

The FreeM Manual

THE OFFICIAL MANUAL OF FREEM
Version 0.44.0

John P. Willis

This manual is for FreeM, (version 0.44.0), which is a free and open-source implementation of the M programming language and database system.

Copyright © 2014-2021 Coherent Logic Development LLC

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover texts, and with no Back-Cover Texts.

Table of Contents

Introduction	1
Production Readiness	1
Contributors	1
1 FreeM Invocation	3
1.1 Synopsis	3
1.2 Command-Line Options	3
1.3 Using FreeM for Shell Scripting	3
2 The FreeM Direct-Mode Environment	5
2.1 Direct-Mode Commands	5
2.2 REPL Functionality	7
3 Intrinsic Special Variables	8
3.1 \$DEVICE	8
3.2 \$ECODE	8
3.3 \$ESTACK	8
3.4 \$ETRAP	8
3.5 \$HOROLOG	8
3.6 \$IO	8
3.7 \$JOB	8
3.8 \$KEY	8
3.9 \$PRINCIPAL	8
3.10 \$REFERENCE	8
3.11 \$QUIT	9
3.12 \$STACK	9
3.13 \$STORAGE	9
3.14 \$SYSTEM	9
3.15 \$TEST	9
3.16 \$TLEVEL	9
3.17 \$TRESTART	9
3.18 \$WITH	9
3.19 \$X	9
3.20 \$Y	10
3.21 \$ZA	10
3.22 \$ZB	10
3.23 \$ZCONTROLC	10
3.24 \$ZDATE	10
3.25 \$ZERROR	10
3.26 \$ZHOROLOG	10
3.27 \$ZINRPT	10
3.28 \$ZJOB	10

3.29	\$ZLOCAL	10
3.30	\$ZPRECISION	10
3.31	\$ZREFERENCE	10
3.32	\$ZSYSTEM	11
3.33	\$ZTIME	11
3.34	\$ZTRAP	11
3.35	\$ZVERSION	11
4	Intrinsic Functions	12
4.1	\$ASCII	12
4.2	\$CHAR	12
4.3	\$DATA	12
4.4	\$EXTRACT	12
4.5	\$FIND	12
4.6	\$FNUMBER	13
4.7	\$GET	13
4.8	\$JUSTIFY	13
4.9	\$LENGTH	13
4.10	\$NAME	13
4.11	\$NEXT	13
4.12	\$ORDER	13
4.13	\$PIECE	13
4.14	\$QLENGTH	14
4.15	\$QSUBSCRIPT	14
4.16	\$QUERY	14
4.17	\$RANDOM	14
4.18	\$REVERSE	14
4.19	\$SELECT	14
4.20	\$STACK	14
4.21	\$TEXT	14
4.22	\$TRANSLATE	15
4.23	\$VIEW	15
4.24	\$ZBOOLEAN	15
4.25	\$ZCALL	15
4.26	\$ZCR	15
4.27	\$ZCRC	15
4.28	\$ZDATA	15
4.29	\$ZDATE	15
4.30	\$ZEDIT	16
4.31	\$ZHOROLOG	16
4.32	\$ZHT	16
4.33	\$ZKEY	16
4.34	\$ZLENGTH	16
4.35	\$ZLSD	16
4.36	\$ZM	16
4.37	\$ZNAME	16
4.38	\$ZNEXT	16
4.39	\$ZORDER	16

4.40	\$ZPIECE	16
4.41	\$ZPREVIOUS	16
4.42	\$ZREPLACE	16
4.43	\$ZSYNTAX	16
4.44	\$ZSORT	16
4.45	\$ZTIME	16
5	Commands	17
5.1	!	17
5.2	!!	17
5.3	ABLOCK	17
5.4	ASSERT	18
5.5	ASTART	18
5.6	ASTOP	18
5.7	AUNBLOCK	19
5.8	BREAK	19
5.9	CLOSE	19
5.10	CONST	20
5.11	DO	20
5.12	ELSE	20
5.13	FOR	20
5.14	GOTO	20
5.15	HALT	20
5.16	HANG	20
5.17	IF	20
5.18	JOB	20
5.19	KILL	20
5.20	KSUBSCRIPTS	20
5.21	KVALUE	21
5.22	LOCK	21
5.23	MERGE	21
5.24	NEW	21
5.25	OPEN	22
5.26	QUIT	23
5.27	READ	23
5.28	SET	23
5.29	TCOMMIT	23
5.30	TRESTART	23
5.31	TROLLBACK	23
5.32	TSTART	23
5.33	USE	23
5.34	VIEW	24
5.35	WATCH	24
5.36	WITH	24
5.37	WRITE	25
5.38	XECUTE	25
5.39	ZALLOCATE	25
5.40	ZBREAK	25

5.41	ZDEALLOCATE	25
5.42	ZGO	25
5.43	ZHALT	25
5.44	ZINSERT	25
5.45	ZJOB	25
5.46	ZLOAD	25
5.47	ZNEW	25
5.48	ZPRINT	25
5.49	ZQUIT	25
5.50	ZREMOVE	25
5.51	ZSAVE	25
5.52	ZTRAP	25
5.53	ZWRITE	25
6 Structured System Variables		27
6.1	^\$CHARACTER	27
6.2	^\$DEVICE	27
6.3	^\$DISPLAY	28
6.4	^\$EVENT	28
6.5	^\$GLOBAL	28
6.6	^\$JOB	29
6.7	^\$LOCK	30
6.8	^\$PDISPLAY	30
6.9	^\$ROUTINE	30
6.10	^\$SYSTEM	30
6.11	^\$WINDOW	30
6.12	^\$ZPROCESS	30
7 Operators		32
7.1	Unary +	32
7.2	Unary -	32
7.3	+ (Add)	32
7.4	+= (Add/Assign)	32
7.5	++ (Postfix Increment)	32
7.6	- (Subtract)	32
7.7	-= (Subtract/Assign)	32
7.8	-- (Postfix Decrement)	32
7.9	* (Multiply)	32
7.10	*= (Multiply/Assign)	32
7.11	/ (Divide)	32
7.12	/= (Divide/Assign)	32
7.13	\ (Integer Divide)	32
7.14	\= (Integer Divide/Assign)	32
7.15	# (Modulo)	32
7.16	#= (Modulo/Assign)	32
7.17	** (Exponentiate)	32
7.18	**= (Exponentiate/Assign)	32
7.19	< (Less Than)	32

7.20	<= (Less Than or Equal To)	33
7.21	> (Greater Than)	33
7.22	>= (Greater Than or Equal To)	33
7.23	_ (Concatenate)	33
7.24	_= (Concatenate/Assign)	33
7.25	= (Equals)	33
7.26	[(Contains)	33
7.27] (Follows)	33
7.28]] (Sorts After)	33
7.29	? (Pattern Match)	33
7.30	& (Logical AND)	33
7.31	! (Logical OR)	33
7.32	' (Logical NOT)	33
7.33	@ (Indirect)	33
8	Sequential I/O	34
9	Network I/O	35
9.1	Opening and Connecting a Client Socket	35
10	Asynchronous Event Handling	36
10.1	Setting Up Async Event Handlers	36
10.2	Registering an Asynchronous Event Handler	36
10.3	Enabling Asynchronous Event Handling	37
10.4	Disabling Asynchronous Event Handling	37
10.5	Temporarily Blocking Asynchronous Event Handling	37
11	Synchronous Event Handling	38
12	GUI Programming with MWAPI	39
13	User-Defined Z Commands	40
14	User-Defined Z Functions	41
15	User-Defined SSVs	42
16	System Library Routines	43
16.1	^%ZCOLUMNS	43
16.2	^%ZFREEM	43
16.3	^%ZHELP	43
16.4	^%ZROWS	43
17	Error Processing	44

18	Debugging	45
19	System Configuration	46
19.1	Installing FreeM	46
20	Accessing FreeM from C Programs	47
20.1	freem_ref_t Data Structure	47
20.2	freem_ent_t Data Structure	48
20.3	freem_init()	49
20.4	freem_version()	50
20.5	freem_set()	50
20.6	freem_get()	51
20.7	freem_kill()	52
20.8	freem_data()	53
20.9	freem_order()	53
20.10	freem_query()	53
20.11	freem_lock()	53
20.12	freem_unlock()	53
20.13	freem_tstart()	53
20.14	freem_trestart()	53
20.15	freem_trollback()	53
20.16	freem_tlevel()	54
20.17	freem_tcommit()	54
20.18	freem_function()	54
20.19	freem_procedure()	54
Appendix A	FreeM Administrator	55
Appendix B	FreeM Legacy Utilities	57
B.1	Global Compactor (gcompact)	57
B.2	Block Examiner (gfix)	57
B.3	Global Repair Tool (grestore)	57
Appendix C	FreeM VIEW Commands and Functions	58
C.1	VIEW 16: Total Count of Error Messages/View Single Error Message	58
C.2	VIEW 17: Intrinsic Z-Commands	58
C.3	VIEW 18: Intrinsic Z-Functions	58
C.4	VIEW 19: Intrinsic Special Variables	58
C.5	VIEW 20: Break Service Code	58
C.6	VIEW 21: View Size of Last Global	58
C.7	VIEW 22: Count VIEW 22 Aliases	58
C.8	VIEW 23: View Contents of Input Buffer	58
C.9	VIEW 24: Maximum Number of Screen Rows	58
C.10	VIEW 25: Maximum Number of Screen Columns	58

C.11	VIEW 26: DO/FOR/XECUTE Stack Pointer	58
C.12	VIEW 27: DO/FOR/XECUTE Stack Pointer (On Error)	59
C.13	VIEW 29: Copy Symbol Table	59
C.14	VIEW 30: Inspect Arguments	59
C.15	VIEW 31: Count Environment Variables	59
Appendix D Implementation Limits		60
Appendix E US-ASCII Character Set		61
Appendix F FreeM Project Coding Standards		
	62
F.1	Module Headers	62
F.2	Variable Naming	62
F.3	Indentation and General Layout	62
F.4	Brace Placement (Functions)	63
F.5	Brace Placement (if-for-while-do)	63
F.6	Labels and goto	64
F.7	Preprocessor Conditionals	64
F.8	coding standards, preprocessor conditionals	64
F.9	Overall Program Spacing	64
F.10	The switch() Statement	64
F.11	Comments	65
Index		66

Introduction

FreeM started its life as *FreeMUMPS*, written for MS-DOS and ported to SCO UNIX by a mysterious individual going by the name of "Shalom ha-Ashkenaz". It was released to MUG Deutschland in 1998. In 1999, Ronald L. Fox ported FreeM to the Red Hat Linux 5 of the GNU/Linux operating system. Thereafter, maintenance was taken over by the Generic Universal M Project, which changed its name first to Public Standard MUMPS and then by popular request to FreeM.

When GT.M was open-sourced in late 1999, FreeM and GUMP were essentially abandoned. L.D. Landis, the owner of the original GUMP SourceForge project, and one of FreeM's significant contributors, passed maintenance of FreeM and ownership of its SourceForge project to John Willis in 2014. At this point, FreeM would not compile or run on modern Linux systems, so steps were taken to remedy the most pressing issues in the codebase. Limitations on the terminal size (previously hard-coded to 80x25) were lifted, and new `$VIEW` functions were added to retrieve the terminal size information. `$X` and `$Y` intrinsic special variables were updated to support arbitrary terminal sizes, and FreeM was once again able to build and run.

In February of 2020, work began in earnest to build a development and support infrastructure for FreeM and begin the careful process of refining it into a more stable and robust product.

Production Readiness

FreeM is not yet production-ready. There are several show-stopping bugs that preclude a general release for public use:

- SSVs, aside from `^$JOB`, `^$DEVICE`, `^$EVENT`, and `^$ZPROCESS` are not implemented.
- `VIEW` commands and `$VIEW` functions are used extensively to configure and inspect the run-time behavior of FreeM, rather than the "canonical" SSV-based approach.
- Server sockets are not yet implemented.

Contributors

Current contributors denoted with a + following their name and role.

- Shalom ha-Ashkenaz (Original Implementer)
- John Best (IBM i and OS/400)
- Jon Diamond (Library, Utilities, Conformance)
- Ronald L. Fox (Initial port to Red Hat 5/libc-6)
- Winfried Gerum (Code, Advice, MTA coordination)
- Greg Kreis (Hardhats coordination, Dependencies)
- Larry Landis (Coordination, Code, Documentation)
- Frederick D.S. Marshall (MDC Standards Conformance) +
- Lloyd Milligan (Code, Testing, Documentation)
- Steve Morris (Code, Microsoft)
- John Murray (Code, Conformance)

- Wilhelm Pastoors (Testing, Documentation)
- Kate Schell (Coordination, Conformance, MTA, MDC, Advice)
- Lyle Schofield (Advice, Prioritization, Tracking, Project Management)
- Jim Stefanik (GNU/Linux on s390x, IBM AIX, IBM z/OS)
- Axel Trocha (Code, Utilities)
- Dick Walters (Project Lead, Chief Coordinator, MTA)
- David Whitten (QA Test Suite, MDC, Advice) +
- David Wicksell (Debugging, Code, Testing) +
- John Willis (Current Maintainer and Project Lead) +
- Steve Zeck (Code)

1 FreeM Invocation

1.1 Synopsis

```
$ ./freem [OPTIONS...] [[-r <entryref>] | [--routine=<entryref>]]
```

When FreeM loads, it searches the `SYSTEM` namespace for the `^%ZFREEM` routine, and begins executing it.

When `-r` or `--routine` are passed on the command line, FreeM will load and run the specified routine instead of `^%ZFREEM`. Beginning with FreeM 0.1.7, routines invoked in this manner are no longer required to perform their own namespace setup with `VIEW` commands.

1.2 Command-Line Options

`'-c', '--config'`

Specify a configuration file other than `$PREFIX/etc/freem.conf`.

`'-h', '--help'`

Display a help message showing valid FreeM options.

`'-i', '--import'`

Causes your UNIX environment variables to be imported into FreeM's local symbol table.

`'-f', '--filter'`

Allows your M routines to be used as UNIX filters.

`'-n <namespace-name>', '--namespace=<namespace-name>'`

Selects the FreeM namespace to be entered on startup. Must be defined in `'/etc/freem.conf'`.

`'-r <entryref>', '--routine=<entryref>'`

Causes `<entryref>` to be executed at load, instead of `^%ZFREEM`.

`'-v', '--version'`

Displays FreeM version information.

`'-x <mcode>', '--execute=<mcode>'`

Executes M code `<mcode>` at startup instead of the startup routine.

1.3 Using FreeM for Shell Scripting

FreeM M routines can be used as shell scripts by providing a *shebang* line beginning with `#!/path/to/freem` as the first line of the routine. The following example presumes that FreeM is installed at `'/usr/local/bin/freem'` and uses the `USER` namespace:

```
#!/usr/local/bin/freem
MYSCRIPT ;
SET ^$JOB($JOB,"NAMESPACE")="USER"
WRITE "This is output from an M routine used as a shell script.",!
Q
```

Currently, the script needs to have a '.m' file extension. You will also need to select an appropriate namespace in your script using the `SET ^$JOB($JOB, "NAMESPACE")=<namespace>` command before attempting to call other routines or access globals.

You will also need to set the script's permissions to *executable* in order for this to work:

```
$ chmod +x myscript.m
```

2 The FreeM Direct-Mode Environment

The FreeM direct-mode environment is the mode entered when FreeM is loaded without the use of ‘-r <entryref>’ or ‘--routine=<entryref>’:

```
Coherent Logic Development FreeM version 0.44.0 (x86_64-pc-linux-gnu)
Copyright (C) 2014, 2020, 2021 Coherent Logic Development LLC
```

```
USER>
```

The prompt (USER>) indicates the currently-active namespace. If any uncommitted direct-mode transactions have been started, the prompt will change to reflect the current value of \$TLEVEL:

```
TL1:USER>
```

In the above example, TL1 indicates that \$TLEVEL is currently 1.

2.1 Direct-Mode Commands

When you are in direct mode, in addition to M commands, a number of internal commands are available to help developers be more productive:

? Accesses FreeM online help. Requires GNU `info(1)` to be installed on your local system.

events Writes a list of *event classes* and their ABLOCK counts:

```
USER> events
```

Event Class	Processing Mode	ABLOCK Count
-----	-----	-----
COMM	Disabled	0
HALT	Disabled	0
IPC	Disabled	0
INTERRUPT	Disabled	0
POWER	Disabled	0
TIMER	Disabled	0
USER	Disabled	0
WAPI	Disabled	0

history Prints a list of all the direct-mode commands you have entered across all sessions.

rcl <history-index>

Allows you to recall command number <history-index> and run it again. Obtain the value for <history-index> from the output of the `history` command.

!! Launches a subshell within the FreeM direct mode, allowing the user to run operating system commands.

```
USER> !!
```

```
Type Ctrl-D to exit from the shell
```

```
$ uname -a
Linux hesperos 4.19.0-17-amd64 #1 SMP Debian 4.19.194-3 (2021-07-18) x86_64
$ exit
```

```
USER>
```

!*<external-command>*

Invokes a shell to run *<external-command>* from within FreeM. This temporarily disables SIGALRM handling in FreeM, which may interrupt the use of event-driven M programming commands including ESTART and ESTOP.

If the < character is supplied immediately preceding *<external-command>*, FreeM will append the contents of M local variable % to *<external-command>* as standard input.

If the > character is supplied immediately preceding *<external-command>*, FreeM will take the standard output stream of *<external-command>* and store it in M local variable %.

% contains the number of lines in the input or output. %(1) . . %(n) contains the data for lines 1-*n*.

tdump Writes detailed information about the status of any pending transactions to \$PRINCIPAL.

If you issue a HALT command at the direct-mode prompt, you will exit out of FreeM. However, if you issue a HALT command when \$TLEVEL is greater than zero, you will be given the opportunity to commit or rollback any pending transactions:

```
USER> TSTART
```

```
TL1:USER> SET ^MYGLOBAL=1
```

```
TL1:USER> HALT
```

```
UNCOMMITTED TRANSACTIONS EXIST:
```

```
$TLEVEL 1*
```

```
Operations for Transaction ID: k8xj1de
```

```
1: action = 0 key = ^MYGLOBAL data = 1
```

```
Would you like to c)ommit or r)ollback the above transactions and their operations? ($
```

```
Transactions have been rolled back.
```

In the above example, the user selected r to rollback the single pending transaction.

2.2 REPL Functionality

FreeM direct mode allows you to enter M expressions directly from the direct-mode prompt, as long as they begin with a number:

```
USER> S DENOM=10
```

```
USER> 100/DENOM
```

```
10
```

```
USER>
```

Such expressions will be immediately evaluated, and the result printed on \$PRINCIPAL.

3 Intrinsic Special Variables

3.1 \$DEVICE

Returns the status of the device currently in use, and is writable.

If \$DEVICE returns *1*, an error condition exists on the current device.

3.2 \$ECODE

Returns a comma-delimited list of error conditions currently present, and is writable. An empty \$ECODE indicates no errors.

3.3 \$ESTACK

Returns the depth of the program execution stack since the last time \$ESTACK was NEWed. NEW-able, but not SET-able. Differs from the \$STACK ISV in that it is NEW-able, and resets to a value of 0 when NEWed.

3.4 \$ETRAP

Sets or retrieves the M code that is run when an error is encountered or \$ECODE is set to a non-blank value. \$ETRAP code executes when \$ECODE becomes non-blank.

3.5 \$HOROLOG

Returns a string containing the current date and time as <days>,<seconds>, where <days> represents the number of days since the M epoch (midnight on 31 December 1840), and <seconds> represents the number of seconds since the most recent midnight.

3.6 \$IO

Represents the current input/output device. Read-only.

3.7 \$JOB

Represents the process ID of the FreeM instance currently in use.

3.8 \$KEY

Represents the sequence of control characters that terminated the last READ command on \$IO.

3.9 \$PRINCIPAL

Represents the primary input/output device. Usually a terminal or virtual terminal.

3.10 \$REFERENCE

Returns the last *gln* referenced. Can be SET, and also stacked with NEW.

3.11 \$QUIT

If the current execution context was invoked as an extrinsic function, `$QUIT` returns `1`. Otherwise, returns `0`.

When `$QUIT` returns `1`, a subsequent `QUIT` command must have an argument.

3.12 \$STACK

Represents the current stack level.

3.13 \$STORAGE

Represents the number of bytes of free space available in FreeM's heap.

3.14 \$SYSTEM

Returns the MDC system ID of FreeM.

3.15 \$TEST

`$TEST` is a writable, `NEW`-able ISV that is `1` if the most recently evaluated expression was *true*. Otherwise, returns `0`.

`$TEST` is implicitly `NEW`ed when entering a new stack frame for extrinsic functions and argumentless `DO`. `$TEST` is *not* implicitly `NEW`ed when a new stack frame is entered with an argumented `DO`.

3.16 \$TLEVEL

Returns a numeric value indicating the current level of transaction nesting in the process. When `$TLEVEL` is greater than `0`, uncommitted transactions exist.

3.17 \$TRESTART

Returns an empty string, as FreeM transaction processing does not yet support restartable transactions.

3.18 \$WITH

Returns the variable prefix set by the `WITH` command.

3.19 \$X

Represents the current column position of the FreeM cursor.

In FreeM, setting `$X` will move the FreeM cursor.

3.20 \$Y

Represents the current row position of the FreeM cursor.

In FreeM, setting \$Y will move the FreeM cursor.

3.21 \$ZA

On the HOME device, always 0. On other devices, represents the byte offset to the beginning of the file.

3.22 \$ZB

Represents the last keystroke.

3.23 \$ZCONTROLC

3.24 \$ZDATE

Returns the current date, in YYYY/MM/DD format.

3.25 \$ZERROR

Returns the last error message.

3.26 \$ZHOROLOG

Output \$HOROLOG-style time, with the addition of milliseconds.

3.27 \$ZINRPT

Gets or sets the interrupt enable/disable flag.

3.28 \$ZJOB

Returns the \$JOB value of the parent process if the current process was started by a JOB command. Otherwise, returns an empty string.

3.29 \$ZLOCAL

Returns the last local variable referenced.

3.30 \$ZPRECISION

Gets or sets the number of digits of numeric precision used for fixed-point decimal arithmetic. Defaults to 100 digits.

3.31 \$ZREFERENCE

Returns the last *gln* referenced.

3.32 \$ZSYSTEM

3.33 \$ZTIME

Returns the system time in `HH:MM:SS` (24-hour) format.

3.34 \$ZTRAP

Sets or retrieves the entryref to be executed when an M program execution error occurs under FreeM-style or DSM 2.0-style error processing.

In FreeM-style error processing, `$ZTRAP` is specific to each program execution stack level.

In DSM 2.0-style error processing, `$ZTRAP` is the same for all program execution stack levels.

When FreeM encounters an error, if `$ZTRAP` is nonempty and `$ETRAP` is empty, FreeM will perform an implicit `GOTO` to the entryref indicated in `$ZTRAP`.

If `$ETRAP` is nonempty when FreeM encounters an error, the value of `$ZTRAP` is ignored, whether FreeM-style or DSM 2.0-style error processing is enabled.

3.35 \$ZVERSION

Returns the version of FreeM in use, as well as the GNU host triplet for the current FreeM build.

See https://wiki.osdev.org/Target_Triplet.

4 Intrinsic Functions

4.1 \$ASCII

Returns the ASCII code (in decimal) for one character in a string.

```
SET RESULT=$ASCII(<string>[,<index>])
```

If *<index>* is not supplied, \$ASCII will return the ASCII code of the first character. Otherwise, returns the ASCII code of the character at position *<index>*.

4.2 \$CHAR

Returns a string of characters corresponding to a list of ASCII codes.

```
SET RESULT=$CHAR(<ascii-code>[,<ascii-code>,...])
```

4.3 \$DATA

Returns a numeric value 0, 1, 10, or 11, depending on whether a referenced node is defined, has data, or has children:

```
SET RESULT=$DATA(<node>)
```

The return values are as follows:

```
0: <node> is undefined
1: <node> has data but no children
10: <node> has children but no data
11: <node> has children and data
```

4.4 \$EXTRACT

Extracts a substring of a string.

The first argument is the source string.

The optional second argument specifies the starting position of the substring to extract, and defaults to 1.

The optional third argument specifies the ending position of the substring to extract, and defaults to the value of the second argument, or 1.

This example will extract the string *FreeM* into the local variable M.

```
SET NAME="FreeM is the best!"
SET M=$EXTRACT(NAME,1,5)
```

4.5 \$FIND

Finds the character immediately following the first occurrence of a substring within a string.

The first argument is the source string.

The second argument is the substring to be located.

The optional third argument indicates the position within the source string at which to begin searching.

4.6 \$FNUMBER

Formats a number according to a particular set of formatting codes.

The first argument is the number to format.

The second argument is the series of formatting codes.

4.7 \$GET

Returns the value of a local, global, or SSVN if the specified item is defined, or a default value otherwise.

The first argument is the local, global, or SSVN to be examined.

The optional second argument is the default value to be returned if the referenced item is undefined, and defaults to the empty string.

4.8 \$JUSTIFY

Right-justifies a string based on a specified fixed length.

The first argument is the source string.

The second argument is the character length of the output.

The optional third argument controls the number of fractional digits to be included in the output, and defaults to the number of digits specified in the first argument.

4.9 \$LENGTH

Returns the length of a string, or the number of items in a list delimited by a specified character (as used by \$PIECE).

The first argument is the source string.

The optional second argument is the list delimiter to be used. When this argument is omitted, the length of the string in characters is returned.

4.10 \$NAME

Returns the canonical name reference along with some or all of its subscripts.

The first argument is the source name.

The optional second argument indicates the maximum subscript count to be returned, and defaults to the subscript count of the source name.

4.11 \$NEXT

4.12 \$ORDER

4.13 \$PIECE

SYNTAX:

`$PIECE(s, d[, n[, end]])`

Accesses the *n*th through *end* *d*-delimited pieces of string *s*.

The first argument is the string to be evaluated.

The second argument is the delimiter to be used.

The optional third argument is the first *d*-delimited piece to access, and defaults to 1.

The optional fourth argument is the final *d*-delimited piece to access, and defaults to the value of the third argument (*n*).

Can be used on the right-hand side of an expression in order to SET a value into a *d*-delimited piece of *s*.

4.14 \$QLENGTH

4.15 \$QSUBSCRIPT

4.16 \$QUERY

4.17 \$RANDOM

4.18 \$REVERSE

4.19 \$SELECT

4.20 \$STACK

Returns information about the program execution stack. The \$STACK intrinsic function has both a one-argument form and a two-argument form.

Syntax (One-Argument)

`$STACK(<num>)`

If *num* is 0, returns the command with which this FreeM instance was invoked.

If *num* is -1, returns the current program execution stack level.

If *num* represents a valid program execution stack depth above 0, returns one of the following values indicating the reason for which the referenced program execution stack level was created:

\$\$ If `$STACK(<num>)= "$$"`, program execution stack level *num* was created as the result of an extrinsic function call

<m-command>

If `$STACK(<num>)` returns a valid M command, the referenced program execution stack level was created as a result of the *m-command* command.

Syntax (Two-Argument)

`$STACK(<num>, "[ECODE|MCODE|PLACE] ")`

Returns the error codes, M program code, or entryref applicable to the action that created program execution stack level *num*.

4.21 \$TEXT

4.22 \$TRANSLATE

4.23 \$VIEW

4.24 \$ZBOOLEAN

Performs *boolean-operation* on numeric arguments *A* and *B*.

Syntax

```
SET RESULT=$ZBOOLEAN(A,B,boolean-operation)
```

\$ZBOOLEAN Operations (*boolean-operation* values)

0	Always <i>false</i>
1	A AND B
2	A AND NOT B
3	A
4	NOT A AND B
5	B
6	A XOR B
7	A OR B
8	A NOR B
9	A EQUALS B
10	NOT B
11	A OR NOT B
12	NOT A
13	NOT A OR B
14	A NAND B
15	Always <i>true</i>

4.25 \$ZCALL

4.26 \$ZCR

4.27 \$ZCRC

4.28 \$ZDATA

4.29 \$ZDATE

4.30 \$ZEDIT

4.31 \$ZHOROLOG

4.32 \$ZHT

4.33 \$ZKEY

4.34 \$ZLENGTH

4.35 \$ZLSD

4.36 \$ZM

4.37 \$ZNAME

4.38 \$ZNEXT

4.39 \$ZORDER

4.40 \$ZPIECE

4.41 \$ZPREVIOUS

4.42 \$ZREPLACE

4.43 \$ZSYNTAX

4.44 \$ZSORT

4.45 \$ZTIME

5 Commands

5.1 !

Invokes a shell to run *<external-command>* from within FreeM. This temporarily disables SIGALRM handling in FreeM, which may interrupt the use of event-driven M programming commands including `ESTART` and `ESTOP`.

If the `<` character is supplied immediately preceding *<external-command>*, FreeM will append the contents of M local variable `%` to *<external-command>* as standard input.

If the `>` character is supplied immediately preceding *<external-command>*, FreeM will take the standard output stream of *<external-command>* and store it in M local variable `%`.

`%` contains the number of lines in the input or output. `%(1)..(n)` contains the data for lines 1-*n*.

5.2 !!

Launches a subshell within the FreeM direct mode, allowing the user to run operating system commands.

```
USER> !!
```

```
Type Ctrl-D to exit from the shell
```

```
$ uname -a
```

```
Linux hesperos 4.19.0-17-amd64 #1 SMP Debian 4.19.194-3 (2021-07-18) x86_64 GNU/Linux
```

```
$ exit
```

```
USER>
```

5.3 ABLOCK

Increments the event block counter for one or more event classes. While the block counter for an event class is greater than zero, registered event handlers for that event class will not execute, and will instead be queued for later execution once the block counter reaches zero (all blocks removed).

An implicit `ABLOCK` on all event classes occurs when an event handler subroutine is executing. As soon as a `QUIT` is reached within an event handler, an implicit `ABLOCK` will occur.

Syntax

```
ABLOCK:postcondition
```

In its argumentless form, `ABLOCK` increments the block counter for *all* event classes, provided the optional *postcondition* is either *true* or omitted.

```
ABLOCK:postcondition evclass1...evclassN
```

In its inclusive form, `ABLOCK` increments the block counters for all event classes named in the list, provided the optional *postcondition* is either *true* or omitted.

```
ABLOCK:postcondition (evclass1...evclassN
```

In its exclusive form, `ABLOCK` increments the block counters for all event classes *except for* those named in the list, provided the optional *postcondition* is either *true* or omitted.

5.4 ASSERT

Triggers error ASSERT if the supplied truth-valued expression *tvexpr* is *false* (*1* is *true*, and *0* is *false*).

The ASSERT error is catchable whether using standard-style, FreeM-style, or DSM 2.0-style error processing.

Syntax

```
ASSERT <tvexpr>
```

Example

```
USER> ASSERT 1=1
```

```
USER> ASSERT 1=0
```

```
>> Error ASSERT: programmer assertion failed in SYSTEM::~%ZFREEEM [$STACK = 0]
>> ASSERT 1=0
^
```

5.5 ASTART

Enables asynchronous event handling for one or more event classes.

Syntax

```
ASTART:postcondition
```

In its argumentless form, ASTART enables asynchronous event handling for all event classes, provided the optional *postcondition* is either *true* or omitted.

```
ASTART:postcondition evclass1...,evclassN
```

In its inclusive form, ASTART enables asynchronous event handling for all event classes named in the list, provided the optional *postcondition* is either *true* or omitted.

```
ASTART:postcondition (evclass1...,evclassN)
```

In its exclusive form, ASTART enables asynchronous event handling for all event classes *except* for those named in the list, provided the optional *postcondition* is either *true* or omitted.

5.6 ASTOP

Disables asynchronous event handling for one or more event classes.

Syntax

```
ASTOP:postcondition
```

In its argumentless form, ASTOP disables asynchronous event handling for all event classes, provided the optional *postcondition* is either *true* or omitted.

```
ASTOP:postcondition evclass1...,evclassN
```

In its inclusive form, ASTOP disables asynchronous event handling for all event classes named in the list, provided the optional *postcondition* is either *true* or omitted.

ASTOP:postcondition (*evclass1...evclassN*)

In its exclusive form, **ASTOP** disables asynchronous event handling for all event classes *except* for those named in the list, provided the optional *postcondition* is either *true* or omitted.

5.7 AUNBLOCK

Decrements the event block counter for one or more event classes.

Syntax

AUNBLOCK:postcondition

In its argumentless form, **AUNBLOCK** decrements the block counter for *all* event classes, provided the optional *postcondition* is either *true* or omitted.

AUNBLOCK:postcondition *evclass1...evclassN*

In its inclusive form, **AUNBLOCK** decrements the block counters for all event classes named in the list, provided the optional *postcondition* is either *true* or omitted.

AUNBLOCK:postcondition (*evclass1...evclassN*)

In its exclusive form, **AUNBLOCK** decrements the block counters for all event classes *except* for those named in the list, provided the optional *postcondition* is either *true* or omitted.

5.8 BREAK

Interrupts running routine to allow interactive debugging.

Syntax

BREAK:postcondition

In its argumentless form, **BREAK** suspends execution of running code, provided the optional *postcondition* is *true* or omitted.

BREAK:postcondition *breakflag*

In its single-argument form, **BREAK** sets *Ctrl-C* handling and error handling characteristics, provided the optional *postcondition* is *true* or omitted. The following table enumerates the possible values of *breakflag*

0	Disables <i>Ctrl-C</i> handling
-2	Enables normal FreeM error handling
2	Enables <i>Digital Standard MUMPS v2</i> error handling
"default"	Enables <i>Ctrl-C</i> handling

5.9 CLOSE

Closes an input/output device.

Syntax

CLOSE:postcondition

In its argumentless form, **CLOSE** closes all I/O devices except for device 0 (the **HOME** device), provided the optional *postcondition* is *true* or omitted.

CLOSE:*postcondition channel*

In its single-argument form, **CLOSE** closes the I/O device associated with channel *channel*, provided that *channel* represents a currently-open device, and the optional *postcondition* is *true* or omitted.

5.10 CONST

Defines a local *constant*, or variable that cannot be altered after its initial definition, provided the optional *postcondition* is *true* or omitted.

Constants must only be locals, and globals are not supported.

Syntax

CONST:*postcondition mref1=initial-value1,...,mrefN=initial-valueN*

5.11 DO

5.12 ELSE

5.13 FOR

5.14 GOTO

5.15 HALT

5.16 HANG

5.17 IF

5.18 JOB

5.19 KILL

5.20 KSUBSCRIPTS

Kills only the descendant subscripts (but not the data value) of a referenced global, local, or SSV (where allowed).

Syntax

KSUBSCRIPTS:*postcondition var1,...*

In the above *inclusive* form, **KVALUE** will kill the descendant subscripts at each local, global, or SSV node specified in the list (provided that the optional *postcondition* is *true* or omitted), but will leave the data value intact.

Note The below *argumentless* and *exclusive* forms of **KSUBSCRIPTS** are not implemented in FreeM, as of version 0.3.3, but are planned for a future release.

KSUBSCRIPTS:postcondition

In the above *argumentless* form, KSUBSCRIPTS will kill the descendant subscripts at the root of each local variable (provided that the optional *postcondition* is *true* or omitted), but will leave data values intact.

KSUBSCRIPTS:postcondition (var1,...)

In the above *exclusive* form, KSUBSCRIPTS will kill the descendant subscripts of all local variables, *with the exception of* those named in the list, provided that the optional *postcondition* is *true* or omitted, while leaving their data values intact.

5.21 KVALUE

Kills only the data value (but not descendant subscripts) of a referenced global, local, or SSV (where allowed).

Syntax

KVALUE:postcondition var1,...

In the above *inclusive* form, KVALUE will kill the data values at each local, global, or SSV node specified in the list (provided that the optional *postcondition* is *true* or omitted), but will leave descendant subscripts intact.

Note The below *argumentless* and *exclusive* forms of KVALUE are not implemented in FreeM, as of version 0.44.0, but are planned for a future release.

KVALUE:postcondition

In the above *argumentless* form, KVALUE will kill the data values at the root of each local variable (provided that the optional *postcondition* is *true* or omitted), but will leave descendant subscripts intact.

KVALUE:postcondition (var1,...)

In the above *exclusive* form, KVALUE will kill the data values of all local variables, *with the exception of* those named in the list, provided that the optional *postcondition* is *true* or omitted, while leaving their descendant subscripts intact.

5.22 LOCK

5.23 MERGE

Merges the contents of one global, local, or SSV subtree to another global, local, or SSV.

Syntax

MERGE A=~\$JOB

The above example will merge the `~$JOB` SSV into the `A` local. Note that the FreeM implementation of `MERGE` does not yet support multiple merge arguments. Returns error `M19` if either the source or the target variable are descendants of each other.

5.24 NEW

5.25 OPEN

Opens sequential or socket I/O devices and files and associates them with a numeric FreeM input/output channel.

Syntax (Sequential Files)

```
OPEN:postcondition channel:"filename/access-mode"
```

Opens *filename* for reading and/or writing, and associates the file with FreeM I/O channel *channel*, provided that the optional *postcondition* is *true* or omitted. The below table lists the valid options for *access-mode*:

r	Read-only access
w	Create a new file for write access
a	Write access; append to existing file
r+	Read/write access

I/O Path

You cannot specify a fully-qualified filesystem path in the FreeM OPEN command. By default, FreeM will assume that *filename* exists in the directory indicated in `^$JOB($JOB,"CWD")`. If you wish to access files in other directories, you must first set the *I/O Path* in `^$JOB($JOB,"IOPATH")`.

The following example will set the I/O path to `/etc`:

```
SET ^$JOB($JOB,"IOPATH")="/etc"
```

If *channel* was already OPENed in the current process, calling OPEN on the same channel again implicitly closes the file or device currently associated with *channel*.

Syntax (Network Sockets)

Network sockets use a dedicated range of FreeM I/O channels ranging from 100-255. OPENing a socket I/O channel does *not* implicitly connect the socket. Connecting the socket to the specified remote host is accomplished by the /CONNECT control mnemonic supplied to the USE command.

```
OPEN:postcondition socket-channel:"hostname-or-address:port:address-family:connectio
type"
```

Socket Parameters

socket-channel

The socket I/O channel to use. This must be in the range of 100-255.

hostname-or-address

The hostname or IP address to connect to. If a hostname is supplied, OPEN will implicitly do a name lookup, the mechanism of which is typically determined by the configuration of `/etc/nsswitch.conf` on most UNIX and UNIX-like platforms.

port

The TCP or UDP port to which the socket will connect on the remote host.

address-family

The address family to use. Either *IPV4* or *IPV6*.

connection-type

Which connection type to use. Either *TCP* or *UDP*.

If you do not specify the address family and connection type, they will default to *IPV4* and *TCP*, respectively.

5.26 QUIT

5.27 READ

5.28 SET

5.29 TCOMMIT

5.30 TRESTART

5.31 TROLLBACK

5.32 TSTART

5.33 USE

Sets *\$IO* to a particular FreeM I/O channel, allowing *READS* from and *WRITES* to the associated terminal, sequential file, or network socket. Also sets various device parameters.

Syntax (Terminal)

```
USE:postcondition io-channel[:(right-margin:input-field-length:device-
status-word:position:line-terminator:break-key)]
```

For terminals, *io-channel* must be 0.

Semantic and functional description of each device parameter TBA.

Syntax (Sequential Files)

```
USE:postcondition io-channel[:seek-position:terminator:nodelay)]
```

For sequential files, *io-channel* must be in the range 1-99.

Semantic and functional description of each device parameter TBA.

Syntax (Network Sockets)

```
USE:postcondition io-channel
```

The above syntax will set *\$IO* to *io-channel*, directing successive *READS* and *WRITES* to *io-channel*, provided the optional *postcondition* is *true* or omitted.

```
USE:postcondition io-channel:/CONNECT
```

The above syntax will set *\$IO* to *io-channel*, as in the prior example, but will also attempt to connect to the host and port specified for *io-channel* when it was *OPENed*. The */CONNECT* control mnemonic is only valid for socket channels whose connection type is *TCP*. Using */CONNECT* on a *UDP* socket channel will throw *SCKAERR* (error code 55).

For network sockets, *io-channel* must be in the range 100-255.

5.34 VIEW

5.35 WATCH

Sets a watchpoint on a global, local, or SSV node.

Syntax

In its *argumentless* form, WATCH toggles watchpoints on and off, provided the optional *postcondition* is *true* or omitted.

`WATCH:postcondition`

In its *inclusive* form, WATCH adds, removes, or examines watchpoints, provided the optional *postcondition* is *true* or omitted.

A + adds a new watchpoint to the following variable.

A - removes an existing watchpoint for the following variable.

A ? examines the status of a watchpoint for the following variable.

`WATCH:postcondition [+|-|?] var1... [+|-|?] varN`

The following example demonstrates turning watchpoint processing on and adding a watchpoint for global variable `^jpw(1)`. It then changes the value of `^jpw(1)`.

```
USER> WATCH
```

```
Watchpoints enabled.
```

```
USER> WATCH +^JPW(1)
```

```
Added '^JPW("1")' to the watchlist.
```

```
USER> SET ^JPW(1)="new value"
```

```
>> WATCHPOINT: ^JPW("1") => 'new value' (changed 1 times)
```

The following example will remove that watchpoint:

```
USER> WATCH -^JPW(1)
```

```
Removed '^JPW("1")' from the watchlist.
```

```
USER> WATCH ?^JPW(1)
```

```
'^JPW("1")' is not being watched.
```

5.36 WITH

Sets a prefix to be applied to all subsequent local variable or constant references.

Syntax

`WITH:postcondition var-prefix`

In the above single-argument form, sets the \$WITH prefix to *var-prefix*, provided that the optional *postcondition* is either *true* or omitted.

The *var-prefix* argument may be a string literal or any valid FreeM expression.

`WITH:postcondition`

In the above argumentless form, clears the `$WITH` prefix, provided the optional *postcondition* is either *true* or omitted. Equivalent to `WITH ""`.

5.37 WRITE

5.38 XECUTE

5.39 ZALLOCATE

5.40 ZBREAK

5.41 ZDEALLOCATE

5.42 ZGO

5.43 ZHALT

5.44 ZINSERT

5.45 ZJOB

5.46 ZLOAD

5.47 ZNEW

5.48 ZPRINT

5.49 ZQUIT

5.50 ZREMOVE

5.51 ZSAVE

5.52 ZTRAP

5.53 ZWRITE

Writes the names and values of M variables to `$IO`.

Syntax

ZWRITE:postcondition

In the argumentless form, writes the names and values of all local variables to \$IO if the optional *postcondition* is *true* or omitted.

ZWRITE:postcondition ArrayName,...

In the inclusive form, writes the names and values of all local, global, or structured system variables specified in the list of *ArrayNames* to \$IO if the optional *postcondition* is *true* or omitted.

ZWRITE:postcondition (ArrayName,...)

In the exclusive form, writes all local variables *except* those specified in the list of *ArrayNames* to \$IO if the optional *postcondition* is *true* or omitted.

6 Structured System Variables

SSV subscripts are each described in the following format:

`<ssvn-subscript-name> +/-R +/-U +/-D`

The R, U, and D flags represent Read, Update, and Delete. A minus sign indicates that the given operation is *not* allowed, and a plus sign indicates that the given operation *is* allowed.

6.1 ^\$CHARACTER

The ^\$CHARACTER SSV is not yet implemented.

6.2 ^\$DEVICE

FreeM implements several important pieces of functionality in the ^\$DEVICE SSV.

The first subscript of ^\$DEVICE represents the I/O channel of an OPENed device.

The following values for the second subscript are supported:

EOF +R -U -D

Returns 1 if the I/O channel has encountered an end-of-file condition; 0 otherwise. Only valid if the I/O channel is connected to a sequential file.

LENGTH +R -U -D

Returns the length of the file connected to the I/O channel. Only valid if the I/O channel is connected to a sequential file.

MNEMONICSPACE +R -U -D

Returns the current *mnemonic-space* in use for the referenced I/O channel. Always X364 for terminals and blank for sequential files.

DSW +R +U -D

Sets or returns the current *Device Status Word* controlling terminal characteristics. Only valid for I/O channel 0.

TERMINATOR +R +U -D

Sets or returns the **READ** terminator for the specified I/O channel. Must be either `$(13,10)` or `$(10)`. Currently only supported for socket devices (those having an I/O channel of 100-255).

TERMID +R -U -D

Returns the type of terminal connected to channel 0. Only valid for I/O channel 0.

ECHOON +R +U -D

Enables or disables local echo of characters typed in a **READ** command. Only valid for I/O channel 0. Corresponds to bit 0 of the Device Status Word.

DELMODE +R +U -D

Enables or disables visual backspace during a **READ** command. Only valid for I/O channel 0. Corresponds to bit 2 of the Device Status Word.

ESCSEQPROC +R +U -D

Enables or disables escape sequence processing during a **READ** command. Only valid for I/O channel 0. Corresponds to bit 6 of the Device Status Word.

CONVUPPER +R +U -D

Enables or disables automatic conversion to uppercase of alphabetical characters during a **READ** command. Only valid for I/O channel 0. Corresponds to bit 14 of the Device Status Word.

DELEMPY +R +U -D

Enables or disables the automatic deletion of empty strings supplied to a **READ** command. Only valid for I/O channel 0. Corresponds to bit 19 of the Device Status Word.

NOCTRLS +R +U -D

TBD. Only valid for I/O channel 0. Corresponds to bit 20 of the Device Status Word.

CTRLOPROC +R +U -D

Enables or disables *Ctrl-O* processing during **READ** commands. Only valid for I/O channel 0. Corresponds to bit 21 of the Device Status Word.

NOTYPEAHEAD +R +U -D

Enables or disables typeahead buffering during **READ** commands. Only valid for I/O channel 0. Corresponds to bit 25 of the Device Status Word.

Example

The following example M code opens `/etc/freem.conf` and reads its contents line-by-line until the end of the file is reached.

```
SET ^$JOB($JOB,"IOPATH")="/etc" ; set I/O path to /etc
OPEN 1:"freem.conf/r" ; open freem.conf for reading
;
; read until we run out of lines
;
FOR USE 1 READ LINE USE 0 QUIT:~$DEVICE(1,"EOF") D
. WRITE LINE,!
;
CLOSE 1
QUIT
```

6.3 ^\$DISPLAY

The `^$DISPLAY` SSV is not yet implemented.

6.4 ^\$EVENT

The `^$EVENT` SSV is not yet implemented.

6.5 ^\$GLOBAL

The `^$GLOBAL` SSV is not yet implemented.

6.6 `^$JOB`

FreeM fully implements `^$JOB` per ANSI X11.1-1995, as well as several extensions proposed in the M Millennium Draft Standard.

The first subscript of `^$JOB` represents the `$JOB` of the process.

If you KILL a first-level subscript of `^$JOB`, the `SIGTERM` signal will be sent to the corresponding UNIX process, causing pending transactions to be rolled back and the process to be terminated. If the targeted process is in direct mode, the user will be prompted with options of either rolling back or committing any pending transactions.

The following subscripts are supported:

CHARACTER +R -U -D

Returns the character set of the job.

CWD +R +U -D

Returns or sets the current working directory of the job.

EVENT +R +U +D

The subtree contained under `^$JOB($J, "EVENT")` defines asynchronous event handlers for the current job. Please see *Asynchronous Event Handling* for more information.

GLOBAL +R -U -D

Returns the global environment of the job.

IOPATH +R +U -D

Returns or sets the *I/O path* to be used by the `OPEN` command.

PRIORITY +R +U -D

Returns or sets the *nice* value of the FreeM job.

REVSTR +R +U -D

When set to 1, allows `$EXTRACT` to accept negative values.

ROUTINE +R -U -D

Returns the name of the routine currently being executed by the job.

SYMTAB +R +U -D

Returns or sets the current local variable symbol table in use.

FreeM supports two unique and independent symbol tables, allowing FreeM programs to maintain two independent sets of identically- or differently-named local variables per process.

The default symbol table is 0, and the alternate symbol table is 1, corresponding to the valid values for `^$JOB($JOB, "SYMTAB")`.

Setting this subscript to values other than 0 or 1 will result in a `ZINVEXPR` error.

\$PRINCIPAL +R -U -D

Returns the value of `$PRINCIPAL` for the job.

\$TLEVEL +R -U -D

Returns the current transaction level (value of `$TLEVEL` for the job).

\$IO +R -U -D

Returns the current value of **\$IO** for the job.

USER +R -U -D

Returns the UID of the user owning the job.

GROUP +R -U -D

Returns the GID of the group owning the job.

NAMESPACE +R +U -D

Returns or sets the name of the job's currently-active namespace.

MATH +R +U -D

Returns or sets the mode in which decimal comparisons and arithmetic calculations are conducted. Valid values are **FIXED**, for fixed-point decimals having up to 20,000 digits of precision, as determined by the **\$ZPRECISION** intrinsic special variable, and **IEEE754**, to use IEEE 754 floating-point decimals. When in **IEEE754** mode, floating-point numbers maintain 18 digits of numeric precision. **IEEE754** mode will make mathematical calculations significantly faster, especially when accelerated by a floating-point processor, at the expense of precision and accuracy.

FIXED mode is recommended for financial calculations, or where precision and accuracy are valued over performance. **FIXED** is the default mode of FreeM operation.

Attempting to **SET** this node to values other than **FIXED** or **IEEE754** will set **\$ECODE** to M29.

6.7 ^\$LOCK

The ^\$LOCK SSV is not yet implemented.

6.8 ^\$PDISPLAY

The ^\$PDISPLAY SSV is not yet implemented.

6.9 ^\$ROUTINE

The ^\$ROUTINE SSV is not yet implemented.

6.10 ^\$SYSTEM

The ^\$SYSTEM SSV is not yet implemented.

6.11 ^\$WINDOW

The ^\$WINDOW SSV is not yet implemented.

6.12 ^\$ZPROCESS

Provides access to **procfs**, which is a filesystem-like abstraction for UNIX process metadata contained in **/proc**, as well as features for examining and controlling the state of processes external to the FreeM interpreter.

The first subscript always represents the *process ID* of the external process being acted upon.

The following values for the second subscript are supported:

EXISTS +R -U -D

Returns 1 if the referenced process exists; 0 otherwise.

ATTRIBUTES +R -U -D

Exposes the `/proc` files as descendant subscripts, i.e., `WRITE ^$ZPROCESS(2900,"ATTRIBUTES","cmdline"),!` would print the initial command line used to invoke process ID 2900.

SIGNAL -R +U -D

Allows signals to be sent to the referenced process. The following subscript is an integer value corresponding to the desired signal number. You may obtain a list of signal numbers on most UNIX systems with the command `kill -l`.

7 Operators

7.1 Unary +

7.2 Unary -

7.3 + (Add)

7.4 += (Add/Assign)

7.5 ++ (Postfix Increment)

7.6 - (Subtract)

7.7 -= (Subtract/Assign)

7.8 -- (Postfix Decrement)

7.9 * (Multiply)

7.10 *= (Multiply/Assign)

7.11 / (Divide)

7.12 /= (Divide/Assign)

7.13 \ (Integer Divide)

7.14 \= (Integer Divide/Assign)

7.15 # (Modulo)

7.16 #= (Modulo/Assign)

7.17 ** (Exponentiate)

7.18 **= (Exponentiate/Assign)

7.19 < (Less Than)

7.20 <= (Less Than or Equal To)

7.21 > (Greater Than)

7.22 >= (Greater Than or Equal To)

7.23 _ (Concatenate)

7.24 _= (Concatenate/Assign)

7.25 = (Equals)

7.26 [(Contains)

7.27] (Follows)

7.28]] (Sorts After)

7.29 ? (Pattern Match)

7.30 & (Logical AND)

7.31 ! (Logical OR)

7.32 ' (Logical NOT)

7.33 @ (Indirect)

8 Sequential I/O

9 Network I/O

Network I/O in FreeM is supplied through I/O channels 100-255. The normal `READ` and `WRITE` syntax will work with network sockets, with a few exceptions.

9.1 Opening and Connecting a Client Socket

To open a client socket and connect to it, you will need to call the `OPEN` command and the `USE` command:

```
;  
; Set socket read terminator to LF  
;  
SET ^$DEVICE(100,"TERMINATOR")=$C(10)  
;  
; Open an IPv4 TCP socket to mail.mydomain.com on port 25 (SMTP)  
; and connect to it  
;  
OPEN 100:"mail.mydomain.com:25:IPV4:TCP"  
USE 100:/CONNECT  
;  
; Read a line of input from the remote host and write it to the terminal█  
;  
NEW LINE  
READ LINE  
USE 0  
WRITE LINE,!  
;  
; CLOSE the socket and disconnect  
;  
CLOSE 100  
QUIT
```

10 Asynchronous Event Handling

Asynchronous event handling in FreeM follows the specifications of the unpublished MDC *Millennium Draft Standard*.

10.1 Setting Up Async Event Handlers

Asynchronous handlers are configured through the `^$JOB SSV`. In order to become proficient in writing asynchronous event handling code, you need to be aware of several important concepts:

Event Classes

Event classes denote particular categories of events. These include `COMM`, `HALT`, `IPC`, `INTERRUPT`, `POWER`, `TIMER`, and `USER` event classes. At present, only `INTERRUPT` events are supported.

Event Identifiers

Event identifiers denote the precise nature of the event that has occurred. For instance, resizing the terminal window in which a FreeM job is running will send an event of class `INTERRUPT` with an event identifier of `SIGWINCH` (short for *SIG*nal *WIN*dow *CH*ange).

Event Handlers

Event handlers are M routines or subroutines that can be registered to run when an event of a certain event class occurs.

Event Registration

Event registration is the process of modifying the `^$JOB SSV` to associate a particular event class and event identifier with an event handler routine or subroutine.

Event Block

Event blocking is the means by which asynchronous event handling can be temporarily suspended. For example, asynchronous events are temporarily and implicitly blocked for the duration of event handler execution, unless explicitly un-blocked within the event handler. Event handling can also be blocked and unblocked programatically from M code using the `ABLOCK` and `AUNBLOCK` commands.

The following sections of this chapter will take you step-by-step through setting up an event handler for `SIGWINCH` signal handling.

10.2 Registering an Asynchronous Event Handler

To register an event handler, use the following syntax:

```
SET ^$JOB($JOB,"EVENT",event-class,event-identifier)=entryref
```

For example, use the following to register `^RESIZE` as an asynchronous event handler for `SIGWINCH` events:

```
SET ^$JOB($JOB,"EVENT","INTERRUPT","SIGWINCH")="^RESIZE"
```

This by itself will not enable asynchronous event handling, as it merely *registers* an event handler, associating it with event class `INTERRUPT` and event identifier `SIGWINCH`.

10.3 Enabling Asynchronous Event Handling

In order to enable asynchronous event handling, the `ASTART` command is used. In the following example, we will enable asynchronous event handling for the `INTERRUPT` event class:

```
ASTART "INTERRUPT"
```

Omitting the `"INTERRUPT"` argument will enable asynchronous event handling for *all* event classes. See `ASTART` in the commands section for more details.

Once this is done, any event handlers registered for the `INTERRUPT` event class in `^$JOB` will be executed asynchronously as appropriate.

10.4 Disabling Asynchronous Event Handling

To disable asynchronous event handling, the `ASTOP` command is used. In the following example, we will disable asynchronous event handling for the `INTERRUPT` event class:

```
ASTOP "INTERRUPT"
```

Omitting the `"INTERRUPT"` argument will disable asynchronous event handling for *all* event classes. See `ASTOP` in the commands section for more details.

You may also disable asynchronous event handling for a specific event identifier by `KILL`ing the appropriate node in the `^$JOB SSV`, which unregisters the event handler altogether. The following example will unregister the event handler for the `SIGWINCH` event identifier:

```
KILL ^$JOB($JOB, "EVENT", "INTERRUPT", "SIGWINCH")
```

10.5 Temporarily Blocking Asynchronous Event Handling

To temporarily block processing of specific event classes, you will use the `ABLOCK` command. `ABLOCK` functions incrementally, that is, each successive call to `ABLOCK` will increment a counter of blocks held for the specified event class or classes, and each successive call to `AUNBLOCK` will decrement that counter. Event handling for the specified event classes will be blocked as long as the `ABLOCK` counter for those classes is greater than zero. Thus, event blocking is cumulative, in a manner similar to M incremental locks.

The following example blocks asynchronous event handling for the `INTERRUPT` event class:

```
ABLOCK "INTERRUPT"
```

Note that entering an event handler causes an implicit `ABLOCK` of *all* event classes, to prevent event handlers from interrupting other event handlers during their execution. This may be overridden by calling `AUNBLOCK` for one or more event classes within an event handler. However, unblocking event handling during an event handler should be done with great caution, as this can make the flow of code execution somewhat unpredictable, especially if M globals are modified inside of an event handler routine or subroutine.

Modifying M globals within event handlers is allowed but strongly discouraged, as doing so can lead to logical corruption of the database. If you must modify an M global within an event handler, guard all such operations with prodigious and careful use of `LOCKS`, ensuring that such modifications occur in the desired logical order.

11 Synchronous Event Handling

12 GUI Programming with MWAPI

13 User-Defined Z Commands

14 User-Defined Z Functions

15 User-Defined SSVs

16 System Library Routines

16.1 `^%ZCOLUMNS`

This routine is the implementation of the `$ZCOLUMNS` intrinsic special variable.

16.2 `^%ZFREEM`

This routine is the default startup routine for FreeM running in direct mode.

Running `DO INFO` from direct mode will use this routine to display information about the current FreeM status and namespace configuration.

16.3 `^%ZHELP`

This routine implements the online help feature of FreeM, invoked by typing `?` in direct mode. It simply asks the underlying system to execute the command `info freem`.

16.4 `^%ZROWS`

This routine is the implementation of the `$ZROWS` intrinsic special variable.

17 Error Processing

FreeM exposes three means of processing M program execution errors:

FreeM-style error processing

FreeM-style error processing exposes a read/write error trap in `$ZTRAP`. The contents of `$ZTRAP` must be either empty or a valid M entryref, to which FreeM will `GOTO` if an error occurs. Each program stack execution level can have its own `$ZTRAP` error handler enabled.

DSM 2.0-style error processing

DSM 2.0-style error processing emulates the `$ZTRAP` behavior of Digital Standard MUMPS v2. It has the same behavior as FreeM-style error handling, with the exception that in DSM 2.0-style error processing, only one `$ZTRAP` error handler is set across all program stack execution levels.

Standard error processing

Standard error processing uses the `NEW`-able `$ETRAP` variable to store error handler code, which may be any valid M code. The code in `$ETRAP` will run when an error occurs or the `$ECODE` ISV becomes non-empty. Stack information for standard error handling is provided by the `$STACK` ISV, the `$STACK()` intrinsic pseudo-function, and the `NEW`-able `$ESTACK` ISV.

If `$ETRAP` is non-empty when an error condition occurs, `$ZTRAP` is ignored, regardless of whether FreeM-style or DSM 2.0-style error processing is enabled at the time of the error.

For further information on switching between FreeM-style and DSM 2.0-style `$ZTRAP` error handling, see the documentation for the `BREAK` command.

18 Debugging

19 System Configuration

19.1 Installing FreeM

20 Accessing FreeM from C Programs

FreeM provides a library, ‘libfreem.so’, as well as corresponding header file ‘freem.h’, allowing C programmers to write programs that access FreeM globals, locals, structured system variables, subroutines, and extrinsic functions. This functionality can be used to implement language bindings and database drivers for external systems.

In order to be used in your C programs, your C programs must link with ‘libfreem.so’ and include ‘freem.h’. This will allow your C code access to the function prototypes, data structures, and constants required for calling the ‘libfreem.so’ APIs.

You must exercise caution in developing programs that interface with FreeM through ‘libfreem.so’ to ensure that all ‘libfreem.so’ API calls are serialized, as FreeM and the ‘libfreem.so’ library are neither thread-safe nor reentrant.

You must also avoid setting signal handlers for SIGALRM, as FreeM uses SIGALRM to manage timeouts for LOCK, READ, and WRITE.

20.1 freem_ref_t Data Structure

The libfreem API uses a struct of type freem_ref_t in order to communicate state, pass in values, and return results.

The data structure, defined in ‘freem.h’, looks like this:

```
typedef struct freem_ref_t {

    /*
     * The 'reftype' field can be one of:
     *
     * MREF_RT_LOCAL
     * MREF_RT_GLOBAL
     * MREF_RT_SSV
     */
    short reftype;

    /*
     * The 'name' field is the name of the local variable,
     * global variable, or SSV (without ^ or ^$).
     */
    char name[256];

    /*
     * Returned data goes in a string, so you've got to figure out the
     * whole M canonical number thing yourself. Good luck. :-)
     */
    char value[STRLEN];

    short status;

    unsigned int subscript_count;
};
```



```

    char subscripts[255][256];

} freem_ref_t;
freem_ref_t Members

```

‘reftype’ The ‘reftype’ member determines whether we are operating on a local variable, a global variable, or a structured system variable. It may be set to any of following constants: MREF_RT_LOCAL, MREF_RT_GLOBAL, or MREF_RT_SSV.

‘name’ The ‘name’ member contains the name of the global, local, or SSV to be accessed. You *must not* include leading characters, such as ^ or ^\$.

‘value’ This member contains the value read from or the value to be written to the global, local, or SSV.

‘status’ This member gives us various API status values after the API call returns. In general, this value is also returned by each API function.

‘subscript_count’
The number of subscripts to be passed into the API function being called. This value represents the maximum index into the first dimension of the `subscripts` array.

‘subscripts’
A two-dimensional array containing the subscripts to which we are referring in this API call.

20.2 freem_ent_t Data Structure

The `freem_function()` and `freem_procedure()` APIs in `libfreem` use the `freem_ent_t` struct in order to indicate the name of the entry point being called, any arguments being passed to it, and the return value of the called function (not used for `freem_procedure()`).

The data structure, defined in ‘`freem.h`’, looks like this:

```

typedef struct freem_ent_t {

    /* name of function or procedure entry point */
    char name[256];

    /* return value */
    char value[STRLEN];

    /* value of ierr on return */
    short status;

    /* argument count and array */
    unsigned int argument_count;
    char arguments[255][256];

} freem_ent_t;

```

freem_ent_t Members

<code>'name'</code>	The <code>'name'</code> member contains the name of the extrinsic function or procedure to be called.
<code>'value'</code>	This member contains the value returned by the function called. Not used by <code>freem_procedure()</code> .
<code>'status'</code>	This member gives us the value of <code>ierr</code> after the function or procedure call returns. The possible values of <code>ierr</code> are listed in <code>merr.h</code> .
<code>'argument_count'</code>	The number of arguments to be passed into the extrinsic function or procedure being called. This value represents the maximum index into the first dimension of the <code>arguments</code> array.
<code>'arguments'</code>	A two-dimensional array containing the arguments to be passed into the extrinsic function or procedure being called.

20.3 freem_init()

Initializes `libfreem` in preparation for calling other APIs.

Synopsis

```
pid_t freem_init(char *namespace_name);
```

Parameters

`namespace_name`

Specifies the namespace to use.

Return Values

Returns the process ID of the `libfreem` process on success, or `-1` on failure.

Example

This example prompts the user to enter a FreeM namespace and then attempts to initialize `libfreem` to use the selected namespace.

```
#include <stdio.h>
#include <string.h>
#include <freem.h>

int main(int argc, char **argv, char **envp)
{
    char namespace[256];

    /* get the namespace name to use */
    printf("Enter FreeM namespace to use: ");
    fgets(namespace, 255, stdin);

    /* remove the trailing newline */
    namespace[strcspn(buffer, "\n")] = '\0';

    /* initialize libfreem using the provided namespace */
```

```

    if(freem_init(namespace) == TRUE) {
        printf("\nSuccess\n");
    }
    else {
        printf("\nFailure\n");
    }

    return 0;
}

```

20.4 freem_version()

Returns the version of FreeM in use.

Synopsis

```
short freem_version(char *result);
```

Parameters

result The **result** parameter is a pointer to a buffer in which the FreeM version information will be returned. The caller must allocate memory for this buffer prior to calling this API. It should be at least 20 bytes in length.

Return Value

Returns 0.

Example

This example will display the FreeM version on standard output.

```

#include <stdio.h>
#include <string.h>
#include <freem.h>

int main(int argc, char **argv, char **envp)
{
    char version[20] = {0};

    freem_init('USER');
    freem_version(version);

    printf('FreeM version: %s\n', version);
}

```

20.5 freem_set()

Sets a FreeM local node, global node, or writable SSV node.

Synopsis

```
short freem_set(freem_ref_t *ref);
```

Parameters

`freem_ref_t`

This parameter is a pointer to a `freem_ref_t` struct. The caller must allocate the memory for this struct.

Return Value

Returns OK on success, or one of the other error values defined in `merr.h`.

Example

This example sets the value `blue` into global node `^car("color")`.

```
#include <stdio.h>
#include <string.h>
#include <freem.h>

int main(int argc, char **argv, char **envp)
{
    freem_ref_t ref;

    /* we're setting a global */
    ref.reftype = MREF_RT_GLOBAL;

    /* access global "car" */
    strcpy(ref.name, "car");

    /* set up the subscripts */
    ref.subscript_count = 1;
    strcpy(ref.subscripts[0], "color");

    /* use the USER namespace */
    freem_init("USER");

    /* write the data out */
    freem_set(&ref);
}
```

20.6 `freem_get()`

Retrieves a FreeM local node, global node, or writable SSV node.

Synopsis

```
short freem_get(freem_ref_t *ref);
```

Parameters

`freem_ref_t`

This parameter is a pointer to a `freem_ref_t` struct. The caller must allocate the memory for this struct.

Return Value

Returns OK on success, or one of the other error values defined in `merr.h`.

Example

This example retrieves the character set of the current process.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <freem.h>

int main(int argc, char **argv, char)
{
    pid_t pid;
    freem_ref_t ref;

    /* get the PID of this process */
    pid = getpid();

    /* we want to access an SSV */
    ref.reftype = MREF_RT_SSV;

    /* set up the name and subscripts */
    strcpy(ref.name, "JOB");

    ref.subscript_count = 2;
    sprintf(ref.subscripts[0], "%d", pid);
    strcpy(ref.subscripts[1], "CHARACTER");

    /* initialize libfreem, using the USER namespace */
    freem_init("USER");

    /* call libfreem API */
    freem_get(&ref);

    /* output the character set info */
    printf("PID %d character set is '%s'\n", pid, ref.value);
}
```

20.7 `freem_kill()`

Deletes a FreeM local node, global node, or killable SSV node, as well as all of its children.

```
short freem_kill(freem_ref_t *ref);
```

Parameters

`freem_ref_t`

This parameter is a pointer to a `freem_ref_t` struct. The caller must allocate the memory for this struct.

Return Value

Returns OK on success, or one of the other error values defined in `merr.h`.

Example

```
#include <stdio.h>
#include <string.h>
#include <freem.h>

int main(int argc, char **argv, char **envp)
{
    freem_ref_t ref;

    /* we're killing a global node */
    ref.reftype = MREF_RT_GLOBAL;

    /* access global "car" */
    strcpy(ref.name, "car");

    /* set up the subscripts */
    ref.subscript_count = 0;

    /* use the USER namespace */
    freem_init("USER");

    /* kill the global and all its descendant subscripts */
    freem_kill(&ref);
}
```

20.8 freem_data()**20.9 freem_order()****20.10 freem_query()****20.11 freem_lock()****20.12 freem_unlock()****20.13 freem_tstart()****20.14 freem_trestart()****20.15 freem_trollback()**

20.16 `freem_tlevel()`

20.17 `freem_tcommit()`

20.18 `freem_function()`

20.19 `freem_procedure()`

Appendix A FreeM Administrator

The `fmadm` utility is the preferred method of managing a FreeM installation, and will eventually replace all of the existing utilities. Unlike the existing, legacy utilities, `fmadm` presents a consistent, simple interface for all FreeM management tasks, and is namespace-aware. This appendix will document each `fmadm` facility as it is implemented, until all of the legacy utilities have been replaced.

The `fmadm` utility's functions all follow the below, consistent syntax:

```
usage:  fmadm <action> <object> <namespace> [OPTIONS]
```

The *action* keyword can be one of the following:

<i>list</i>	Lists instances of <i>object</i>
<i>examine</i>	Examines a single instance of <i>object</i>
<i>verify</i>	Verifies the integrity of <i>object</i>
<i>compact</i>	Compacts <i>object</i>
<i>repair</i>	Repairs integrity problems in <i>object</i>
<i>create</i>	Creates an instance of <i>object</i>
<i>remove</i>	Removes an instance of <i>object</i>
<i>import</i>	Imports an <i>object</i>
<i>export</i>	Exports an <i>object</i>
<i>backup</i>	Creates a backup of <i>object</i>
<i>restore</i>	Restores a backup of <i>object</i>
<i>migrate</i>	Migrates an instance of <i>object</i> from an older FreeM version to the current version
<i>edit</i>	Edits an instance of <i>object</i>

The *object* keyword can be one of the following:

<i>lock</i>	The FreeM LOCK table. Supported actions are <code>list</code> and <code>remove</code> .
<i>zallocate</i>	The FreeM ZALLOCATE table. No actions yet implemented.
<i>journal</i>	FreeM after-image journaling. Supported actions are <code>examine</code> and <code>restore</code> . The <code>examine</code> action will dump the after-image journal entries for the selected namespace in human-readable format. The <code>restore</code> action will play after-image journals forward for the selected namespace.
<i>namespace</i>	FreeM namespaces (collections of M routines and globals). No actions yet implemented.

- global* The database files representing each FreeM *global*.
Supported actions are `list`, `examine`, `remove`, and `verify`.
- routine* An M routine, stored as a `.m` file.
Supported actions are `list`, `examine`, `remove`, `import`, `export`, `backup`, and `edit`.
- job* A UNIX process representing an instance of the FreeM runtime.
Supported actions are `list` and `examine`.

Appendix B FreeM Legacy Utilities

B.1 Global Compactor (gcompact)

Compacts the specified global in place.

Syntax

```
gcompact /path/to/global/file
```

B.2 Block Examiner (gfix)

The *gfix* interactive utility program permits navigation of the B-tree structure of the specified global a block at a time.

Syntax

```
gfix </path/to/global/file>
```

B.3 Global Repair Tool (grestore)

This utility will fix problems with the specified global.

Syntax

```
grestore </path/to/global/file>
```

Appendix C FreeM VIEW Commands and Functions

C.1 VIEW 16: Total Count of Error Messages/View Single Error Message

Unknown semantics

C.2 VIEW 17: Intrinsic Z-Commands

Allows the user to retrieve or specify the list of intrinsic Z-commands that FreeM will attempt to run internally, allowing intrinsic Z-commands implemented internally to be replaced with M equivalents implemented as %-routines in the SYSTEM namespace.

C.3 VIEW 18: Intrinsic Z-Functions

Allows the user to retrieve or specify the list of intrinsic Z-functions that FreeM will attempt to run internally, allowing intrinsic Z-functions implemented internally to be replaced with M equivalents implemented as %-routines in the SYSTEM namespace.

C.4 VIEW 19: Intrinsic Special Variables

Allows the user to retrieve or specify which special variables are implemented internally.

C.5 VIEW 20: Break Service Code

Allows the user to view or specify the code that will be run when a BREAK is encountered.

C.6 VIEW 21: View Size of Last Global

Allows the user to view the size of the last referenced global.

C.7 VIEW 22: Count VIEW 22 Aliases

Retrieves the number of VIEW 22 aliases in effect.

C.8 VIEW 23: View Contents of Input Buffer

Retrieves the contents of the I/O input buffer.

C.9 VIEW 24: Maximum Number of Screen Rows

Retrieves the maximum number of screen rows supported in the current FreeM build.

C.10 VIEW 25: Maximum Number of Screen Columns

Retrieves the maximum number of screen columns supported in the current FreeM build.

C.11 VIEW 26: DO/FOR/XECUTE Stack Pointer

Retrieves the DO, FOR, and XECUTE stack pointer.

C.12 VIEW 27: DO/FOR/XECUTE Stack Pointer (On Error)

Retrieves the DO, FOR, and XECUTE stack pointer (on error).

C.13 VIEW 29: Copy Symbol Table

Copies the symbol table? We aren't currently aware of what this means.

C.14 VIEW 30: Inspect Arguments

Retrieves the arguments passed to the `freem` executable.

C.15 VIEW 31: Count Environment Variables

Allows the user to inspect the number of variables in the process environment table.

Syntax

```
WRITE $VIEW(31),!
```

Appendix D Implementation Limits

Appendix E US-ASCII Character Set

Code	Character
000	<NUL>
001	<SOH>
002	<STX>
003	<ETX>
004	<EOT>
005	<ENQ>
006	<ACK>
007	<BEL>
008	<BS>
009	<HT>
010	<LF>
011	<VT>
012	<FF>
013	<CR>
014	<SO>
015	<SI>
016	<DLE>
017	<DC1>
018	<DC2>
019	<DC3>
020	<DC4>
021	<NAK>
022	<SYN>
023	<ETB>
024	<CAN>
025	
026	<SUB>
027	<ESC>
028	<FS>
029	<GS>
030	<RS>
031	<US>
032	<space>
033	!
034	"
035	#

working on a Windows machine, you must take care to follow this, as Windows will use a carriage return followed by a linefeed by default.

This project follows a modified version of what is known as the Stroustrup indentation style.

F.4 Brace Placement (Functions)

We use modern, ANSI-style function prototypes, with the type specifier on the same line as the function name. You may encounter other styles in the code, but we are transitioning to the new style as time permits.

Below is a correct example:

```
int main(int argc, char **argv, char **envp)
{

}
```

F.5 Brace Placement (if-for-while-do)

The `if` keyword should be followed by one space, then the opening paren and conditional expression. We also use Stroustrup-style `else` blocks, rather than the K&R 'cuddled' `else`:

```
if (x) {
...
}
else {
...
}

while (1) {
...
}

for (i = 1; i < 10; i++) {
...
}

do {
...
} while (x);
```

Single-statement if blocks should be isolated to a single line:

```
if (x) stmt();
not:
if (x)
    stmt ();
```

Notice that there is a space between `if` and `(x)`, and also between `stmt` and `()`. This should be followed throughout the code.

If an `if` block has an `else if` or `else`, all parts of the construct must be bracketed, even if one or more of them contain only one statement:


```
if (x) {
    foo();
}
else if (y) {
    bar();
}
else {
    bas();
}
```

F.6 Labels and goto

Labels must begin in column 1, and have two lines of vertical space above and one beneath.

F.7 Preprocessor Conditionals

F.8 coding standards, preprocessor conditionals

I have struggled with this, but have settled upon the standard practice of keeping them in column 1.

F.9 Overall Program Spacing

- Variable declarations fall immediately beneath the opening curly brace, and should initialize the variable right there whenever initialization is used.
- One line between the last variable declaration and the first line of real code.
- The `return` statement of a function (when used as the last line of a function) should have one blank line above it and none below it.
- Really long functions (those whose entire body is longer than 24 lines) should have a comment immediately following the closing curly brace of the function, telling you what function the closing brace terminates.

F.10 The `switch()` Statement

We indent `case` one level beneath `switch()`, and the code within each `case` beneath the `case`. Each `case` should have one line of vertical whitespace above it:

```
switch(foo) {

    case some_const:
        foo();

        break;

    case some_other_const:
        bar();

        break;
```

```
    default:  
        exit(1);  
  
        break;  
}
```

F.11 Comments

We use C-style comments (`/* comment */`) exclusively, even on single-line comments. C++ comments (`// comment`) are not permitted.

Index

!

!	17
!!	17

\$

\$ASCII	12
\$CHAR	12
\$DATA	12
\$DEVICE	8
\$ECODE	8
\$ESTACK	8
\$ETRAP	8
\$EXTRACT	12
\$FIND	12
\$FNUMBER	13
\$GET	13
\$HOROLOG	8
\$IO	8
\$JOB	8
\$JUSTIFY	13
\$KEY	8
\$LENGTH	13
\$NAME	13
\$NEXT	13
\$ORDER	13
\$PIECE	13
\$PRINCIPAL	8
\$QLength	14
\$QSUBSCRIPT	14
\$QUERY	14
\$QUIT	9
\$RANDOM	14
\$REFERENCE	8
\$REVERSE	14
\$SELECT	14
\$STACK	9, 14
\$STORAGE	9
\$SYSTEM	9
\$TEST	9
\$TEXT	14
\$TLEVEL	9
\$TRANSLATE	15
\$TRESTART	9
\$VIEW	15
\$WITH	9
\$X	9
\$Y	10
\$ZA	10
\$ZB	10
\$ZBOOLEAN	15
\$ZCALL	15
\$ZCONTROL	10
\$ZCR	15

\$ZCRC	15
\$ZDATE	10, 15
\$ZEDIT	16
\$ZERROR	10
\$ZHOROLOG	10, 16
\$ZHT	16
\$ZINRPT	10
\$ZJOB	10
\$ZKEY	16
\$ZLENGTH	16
\$ZLOCAL	10
\$ZLSD	16
\$ZM	16
\$ZNAME	16
\$ZNEXT	16
\$ZORDER	16
\$ZPIECE	16
\$ZPRECISION	10
\$ZPREVIOUS	16
\$ZREFERENCE	10
\$ZREPLACE	16
\$ZSORT	16
\$ZSYNTAX	16
\$ZSYSTEM	11
\$ZTIME	11, 16
\$ZTRAP	11
\$ZVERSION	11

~

~\$CHARACTER	27
~\$DEVICE	27
~\$DISPLAY	28
~\$EVENT	28
~\$GLOBAL	28
~\$JOB	29
~\$LOCK	30
~\$PDISPLAY	30
~\$ROUTINE	30
~\$SYSTEM	30
~\$WINDOW	30
~\$ZPROCESS	30
~%ZCOLUMNS	43
~%ZFREEM	43
~%ZHELP	43
~%ZROWS	43

A

ABLOCK	17
ASSERT	18
ASTART	18
ASTOP	18
AUNBLOCK	19

B

BREAK 19

C

CLOSE 19

coding standards, brace placement, functions... 63

coding standards, brace placement, if-for-while-do
..... 63

coding standards, comments 65

coding standards, goto 64

coding standards, indentation 62

coding standards, labels 64

coding standards, layout 62

coding standards, module headers 62

coding standards, spacing of programs 64

coding standards, switch() 64

coding standards, variable naming 62

command line interface 5

commands 17

commands, ! 17

commands, !! 17

commands, ABLOCK 17

commands, ASSERT 18

commands, ASTART 18

commands, ASTOP 18

commands, AUNBLOCK 19

commands, BREAK 19

commands, CLOSE 19

commands, CONST 20

commands, debugging 18, 24, 25

commands, DO 20

commands, ELSE 20

commands, external 17

commands, FOR 20

commands, GOTO 20

commands, HALT 20

commands, HANG 20

commands, IF 20

commands, implementation-specific 18, 24, 25

commands, JOB 20

commands, KILL 20

commands, KSUBSCRIPTS 20

commands, KVALUE 21

commands, LOCK 21

commands, MERGE 21

commands, NEW 21

commands, OPEN 22

commands, QUIT 23

commands, READ 23

commands, SET 23

commands, TCOMMIT 23

commands, TRESTART 23

commands, TROLLBACK 23

commands, TSTART 23

commands, unimplemented 23

commands, USE 23

commands, VIEW 24

commands, WATCH 24

commands, WITH 24

commands, WRITE 25

commands, XECUTE 25

commands, ZALLOCATE 25

commands, ZBREAK 25

commands, ZDEALLOCATE 25

commands, ZGO 25

commands, ZHALT 25

commands, ZINSERT 25

commands, ZJOB 25

commands, ZLOAD 25

commands, ZNEW 25

commands, ZPRINT 25

commands, ZQUIT 25

commands, ZREMOVE 25

commands, ZSAVE 25

commands, ZTRAP 25

commands, ZWRITE 25

configuration, system 46

CONST 20

contributors, Best, John 1

contributors, Diamond, Jon 1

contributors, Fox, Ronald L. 1

contributors, Gerum, Winfried 1

contributors, ha-Ashkenaz, Shalom 1

contributors, Kreis, Greg 1

contributors, Landis, Larry 1

contributors, Marshall, Frederick D.S. 1

contributors, Milligan, Lloyd 1

contributors, Morris, Steve 1

contributors, Murray, John 1

contributors, Pastors, Wilhelm 1

contributors, Schell, Kate 1

contributors, Schofield, Lyle 1

contributors, Stefanik, Jim 1

contributors, Trocha, Axel 1

contributors, Walters, Dick 1

contributors, Whitten, David 1

contributors, Wicksell, David 1

contributors, Willis, John P. 1

contributors, Zeck, Steve 1

D

debugging 45

direct mode 5

DO 20

E

ELSE 20

error processing 44

event handlers, blocking 37

event handlers, disabling 37

event handlers, enabling 37

event handlers, registration 36

event handling, asynchronous 36

- execution, interactive 5
- F**
- fmadm 55
- FOR 20
- G**
- GOTO 20
- H**
- ha-Ashkenaz, Shalom 62
- HALT 20
- HALT, in direct-mode 6
- HANG 20
- I**
- IF 20
- installation, FreeM 46
- intrinsic functions, \$ASCII 12
- intrinsic functions, \$CHAR 12
- intrinsic functions, \$DATA 12
- intrinsic functions, \$EXTRACT 12
- intrinsic functions, \$FIND 12
- intrinsic functions, \$FNUMBER 13
- intrinsic functions, \$GET 13
- intrinsic functions, \$JUSTIFY 13
- intrinsic functions, \$LENGTH 13
- intrinsic functions, \$NAME 13
- intrinsic functions, \$NEXT 13
- intrinsic functions, \$ORDER 13
- intrinsic functions, \$PIECE 13
- intrinsic functions, \$QLength 14
- intrinsic functions, \$QSUBSCRIPT 14
- intrinsic functions, \$QUERY 14
- intrinsic functions, \$RANDOM 14
- intrinsic functions, \$REVERSE 14
- intrinsic functions, \$SELECT 14
- intrinsic functions, \$STACK 14
- intrinsic functions, \$TEXT 14
- intrinsic functions, \$TRANSLATE 15
- intrinsic functions, \$VIEW 15
- intrinsic functions, \$ZBOOLEAN 15
- intrinsic functions, \$ZCALL 15
- intrinsic functions, \$ZCR 15
- intrinsic functions, \$ZCRC 15
- intrinsic functions, \$ZDATE 15
- intrinsic functions, \$ZEDIT 16
- intrinsic functions, \$ZHOROLOG 16
- intrinsic functions, \$ZHT 16
- intrinsic functions, \$ZKEY 16
- intrinsic functions, \$ZLENGTH 16
- intrinsic functions, \$ZLSD 16
- intrinsic functions, \$ZM 16
- intrinsic functions, \$ZNAME 16
- intrinsic functions, \$ZNEXT 16
- intrinsic functions, \$ZORDER 16
- intrinsic functions, \$ZPIECE 16
- intrinsic functions, \$ZPREVIOUS 16
- intrinsic functions, \$ZREPLACE 16
- intrinsic functions, \$ZSORT 16
- intrinsic functions, \$ZSYNTAX 16
- intrinsic functions, \$ZTIME 16
- intrinsic functions, implementation-specific 15, 16
- intrinsic special variables, \$DEVICE 8
- intrinsic special variables, \$ECODE 8
- intrinsic special variables, \$ESTACK 8
- intrinsic special variables, \$ETRAP 8
- intrinsic special variables, \$HOROLOG 8
- intrinsic special variables, \$IO 8
- intrinsic special variables, \$JOB 8
- intrinsic special variables, \$KEY 8
- intrinsic special variables, \$PRINCIPAL 8
- intrinsic special variables, \$QUIT 9
- intrinsic special variables, \$REFERENCE 8
- intrinsic special variables, \$STACK 9
- intrinsic special variables, \$STORAGE 9
- intrinsic special variables, \$SYSTEM 9
- intrinsic special variables, \$TEST 9
- intrinsic special variables, \$TLEVEL 9
- intrinsic special variables, \$TRESTART 9
- intrinsic special variables, \$WITH 9
- intrinsic special variables, \$X 9
- intrinsic special variables, \$Y 10
- intrinsic special variables, \$ZA 10
- intrinsic special variables, \$ZB 10
- intrinsic special variables, \$ZCONTROL 10
- intrinsic special variables, \$ZDATE 10
- intrinsic special variables, \$ZERROR 10
- intrinsic special variables, \$ZHOROLOG 10
- intrinsic special variables, \$ZINRPT 10
- intrinsic special variables, \$ZJOB 10
- intrinsic special variables, \$ZLOCAL 10
- intrinsic special variables, \$ZPRECISION 10
- intrinsic special variables, \$ZREFERENCE 10
- intrinsic special variables, \$ZSYSTEM 11
- intrinsic special variables, \$ZTIME 11
- intrinsic special variables, \$ZTRAP 11
- intrinsic special variables, \$ZVERSION 11
- intrinsic special variables, implementation-specific 10, 11
- intrinsic special variables, unimplemented 9
- invocation, command-line 3
- J**
- JOB 20
- K**
- KILL 20
- KSUBSCRIPTS 20

KVALUE..... 21

L

libfreem, data structures: freem_ent_t 48
 libfreem, data structures: freem_ref_t 47
 libfreem, freem_data() 53
 libfreem, freem_ent_t.argument_count 49
 libfreem, freem_ent_t.arguments 49
 libfreem, freem_ent_t.name 49
 libfreem, freem_ent_t.status 49
 libfreem, freem_ent_t.value 49
 libfreem, freem_function() 54
 libfreem, freem_get() 51
 libfreem, freem_init() 49
 libfreem, freem_kill() 52
 libfreem, freem_lock() 53
 libfreem, freem_order() 53
 libfreem, freem_procedure() 54
 libfreem, freem_query() 53
 libfreem, freem_ref_t.name 48
 libfreem, freem_ref_t.reftype 48
 libfreem, freem_ref_t.status 48
 libfreem, freem_ref_t.subscript_count 48
 libfreem, freem_ref_t.subscripts 48
 libfreem, freem_ref_t.value 48
 libfreem, freem_set() 50
 libfreem, freem_tcommit() 54
 libfreem, freem_tlevel() 54
 libfreem, freem_trestart() 53
 libfreem, freem_trollback() 53
 libfreem, freem_tstart() 53
 libfreem, freem_unlock() 53
 libfreem, freem_version() 50
 limitations, memory 60
 LOCK..... 21

M

maximum size, global 60
 maximum size, routine 60
 maximum size, string 60
 MERGE..... 21
 modes, programmer 5

N

networks, input and output 35
 networks, opening and connecting client sockets
 35
 NEW..... 21

O

OPEN..... 22
 operators, ! 33
 operators, # 32
 operators, #= 32

operators, & 33
 operators, ' 33
 operators, * 32
 operators, ** 32
 operators, **= 32
 operators, *= 32
 operators, + 32
 operators, ++ 32
 operators, += 32
 operators, - 32
 operators, - 32
 operators, -= 32
 operators, / 32
 operators, /= 32
 operators, < 32
 operators, <= 33
 operators, = 33
 operators, > 33
 operators, >= 33
 operators, ? 33
 operators, @ 33
 operators, [..... 33
 operators,] 33
 operators,]] 33
 operators, - 33
 operators, -= 33
 operators, \ 32
 operators, \= 32
 operators, unary + 32
 operators, unary - 32
 options, command-line 3

Q

QUIT 23

R

READ 23
 REPL, direct-mode 7
 routines, as shell scripts 3

S

SET 23
 shebang line 3
 shell scripting 3
 SSVs 27
 structured system variables 27, 42
 structured system variables, ^\$CHARACTER.. 27
 structured system variables, ^\$DEVICE 27
 structured system variables, ^\$DISPLAY 28
 structured system variables, ^\$EVENT 28
 structured system variables, ^\$GLOBAL 28
 structured system variables, ^\$JOB 29
 structured system variables, ^\$LOCK 30
 structured system variables, ^\$PDISPLAY 30
 structured system variables, ^\$ROUTINE 30

- structured system variables, ^\$SYSTEM 30
 - structured system variables, ^\$WINDOW 30
 - structured system variables, ^\$ZPROCESS..... 30
 - structured system variables, user-defined 42
 - system library routines 43
 - system library routines, ^%ZCOLUMNS 43
 - system library routines, ^%ZFREEM 43
 - system library routines, ^%ZHELP 43
 - system library routines, ^%ZROWS 43
- T**
- TCOMMIT 23
 - TRESTART 23
 - TROLLBACK 23
 - TSTART 23
- U**
- USE 23
 - utilities, fmadm 55
 - utilities, gfix 57
 - utilities, legacy 57
 - utilities, legacy, gcompact 57
 - utilities, legacy, grestore 57
 - utilities, system management 55
- V**
- variables, intrinsic special 8
 - variables, structured system 27
 - VIEW 24
 - VIEW commands/functions, 16, total count of error messages/view single error message .. 58
 - VIEW commands/functions, 17, intrinsic Z-commands 58
 - VIEW commands/functions, 18, intrinsic Z-functions 58
 - VIEW commands/functions, 19, intrinsic special variables 58
 - VIEW commands/functions, 20, break service code 58
 - VIEW commands/functions, 21, view size of last global 58
 - VIEW commands/functions, 22, count VIEW 22 aliases 58
 - VIEW commands/functions, 23, input buffer contents 58
 - VIEW commands/functions, 24, maximum number of screen rows 58
 - VIEW commands/functions, 25, maximum number of screen columns 58
 - VIEW commands/functions, 26, DO/FOR/XECUTE stack pointer 58
 - VIEW commands/functions, 27, DO/FOR/XECUTE stack pointer, on error 59
 - VIEW commands/functions, 29, copy symbol table 59
 - VIEW commands/functions, 30, inspect arguments 59
 - VIEW commands/functions, 31, count environment variables 59
- W**
- WATCH 24
 - WITH 24
 - WRITE 25
- X**
- XECUTE 25
- Z**
- z functions, user-defined 41
 - ZALLOCATE 25
 - ZBREAK 25
 - ZDEALLOCATE 25
 - ZGO 25
 - ZHALT 25
 - ZINSERT 25
 - ZJOB 25
 - ZLOAD 25
 - ZNEW 25
 - ZPRINT 25
 - ZQUIT 25
 - ZREMOVE 25
 - ZSAVE 25
 - ZTRAP 25
 - ZWRITE 25