tmp/ap.log.monday' and list it.
```

pick

provides a front-end to the Unix 'ap' command.

The 'pick' TCL command is a front end to the Unix command 'ap'. Some care must be taken, however, in its usage since it can create a new D^3 user process on top of the existing D^3 process.

The TCL command 'ap' is a synonym to the TCL command 'pick'.

The '-q' form gets status information on the current virtual machine. This is the only legitimate and innocuous usage of this command from TCL.

If "machine" (typically specified as "pick0") is not specified, the current virtual machine is assumed. The "-n" flag may be omitted if there is no ambiguity.

If it is really necessary to 'stack' several D3 processes on the same terminal, use "disc (f)" (or the "exit" macro) and NOT the form "disc (n)" to unstack D3 processes. Otherwise, processes will be left 'hanging' and will create confusion.

Syntax

pick { {-q } {{-n} machine} } {(options}

Options

l Generates the string necessary to automatically log on to the specified virtual machine, as the same user, on the same md and to set the same TERM. This option cannot be used if there is a password, or if the user macro or logon PROC contains any prompt.

q Quiet. Suppresses all messages.

u Generates the necessary string to log on to the specified virtual machine as the same user.

Example

```
ap (l
Connects to the same virtual machine as the current one, logging automatically
to the same account, as the same user.

ap dev2 (u
Connects to the virtual machine 'dev2', as the same user. The md is requested,
as usual.
```

pick0

defines the parameters required for each D3 virtual machine on the system.

This file, located on the Unix directory "/usr/lib/pick", contains implementation-specific data, which can be found in the installation guide provided with your system. It should be edited only using the TCL utility "config" or the D3 installation procedure.

Example

```
#
# Copyright (c) Pick Systems 1992, 1993, 1994.  All rights reserved.
# Fri Jul 29 19:19:19 1994
#
name DGUX pick0
nice 0
user pick
core 81920 10
npibs 200
npht s 32
basic 2048
brkchr 1d
escchr 1b
absbase 18
abssize 708
abslock on
blkfid 2
flush 10
dwqnum 512
disk /dev/rdsk/prod 0 2000000        # disk 0
disk /dev/rdsk/prod1 0 450000        # disk 1
tape /dev/rpdsk/2 500 f lq           # tape 0
tape /dev/rpdsk/3 500 f lh           # tape 1
tape /dev/rmt/0n 16384 q lh          # tape 2
tape /dev/rmt/1n 16384 v l           # tape 3
tape /dev/rmt/2mn 16384 h ls         # tape 4
tape /dev/rmt/3n 16384 d l           # tape 5
tape /usr/opt/pick/bin/abs 500 f lx  # tape 6
```

```
tape /u2/tmp/floppy 500 f lx        # tape 7
tape /tmp/pseudo 500 p lx
```

Pick0 File Statement explanation

```
#
# Copyright (c) PICK Systems 1992. All rights reserved
# Mon Aug 14 16:31:56 1995
#
  The heading (in comments). It shows the Pick Systems copyright and the last
date and time of creation/modification of the configuration file.

name SCO pick0
  The D3 virtual machine platform and name.  The platform specifies the Unix
host, i.e., AIX, DGUX, SCO, etc. The virtual machine name is the name you call
the D3 System on Unix.  The name may be up to 64 characters, or to the end of
the line.  The default name is "pick0".

user pick
  The Unix user who "owns" the virtual machine. If this field is empty, the
virtual machine will be left owned by "root". The Unix user account (user-id)
must be created before the Machine is booted. The default user name is "pick".

groups  group0  { group1 ... }
  List of Unix groups that are granted access to the virtual machine. If the
groups statement is not provided, all users are allowed access to the virtual
machine. Up to 8 groups, separated by spaces, can be listed. Note that the
group to which the user defined in the user statement belongs is not
automatically included. The default is no groups entry; all users have access.

bootmode  s
  Boot mode. Specify whether the boot is done in single-user ("s") or multi-
user ("m") mode. When booting in single-user, only line 0 will have access to
the virtual machine until the TCL command maxusers is used to go into multi-
user mode. The default is "s" (single-user).

bootsleep 3
  Delay in second(s) during which the user has the possibility to stop an
automatic boot, when the virtual machine is booted with the '-a x' option. The
minimum value is 0. The maximum value is 2,147,483,648 seconds (or 68 years).
The default is 3 seconds.

nice 0
  This (signed) value adjusts the relative scheduling priority of the D3
process compared with other regular Unix processes. A negative value gives a
higher priority. A positive value gives a lower priority. Legal values are -19
(highest priority) to +20 (lowest priority). The default value is "0".

niceincr 10
  This value is added to the "nice" value of the flusher, and the result is
used to set all D3 processes. The default value is "10".

core 114688 10
  Virtual machine memory usage and the virtual machine Key. The first value is
the amount of shared memory (in Kb) the D3 process will allocate for itself
while the virtual machine is running. Low values may result in poor
performance of the D3 process due to swapping. High values may result in poor
performance of other Unix processes due to inadequate remaining memory. The
'core' parameter should be reduced when running primarily FlashBASIC
applications on a system. To determine an appropriate size, boot the system,
bring up all users in FlashBASIC, and run the buffers command with the (sl)
options. Note the "referenced" number (not the percentage, but the larger
number before it), multiply it by the frame size, and then divide by 1024 to
```

get a good estimate of the core size required for reasonable performance. Then shutdown the D3 virtual machine, reset the parameter, and reboot. The default values are:
On AIX(with FlashBASIC): 37.5% of the total memory
(without FlashBASIC): 50% of the total memory
On DG/UX (with FlashBASIC): 37.5% of the total memory
(without FlashBASIC): 50% of the total memory
On SCO(with FlashBASIC): (total memory - kernel size) * 90%
(without FlashBASIC): (total memory - kernel size) * 10%
The second value is the virtual machine Key. This number, in hexadecimal, is a unique identifier for this particular D3 virtual machine. Each virtual machine must have its own unique identifier. Therefore, if several D3 virtual machines are running on the system, this identifier must be changed for each one. The default value is "10".

npibs 512
The number of pibs, or ports, or users. This number should be at least equal to the number of users the system is licensed for, plus the expected number of printers (parallel or serial). It is a safe practice to allocate more ports than necessary because it will simplify upgrade procedures when the number of licensed users has to be increased. The only negative side effect for declaring more ports than are actually used is that it takes one frame per port on disk. For example, on a 64-licensed-user system, reserving 128 ports will "waste" 64 frames on disk, which is negligible. The default is "512".

nphts 64
This is the number of phantom ports expected to be used. It should be at least 2. The rule of thumb to determine the number of phantoms is to take one-eighth of the number of licensed users, with a minimum of eight phantoms. This should cover most needs. The default is "64".

basic 1024
This optional field is the additional amount of shared memory set aside to contain the shared Pick/BASIC code produced by FlashBASIC. The size of this segment should be equal to or greater than the size of all FlashBASIC objects that are expected to be active at one time, that is, it should be large enough to be able to contain all of the FlashBASIC code most commonly used by the application. This amount is some fraction of the total disk space required for FlashBASIC object. Failure to set this parameter sufficiently high can result in extremely poor performance and a large increase in required Unix swap space. The usage of this segment can be monitored using the TCL command shpstat. If this field is empty, the code produced by the FlashBASIC compiler will not be shared. The minimum recommended size for this parameter is "1024" (1 Megabyte), but some FlashBASIC applications may require somewhere between "10240" - "15360" (10-15 Megabytes). If you are upgrading your system from a release without FlashBASIC to a release with FlashBASIC, this parameter may have a value of zero. Be sure to manually update this value before using FlashBASIC on the new release. The default values are:
On AIX(with FlashBASIC): 12.5% of the total memory
(without FlashBASIC): 0
On DG/UX (with FlashBASIC): 12.5% of the total memory
(without FlashBASIC): 0
On SCO(with FlashBASIC): (real memory - kernel size) * 10%
(without FlashBASIC): 0
SCO OpenServer systems are limited to a maximum of "4096".

brkchr 00
Break character. This allows redefining a character to simulate a <Break> key for use with devices that do not have a <Break> key, like some graphics terminals. The value is the hexadecimal value of the character. To disable the <Break> character completely, put "ff" in this field.
On AIX:the default value is "00", which is the <Ctrl>@, or <Break> key.

On DG/UX: the default value is "1f", which is the <Ctrl>_ key.
On SCO:the default value is "1d", which is the <Ctrl>] key.

escchr 1b
Escape (push level) character. This allows redefining a character to simulate an <Escape>, or D3 "push level" function for use with applications which need both the level pushing associated with the <Escape> function and the <Esc> character. The value is the hexadecimal value of the character. To disable the <Escape> (push level) function, but not the <Esc> key as a normal character, put "ff" in this field.
On AIX:the default value is "1b", which is the <Esc> key.
On DG/UX: the default value is "1b", which is the <Esc> key.
On SCO:the default value is "1b", which is the <Esc> key.

xon 11
XON character. This allows redefining a character for flow control for applications where the default value (<Ctrl>Q) is not suitable. The value after the statement is the hexadecimal value of the character. To disable the flow control, put "ff" in this field. The default value is "11".

xoff 13
XOFF character. This allows redefining a character for flow control for applications where the default value (<Ctrl>S) is not suitable. The value after the statement is the hexadecimal value of the character. To disable the flow control, put "ff" in this field. The default value is "13".

abssize 5e8
Hexadecimal value of the size (in frames) of the boot ABS. Changing this value on a "live" system will cause data corruption. The value varies with the release.

absbase 24
Base frame-id (fid) of the boot ABS. Changing this value on a "live" system will cause data corruption. The default is "24".

blkfid 2
Blocking factor for the frames in the Unix read/write queue. Changing this value on a "live" system will cause data corruption. This should be "2".

abslock on
Lock the ABS into memory. Turning this option off saves about 2 Mb in memory, but performance loss is significant, so this option should always be on. This option will also create a unique shared memory segment for the abs and all processes will attach to it with read only permissions. This will keep the abs area from becoming corrupted. To disable this feature use the absprotect command in this configuration file or use the -b option to the executable on the command line.
On AIX: AIX systems usually come with at least 8 Mb of physical memory, so it is advised that you leave this set to "on".
On SCO: If the amount of physical memory is less than 2.5 Mb, then it may be turned off.
The default value is "on".

absprotect off
ABS write protection. Setting this value to off will disable the protection of the abs on all pick processes so any can write into the area of memory where the abs are loaded. To disable the absprotection on a single line use the "-b" option to the d3 command.

flush 10
Flush wait period. The number, in seconds, to wait before the system begins

to flush memory back out to disk. The smaller the number, the more frequently the flush takes place and a higher number of disk writes occur. A value of "-1" indicates that the wait period is infinite (flushing is not done). A value of 10 to 20 seconds should be adequate for most systems. A value of "30" should be considered the maximum. The default value is "10". See "Performance Monitoring" for more information.

fflush 10
  Force Flush period. Buffers that reach this age, in seconds, are force-flushed out to disk, so that no frame will remain un-updated for long periods of time. The smaller the number, the younger the "retirement age" of the buffers. A value of "-1" indicates that force flushing is not done. A value of 30 to 60 seconds should be adequate for most systems. The default value is "10". See "Performance Monitoring" for more information.

dwqnum 256
  Defines the length (in bytes) of the internal write queue. Increasing this queue reduces the probability of the situation in which the flusher awakened on critical demand, thus reducing the number of flushes. The drawback to increasing the write queue size is that the flusher works in "bursts", which may overload the disk I/O channel when this phenomenon occurs. The default is "256".

```
disk /dev/rdsk/prod1 0 1222752  # disk 0
disk /dev/rdsk/prod2 0 1600000  # disk 1
```
  Disk statement(s). These statements define the D3 filesystem for the virtual machine.
  The first parameter is "disk".
  The second parameter is the Unix file path of the virtual disk.
  The third parameter is the offset in virtual disk (usually "0").
  The fourth parameter is the size of the virtual disk (may be "1" - "2097151" kilobytes).
  On AIX:There may be up to 64 disk statements. The disk size is obtained by multiplying the number of PP's (Physical Partitions) by 4096.
  On DG/UX: There may be up to 64 disk statements. The size of the divisions is shown in the diskman utility in terms of blocks, which are 512 bytes each.
  On SCO:There may be up to 16 disk statements. The size of the disk divisions (or partitions) are displayed in 1 Kb blocks.
  For maximum performance, try to place these divisions on different disk drives from each other, and on a different disk than the Unix swap area (disk "striping").
  WARNING
  Changing the order of the disk statements on a live file system will cause data corruption. If you wish to increase the file system size by adding disk statements, append the additional disk statements to the end of the list.

```
tape  /dev/rfd0h        500  f  lq  # 3.5 floppy 1.44M high density
tape  /dev/rfd0l        500  f  ld  # 3.5 floppy  720K low  density
tape  /dev/rmt1.1     16384  q  lh  # SCT high density
tape  /dev/rmt1.5     16384  q  ls  # SCT low  density
tape  /dev/rmt0.1     16384  v  l   # 8mm tape
tape  /home/tmp/floppy  500  f  lx  # temporary floppy device
tape  /dev/pipein       500  c  lx  # data in/out from/to 'some other source'
tape  /tmp/pseudo       500  p  lx
```

  Tape statement(s). These statements define the tape devices available for use by the D3 virtual machine.
  The first parameter is "tape".
  The second parameter is the Unix filepath of the tape device. This device must already exist (with any drivers already loaded) for D3 to properly use the device. The device paths listed above are examples only; your device paths may be different. Note that these filepaths refer to the "no rewind" devices, i.e. 'rmt0.1'.
  The third parameter is the device's block size.

```
  * For floppy devices, this is usually "500".
  * For half inch tape devices, this is usually "4096" or "8192".
  * For other tape devices, this is usually "16384".
  See the "t-att  command for more information.
  The fourth parameter specifies the type of tape ("magnetic
media") device.
  Device                       Device Code
  --------------------------  -----------
  floppy diskette                   f
  quarter inch streaming tape       q
  4mm digital tape                  d
  8mm digital tape                  v
  half inch tape h
  other devices                     c      no rewind device (pipe)
  pseudo tape                       p      cascading

  * For pseudo tape devices, data is written compressed and read uncompressed,
using standard compress/uncompress Unix utilities. At a 1GB boundary (before
compression), the Unix file name changes and cascades the pseudo-tape device
to that file name. A data save of more than 1GB cascades to a file
/tmp/pseudo.01, and after 2GB of data cascades to a file /tmp/pseudo.02 and so
on to /tmp/pseudo.0n. The suffix (number of files) may increment indefinitely.
Note: there must be space in the Unix file system, /tmp in this example, to
store these compressed files.

  The fifth parameter specifies the device's read/write density.
          Default  Low      Medium   High
  Device  Density  Density  Density  Density
  ------  -------  -------  -------  -------
  f         .        ld       .        lq
  q         l        ld       .        lh
  d         l        .        .        .
  v         l        .        .        .
  h         l        ld       lm       lh
  c         lx       .        .        .
  The line may contain a "#" (pound sign), which is a comment indicator. The
system ignores any characters following the "#", so you may include a
descriptive statement describing each device. Since tape devices are merely
Unix filenames, you may specify a separate Unix file area as a device, and
read and write data to it at a high speed. Such uses may be:
  * a copy of the ABS diskettes in a pseudo-tape for fast reloading.
  * high-speed file-save backups to a Unix file.
  * a serial device or network link for transmitting data to some other
destination.
  Tape devices are numbered (for use in the set-device command) starting at
"0" from the top of the list. The first device in the list is the default
device.
  On AIX:There may be up to 16 tape statements.
  On DG/UX: There may be up to 16 tape statements.
  On SCO:There may be up to 16 tape statements.
  Typical tape device names (although your device names may very):
  On AIX:
  /dev/rfd0h                  500  fld# 3-1/2" floppy - 1.44 Mb
  /dev/rfd0l                  500  fld# 3-1/2" floppy -  720 K
  /dev/rmt0.1               16384   vl# 8mm tape
  /dev/rmt1.1               16384  qlh# SCT - high density
  /dev/rmt1.5               16384  qls# SCT - low density
  /dev/rmt2.1                4096  hlh# 1/2" tape -high density  (6250 bpi)
  /dev/rmt2.3                4096  hlm# 1/2" tape -med. density  (3200 bpi)
  /dev/rmt2.5                4096  hls# 1/2" tape - low density  (1600 bpi)
  On DG/UX:
  /dev/rpdsk/                2500  flq# 3-1/2" floppy - 1.44 Mb
```

```
  /dev/rpdsk/                3500  flh# 5-1/4" floppy - 1.2  Mb
  /dev/rmt/0n               16384  qlh# SCT tape
  /dev/rmt/1n               16384   dl# 4mm DAT tape
  /dev/rmt/1n               16384   vl# 8mm tape
  /dev/rmt/2mn              16384  hls# 1/2" tape
  /usr/opt/pick/bin/abs       500  flx# Pseudo-floppy ABS restore
  /usr/opt/pick/bin/datafiles 500  flx# Pseudo-floppy Data Files restore
  On SCO:
  /dev/rdsk/f0q18dt           500  flq# 3-1/2" floppy - 1.44 Mb
  /dev/rdsk/f0q9dt            500  fld# 3-1/2" floppy -  720 K
  /dev/rdsk/f1q15dt           500  flh# 5-1/4" floppy -  1.2 Mb
  /dev/rdsk/f1q9dt            500  fls# 5-1/4" floppy -  360 K
  /dev/nrct0                16384  ql/dev/xct0   # SCT (120 Mb)
  /dev/rStp0                16384  vl/dev/xStp0  # 8mm tape
  /dev/nmt0                  4096  hls# 1/2" overland tape

install  /dev/rfd0h
  The tape device used to install D3. As this is only used to facilitate the
installation (and re-installations), this may be a previously-declared tape
device.
  Default install device names (your install device name may vary):
  On AIX:/dev/rfd0h
  On DG/UX: /dev/rmt/0n
  On SCO:/dev/rdsk/f0q18dt
```

rmbi

removes one or more functions from the list of Unix or C built-in functions.

"function.list" is a list of one or more item-ids (separated by spaces) indicating functions to add to the existing built-in functions.

Functions in the "function.list" list are no longer available to be called from within FlashBASIC. If "function.list" is not specified, then a list is presumed to be active. If no list is active, the file-defining the user built-in functions is rebuilt to match the list of built-in functions.

Syntax

rmbi {function.list}

tcl

allows execution of D3 TCL commands directly from a Unix shell.

It is installed as a Unix shell script during D3 installation. To function properly, the "tcl" command requires that the user set several shell environment variables appropriately:

PICKVM is the D3 virtual machine name. If this is not set, then the default of "pick0" is assumed.

PICKUSER is the D3 user name. If not set, the D3 user is assumed to be the same as the Unix user.

PICKUPASS is the D3 user password. This need only be set if a user password is present.

PICKMD is the D3 master dictionary. This must be set unless the user macro automatically logs into a dictionary.

PICKMDPASS is the D3 master dictionary password. This need only be set if a master dictionary password is present.

Several variants of the "tcl" command are available.

The "tcl" command by itself will push a D3 TCL shell above the current Unix one. This is equivalent to the Unix "sh" command. Logging off of D3 will automatically return to Unix. Note that although no logon messages appear, the login prompt and user macro are executed without output. The master dictionary macro is executed normally.

The "tcl {tcl.command}" form executes a single TCL command and returns to Unix. Remember that the Unix shell must parse the TCL command before sending it to D3, and therefore, all special characters, including "(", must be escaped to avoid errors. Also note that the "tcl" command logs into and out of D3 on every execution.

The "tcl -boot" form will attempt to boot the desired D3 virtual machine. When the boot is completed, D3 returns control to Unix rather than going to a D3 logon. This can be useful for shell scripts that are automatically run when the Unix machine is booted.

Syntax

tcl

tcl {tcl.command}

tcl -boot

Options

-boot Boots the D3 machine.

Example

```
The following sequence of commands sets up the necessary environment
variables, pushes a D3 TCL shell, executes a few TCL commands, and
exits back to the Unix shell. Note that this example shows input and
output, and that "bsh" or "ksh" is assumed to be the active Unix
shell.

$ PICKUSER="dm"; export PICKUSER
$ PICKMD="dm"; export PICKMD
$ tcl
:who
6 dm dm
:time
08:14:12  01 Mar 1993  Monday
:off
$

The "tcl" command may be easily incorporated into Unix shell scripts.
The following example is a shell script called "up" which invokes the
D3 Update processor from a Unix shell:

#Shell script to run Update processor
exec tcl u $*

This shell script should be typed into an editor like "vi" and saved
into the Unix file system. Then it must have its permissions changed
so that it is executable with a "chmod a+x up" command. Finally, it
must be placed in a directory which is contained in all users'
"$PATH" variables. The users may have to log off and log back in
again in order to recognize the new command. To run the shell script,
users can type something like:

up bp date

This will invoke the Update processor on the "date" program in the
"bp" file. It may be filed, or even run (with the ctl-X R command).
When finished, the Update processor will return back the Unix shell.
```

```
The "tcl" command also gives the capability of transferring data from
D3 to Unix via pipes. The following example places a partial
listing of the D3 "bp" file into the Unix file "bp.list".

tcl list bp sampling 5 \(hcnb | tr -d "\015" > bp.list

There are several points to note about this statement. First, notice
that the "(" command must be preceded by a back-slash because the
Unix shell would otherwise interpret the "(" as a special character.
Also notice that the output must first be piped into the "tr" command
which strips off carriage returns. This is necessary because D3
outputs in "raw" mode.

In the next example, various Unix shell capabilities are
demonstrated.  The first line puts the name of the first program in
the "bp" file into the Unix shell variable "PROGRAM".  Next, that
variable is "export"ed so that sub-shells can access it.  Finally,
that first program in the "bp" file is copied into an item called
"backup" and the results of the operation are stored in the Unix
"log" file.

PROGRAM=`tcl list bp sampling 1 \(hcnb | tr -d "\015"`
export PROGRAM
echo "backup\r" | tcl copy bp $PROGRAM \(o | tr -d "\015" > log

If the "\r" is left off on the input pipe, the command will terminate
and return to Unix without doing anything.

A different mode is used if the actual commands are piped into the
"tcl" program as in the following example. Note that this example
shows input and output, and that "bsh" or "ksh" is assumed to be the
active Unix shell.

$ echo "select bp with a0 \"date]\"\rcompile bp\r" | tcl
:select bp with a0 "date]"

[404] 2 items selected out of 279 items.
:compile bp
date
***************  ...
[241] successful compile!   3 frames used.
date.iconv
******
[241] Successful compile!   1 frame(s) used.
:

When commands are piped in as the only input to the "tcl" statment,
D3 behaves exactly as if those commands were typed directly at the
keyboard.  All simulated input and all output, including prompts, are
sent to standard-out.  Also note that multiple commands may be passed
to the "tcl" program using this mode and that select lists will be
preserved between each of those commands provided they are executed
within a single execution of the "tcl" program.
```

unix

invokes the "sh" (Unix) command, to 'push' a shell, and automatically returns to D3 upon completion.

To return to D3, type 'exit' at the Unix prompt.

The user can specify an optional command string to the Unix command. This command string can include D3 file and item pairs which will interact with the specified Unix command exactly as if those parameters were Unix files. Any changes made to the specified parameters by the Unix command will also be reflected in the D3 items after the command has completed.

During the substitution of D3 items for Unix files, attribute marks are converted to new-lines and groups of 4 spaces are converted to tabs. When items are brought back from the Unix command, these conversions are reversed.

Options can be placed after a dash as the first argument on the command line or can be put into attribute 6 of the "unix" master dictionary item.

Syntax

unix {-options} {unix.cmd} {{D3.file D3.item}}

Options

{number} Indicates the number of spaces which represent a tab.

c Do not convert attribute marks to new-lines and vice-versa.

i Use item name as export tag. This is useful when a Unix based word-processing package is used and it is desirable that the filename eventually passed to that package be related to the original D3 item name.

n No tab conversion.

v Convert value marks to new-lines. Note that this conversion is one-way and will loose the value marks at the completion of the command.

Example

```
unix spell {ba,doc, stock.report} > {dm,pointer-file, sp}

This example creates a list of all mis-spelled words existing in the
D3 item "stock.report" and places that list into an item called
"sp" in the "pointer-file".

unix -2 emacs {bp my.prog}

This allows editing the "my.prog" item with the "emacs" editor.
After exiting "emacs", any modifications to data will be updated in
"my.prog".
```

edits either a D3 item or a Unix file using the "vi" editor.

If passed a Unix file name, the "vi" command edits that Unix file.

If passed a D3 file and item list, then the "vi" command edits that item list.

The following options can be used on D3 items only.

Syntax

vi {D3.filename {D3.itemname {D3.itemname}...}} {(options}

vi {Unix.filename} {Unix.filename} ...

Options

c Do not convert between attribute marks and new-lines.

m Print an update message when the item has changed.

n Do not convert tabs.

r Use a random number tag rather than the D3 item name.

v Convert value marks to new-lines. Note that this works one-way only and that value marks will be lost when the item is updated in D3.

Example

```
vi dm,bp, date
```

Index

Customer Service

Pick Systems Customer Service representatives are available to provide assistance per the following schedule. All times listed are Pacific Time and are subject to change. Please have your software serial number available when calling. Your software serial number is printed on your D^3 diskettes or tapes or, at the TCL prompt, enter **which** to display the system serial number.

Monday-Friday 5:00 AM to 9:00 PM and Saturday 8:00 AM to 4:00 PM

By phone at (714)261-1875 or by fax at (714)261-5308 for VAR service, pre-paid service, and activation numbers.

Saturday 4:00 PM to Sunday 11:00 PM and during holiday closures

Extended support services are available with advance notice. Please call (714)261-1875, Monday through Friday, to make special arrangements.

Activations

Activations are available 24 hours a day, seven days a week at (714)261-1875, or through the World Wide Web at http://www.picksys.com/cgi-bin/activation/act.pl.

Holiday Closures

New Years Day, Memorial Day, Labor Day, Independence Day, Thanksgiving Day, and Christmas Day.

Support packages, including software upgrades, are available. Contact a Pick Systems sales person at (714)261-7425 for details.

If you have any other problems installing this product, please contact your VAR:

Pick Systems
Customer Service
1691 Browning
Irvine, CA 92606
United States

Phone: [1] 714/261-1875
Fax: [1] 714/261-5308

Pick Systems Ltd.
Cowdray House
2-4 High Street
SL9 9QA Chalfont St. Peter SL
United Kingdom

Phone: [44] 1753 891800
Fax: [44] 1753 891801

Pick Systems Africa (Pty.) Ltd.
P.O. Box 15277
Vlaeberg 8018 Capetown
South Africa

Phone: [27] 21-240656
Fax: [27] 21-247761

Pick Systems France
40, Avenue Des Terroirs De France
75012 Paris France

Phone: [3] 3144 745540
Fax: [3] 3144 745533

Reader's Comments

We hope you find this document useful and informative. We value your opinion, so please take a few minutes to complete the form below and return it to:

Pick Systems
Documentation Department
1691 Browning Ave.
Irvine, CA 92606
U.S.A.

Phone: 714/261-7425
Fax: 714/250-8187

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
This manual is well organized.	❒	❒	❒	❒	❒
This manual is complete.	❒	❒	❒	❒	❒
This manual is easy to read.	❒	❒	❒	❒	❒
The examples are clear and helpful.	❒	❒	❒	❒	❒
The index is thorough.	❒	❒	❒	❒	❒
I can find the information I need.	❒	❒	❒	❒	❒
This manual meets my expectations.	❒	❒	❒	❒	❒
I feel comfortable with this manual.	❒	❒	❒	❒	❒
Overall, I rate this manual highly.	❒	❒	❒	❒	❒

Additional comments are always welcome.