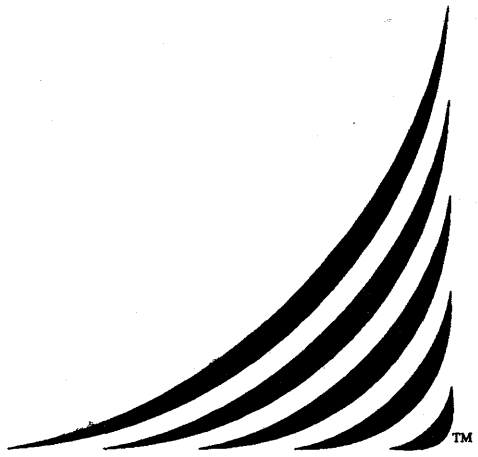
V12

MAINSAIL®

Language Manual, Volume II



MAINSAIL®

Language Manual, Part II:

System Procedures, Macros, and Variables

24 March 1989

Table of Contents

List of Examples

List of Figures

List of Tables

1. System Procedures, Macros, and Variables

System procedures and macros provide services that support the execution of MAINSAIL programs. System variables are accessible from any module; they are interface variables of runtime modules to which every module is guaranteed to have linkage.

Sections 1.1, 1.2, and 1.3 describe the procedure qualifiers "COMPILETIME", "$BUILTIN", and "SPECIAL". These qualifiers apply only to system procedures (the programmer cannot use them) and are described to help the user understand the system procedure declarations.

Section 1.5 gives a summary of the system procedures, macros, and variables. The remaining sections give the procedure, macro, and variable descriptions in alphabetic order, except that the load procedures (named "xLoad", where x is a data type abbreviation) are grouped together under "L". The complete headers of the procedures are specified, providing sufficient information (e.g., about parameters) for a programmer to know how to call the procedures.

Macro declarations are shown as if they were procedures, except that "<macro>" is used instead of "PROCEDURE". "COMPILETIME" is shown before the declaration of macros evaluated at compiletime; other macros should be assumed to be evaluated at runtime. XIDAK reserves the right to replace macros with equivalently declared variables or procedures and vice versa without notice.

System variables are marked with the comment "# system variable".

Many of the system procedures are generic. In the procedure descriptions, only the generic names are listed. XIDAK reserves the right to change the instance procedure names without notice. XIDAK may also change a procedure that is not a generic into a generic procedure without notice. In the description of a generic procedure, the instances of the procedure are listed in the order in which they appear in the generic declaration.

Type codes are listed in Appendix A of part I of the "MAINSAIL Language Manual".

The "USES", "PRODUCES", and "MODIFIES" parameter qualifiers are described in Section 9.5 of part I of the "MAINSAIL Language Manual". "OPTIONAL" is discussed in Section 9.5.4 of part I of the "MAINSAIL Language Manual", and "REPEATABLE" in Section 9.5.5 of part I of the "MAINSAIL Language Manual".

Some procedures may have optional parameters that are not mentioned in the procedure descriptions. If the user specifies a non-Zero value for such a parameter, the effect is undefined.

When a procedure description refers to "the only valid bits" in a bits parameter b, only the bits specified should ever be set in b. Setting bits not mentioned in the description has undefined consequences.

Macros and the procedures qualified with "$BUILTIN" cannot trigger a garbage collection, nor can those qualified with "COMPILETIME" when all their arguments are constants, except as noted in the description. It may be assumed (unless otherwise noted in the description) that any other system procedures and macros may cause a collection to occur before returning.

If an error or an exception occurs during a call to a procedure or macro, a garbage collection may occur regardless of whether the procedure's or macro's description says it cannot trigger a collection.

1.1. The "COMPILETIME" Procedure Qualifier

A call to a procedure qualified with "COMPILETIME" is evaluated by the compiler at compiletime if all the arguments are constants.

For example, the system procedure "length" is declared as:

```
$BUILTIN COMPILETIME INTEGER PROCEDURE length (STRING s)
```

Writing "length("abc")" has the same effect as writing "3" into the program since the compiler evaluates it.

1.2. The "$BUILTIN" Procedure Qualifier

Calls to a procedure qualified with "$BUILTIN" usually generate efficient inline code sequences. XIDAK reserves the right to change any built-in procedure to a non-built-in procedure or to change a non-built-in to an built-in procedure without notice.

1.3. The "SPECIAL" Procedure Qualifier

Calls to a procedure qualified with "SPECIAL" require some sort of special attention from the compiler. Such procedures have properties that cannot be duplicated by the ordinary user.

1.4. Area Facility Declarations

In all system procedures with an optional "POINTER($area)" parameter, the parameter defaults to $defaultArea if omitted. The destination area specified for a string operation is used if the

operation generates new text; otherwise, the result text may or may not be in the specified area. For example:

```
ss := cvu(s,myArea)
```

puts the text into myArea if any characters are converted to uppercase; if no characters change case, then the text referenced by ss may refer to the same area as the text referenced by s. To ensure that text for a string is in a particular area, use $getInArea.

1.5. System Procedures, Variables, and Macros Summary

Table 1.5-1 contains a summary of all MAINSAIL system procedures, variables, and macros.

```
open              open a file
$reOpen           open a file with new open bits
close             close a file
$closedFile       determine whether a file has been closed

$createUniqueFile
                  create file with unique name

$devModBrk        device module name break character
$devModBrkStr     string consisting of $devModBrk

$delete           delete a file
$rename           rename a file

$copyFile         copy (part of) one file to another

$truncateFile     truncate a file to given length

getPos            get file position
setPos            set file position
relPos            set relative file position
$getEofPos        get end-of-file position of byte-stream
                  file
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

eof	true when positioned at or beyond end-of-file
$gotValue	determine if actually read last value; better way of checking for end-of-file
read	read values
write	write values
$storageUnitRead	read a number of data efficiently from a file
$storageUnitWrite	write a number of data efficiently to a file
$characterRead	read a number of characters efficiently from a file
$characterWrite	write a number of characters efficiently to a file
$pageRead	read a page of data from a file
$pageWrite	write a page of data to a file
cRead	read a character from file, string, or charadr
cWrite	write characters to file, string, or charadr
$clearFileCache	uncache all or part of file
$queryFileCacheParms	information about file cache
$setFileCacheParms	control file cache
$concat	concatenate strings (same as "&" operator)
$dup	perform multiple concatentations
rcRead	reverse character read (from the end of a string)
rcWrite	reverse character write (to the beginning of a string)
fldRead	read a string field
fldWrite	write a string field

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
ttyRead          read a line from "TTY"
ttyWrite         write values to "TTY"
ttycWrite        write characters to "TTY"

$removeBoolean   parse boolean string
$removeBits      parse bits string
$removeInteger   parse integer string
$removeReal      parse real string

confirm          get yes/no confirmation
cmdMatch         match a command (command recognition)
errMsg           raise an exception and/or write a message
                 and get a response

cmdFile          standard input file
logFile          standard output file

enterLogicalName
                 establish logical file name
lookUpLogicalName
                 find logical file name
$setSearchPath   set file searchpath

$globalLookup    look up global symbol
$globalEnter     enter global symbol
$globalRemove    remove global symbol

$registerException
                 register an exception name so that it can
                 be raised in response to an errMsg prompt
$deRegisterException
                 undo $registerException

$newException    assign a unique exception name

$raise           raise an exception
$raiseReturn     terminate an exception handler
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

$exceptionBits	return information about current exception
$exceptionName	return name of current exception
$exceptionCoroutine	return raising coroutine of current exception
$exceptionPointerArg	return pointer argument of current exception
$exceptionStringArg1, $exceptionStringArg2	return a string argument of current exception
scanSet	set up scan bit
$scanSet	set up scan integer
scanRel	release scan bits or integers
scan	scan a file or string according to a scan specification
$removeLeadingBlankSpace, $removeTrailingBlankSpace	remove blank space from string
$removeWord	remove non-blank chars from string
$formParagraph	fill and justify string
$cvbo	convert to boolean
cvi	convert to integer
cvli	convert to long integer
cvr	convert to real
cvlr	convert to long real
cvb	convert to bits
cvlb	convert to long bits
cvs	convert to string
cvp	convert to pointer
cva	convert to address
cvc	convert to charadr
cvAry	convert to array
cvcs	convert a character code to a single-character string
cvu	convert to upper case
cvl	convert to lower case

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

$length	length of result of cvs
first	first character of a string
last	last character of a string
length	number of characters in a string
compare	-1, 0 or 1 as result of (optionally "caseless") comparison of two strings
equ	checks (optionally "caseless") equality of two strings
isLowerCase	true if argument is a lowercase letter ("a" through "z")
isUpperCase	true if argument is an uppercase letter ("A" through "Z")
isAlpha	true if argument is a letter ("A" through "Z" or "a" through "z")
nextAlpha	alphabetically next character after argument character
prevAlpha	alphabetically previous character before argument character
isNul	true if argument is a "null" character
copy	copy a record, array, memory, or characters
clear	clear a record, array, memory, or characters
newUpperBound	adjust the upper bound of a one-dimensional array
$adrOfFirstElement	get the address of the first element of an array

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

new	allocate a record, array, or data section
$newRecords	allocate multiple records
dispose	dispose of a record, array, data section, or module
bind	bind a module
unBind	unbind a module
$canFindModule	whether a module can be allocated without error
$isBound	whether a module is already bound
$invokeModule	invoke a module the way MAINEX does
$useProgramInterface	true if bound because an interface procedure called
$programName	name under which MAINSAIL was invoked
$getCommandLine	get program arguments
$setCommandLine	set program arguments
thisDataSection	return pointer to current data section
$moduleName	return name of module, given data section pointer
$searchCallChain	find caller from particular module
$writeCalls	show call stack of coroutine
$fieldInfo	return information about a record or data section field
$className	return name of class of a pointer
$classInfo	return names and types of record or data section fields
$dscrPtr	class descriptor for pointer
$classDscrFor	class descriptor for a given class
$isArray	true if pointer points to an array

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

$createClassDscr	create a new class at runtime
$createRecord	create a record given a class descriptor
openLibrary	open a module library file
closeLibrary	close a module library file
setModName	set a module name association
relModName	release a module name association
setFileName	set a module file name association
relFileName	release a module file name association
exit	orderly exit from MAINSAIL
fastExit	fast exit from MAINSAIL
$setExitCode	set exit code for operating system
floor	largest (long) integer not exceeding a (long) real
ceiling	smallest (long) integer not exceeded by a (long) real
truncate	truncate a (long) real to a (long) integer
abs	absolute value of a (long) integer or (long) real
bMask	form a bits mask (sequence of 1-bits)
lbMask	form a long bits mask (sequence of 1-bits)

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
sin                 (long) real sine
cos                 (long) real cosine
tan                 (long) real tangent
$cot                (long) real cotangent
aSin                (long) real arcsine
aCos                (long) real arccosine
aTan                (long) real arctangent
$atan2              (long) real two-argument arctangent
sinh                (long) real hyperbolic sine
cosh                (long) real hyperbolic cosine
tanh                (long) real hyperbolic tangent
exp                 (long) real exponential
ln                  (long) real natural logarithm
log                 (long) real base-10 logarithm
sqrt                (long) real square root

$log2               truncated base 2 logarithm of constant

$hash               compute hash code

size                size of a class or data type
$ioSize             size of data type when written to file
$bitsPerStorageUnit
                    bits in a storage unit
$bitsPerChar        bits in a character unit

$typeName           name of a type, given type code

displace            displace a pointer, address, or charadr
displacement, lDisplacement
                    distance from one address or charadr to
                    another

eol                 end-of-line string
eop                 end-of-page string
tab                 tab string
$nulChar            null character

$pageSize           storage units per page
$charsPerPage       character units per page
$charsPerStorageUnit
                    character units per storage unit
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
(x)Load          load a value (of type x) from an address
cLoad            load a character from a charadr
store            store a value into an address or charadr

newString        make a string from a charadr and an
                 integer (length)

$getToTop        put a string at top of string space
$getInArea       put a string in an area's string space

newPage          get some pages
pageDispose      dispose of pages

newScratch       get some scratch space
$newScratchChars
                 get some scratch space measured in chars
scratchDispose   dispose of scratch space

$date            get the date
$time            get the time
$dateAndTime     get the date and time simultaneously
$setTheDate      set the date, if necessary

$assembleDate    convert year-month-date combination into
                 standard representation
$assembleTime    convert hour-minute-second combination
                 into standard representation
$assembleDateAndTime
                 combined $assembleDate and $assembleTime
$disassembleDate
                 convert standard representation into
                 year-month-date combination
$disassembleTime
                 convert standard representation into
                 hour-minute-second combination
$disassembleDateAndTime
                 $disassembleDate and $disassembleTime
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
$dateToStr          convert date representation to string
$timeToStr          convert time representation to string
$dateAndTimeToStr
                    combined $dateToStr and $timeToStr
$strToDate          convert string to date representation
$strToTime          convert string to time representation
$strToDateAndTime
                    combined $strToDate and $strToTime
$removeDateAndTime
                    parse date and time string

$addToDateAndTime
                    add two dates and times
$dateAndTimeDifference
                    subtract two dates and times
$dateAndTimeCompare
                    compare two dates and times

$dateFormat         whether date is GMT, local, or difference
$timeFormat         whether time is GMT, local, or difference

$convertDateAndTime
                    convert GMT time to local or vice versa
$timeSubcommandsSet
                    whether GMT conversion info available

$cpuTime            get system-dependent CPU time for current
                    program
$cpuTimeResolution
                    number of $cpuTime units per second

$timeout            pause for specified period

$userID             return the system-dependent user ID, if
                    available
$cpuID              return the system-dependent CPU ID, if
                    available
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
$currentDirectory
                name of system-dependent current working
                or connected directory or catalog
$homeDirectory  home directory or catalog of current user
$directory      list files in a directory
$fileInfo       return information about a file

$moduleInfo     information about objmod

$collect        perform a garbage collection
$checkConsistency
                verify that MAINSAIL data structures are
                in order
$addMemMngModule
                specify module to invoke before memory
                management operations
$removeMemMngModule
                undo $addMemMngModule

$collectLock    used to prevent/permit garbage collections

$overheadPercentExitValue
                used to prevent thrashing

$areaOf         determine area of pointer or string
$clearArea      empty an area
$clearStrSpc    empty an area's string space
$defaultArea    default area
$disposeArea    reclaim an area
$disposeDataSecsInArea
                dispose only data sections in area
$findArea       find area with given title
$inArea         determine if pointer or string in given
                area
$newArea        allocate area
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
$createCoroutine
                create a coroutine
$resumeCoroutine
                continue or start execution in a coroutine
$killCoroutine  get rid of a coroutine
$killedCoroutine
                determine whether a coroutine has been
                killed
$moveCoroutine  move coroutine to another point in tree
$findCoroutine  return a pointer to a coroutine record,
                given its name
$thisCoroutine  current coroutine

$majorVersion, $minorVersion
                get MAINSAIL version number

$maxChar        maximum character code
$maxInteger     maximum integer
$maxLongInteger maximum long integer
$minInteger     minimum integer
$minLongInteger minimum long integer

$platformNameAbbreviation, $platformNameFull,
   $platformNumber
                identify target platform
$systemNameAbbreviation, $systemNameFull, $systemNumber
                identify target operating system
$processorNameAbbreviation, $processorNameFull,
   $processorNumber
                identify target processor

$attributes     attributes of target system

$charSet        character set of target operating system

$preferredRadix "natural" radix for addresses, etc.

$compileTimeValue
                information about current compilation
$thisFileName   file name currently being compiled
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (continued)

```
$clrConfigurationBit
                clear bit governing runtime system
$clrSystemBit   clear bit governing runtime system
$setConfigurationBit
                set bit governing runtime system
$setSystemBit   set bit governing runtime system
$tstConfigurationBit
                examine bit governing runtime system
$tstSystemBit   examine bit governing runtime system
```

Table 1.5-1. System Procedures, Macros, and Variables Summary (end)

1.6. $abortProcedureExcpt

```
# system variable
STRING $abortProcedureExcpt;
```

Table 1.6-1. $abortProcedureExcpt

$abortProcedureExcpt is a predefined exception that is raised when the execution of a procedure is aborted, as described in Section 16.6 of part I of the "MAINSAIL Language Manual". $abortProcedureExcpt should be caught by all procedures that must clean up after themselves in some way.

1.7. $abortProgramExcpt

```
# system variable
STRING $abortProgramExcpt;
```

Table 1.7-1. $abortProgramExcpt

$abortProgramExcpt is a predefined exception that is raised to abort a program. $abortProgramExcpt is registered (with $registerException) at the start of a MAINSAIL execution, so it can be raised by giving an appropriate reply to the "Error response:" prompt. It is handled by falling out of the handler by MAINEX, MAINEDIT, MAINDEBUG, and other system programs that operate as command interpreters, and should be so handled by user programs that allow the invocation of arbitrary modules. $abortProgramExcpt basically means to abort the current program and return to the level of command interpreter from which it was invoked.

See also Section 16.8 of part I of the "MAINSAIL Language Manual".

1.8. abs

```
$BUILTIN COMPILETIME
INTEGER
PROCEDURE    abs          (INTEGER v);

$BUILTIN
REAL
PROCEDURE    abs          (REAL v);

$BUILTIN COMPILETIME
LONG INTEGER
PROCEDURE    abs          (LONG INTEGER v);

$BUILTIN
LONG REAL
PROCEDURE    abs          (LONG REAL v);
```

Table 1.8-1. abs (Generic)

abs returns the absolute value of a (long) integer or (long) real.

1.9. aCos

```
REAL
PROCEDURE    aCos         (REAL r);

LONG REAL
PROCEDURE    aCos         (LONG REAL r);
```

Table 1.9-1. aCos (Generic)

aCos returns the arccosine of its argument, which is in radians.

1.10. $addMemMngModule

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
PROCEDURE    $addMemMngModule
                          (POINTER dataSec);
```

Table 1.10-1. $addMemMngModule

The user may write a module that provides procedures to be invoked at the start and end of a garbage collection (garbage collections are not signaled through the exception mechanism, since the exception mechanism itself can trigger garbage collections).

A garbage collection interception module's class must have the predeclared class $memMngModule as its prefix class; see Table 1.10-2.

```
CLASS $memMngModule (
    PROCEDURE $startOfMemMng;
    PROCEDURE $endOfMemMng;
);
```

Table 1.10-2. The Class $memMngModule

The user indicates that a module of the class $memMngModule is to be a garbage collection interception module by calling $addMemMngModule. dataSec is the data section of the module. $addMemMngModule locks the data section in memory so that it is not swapped out and is therefore resident when called.

Immediately before a collection starts, $startOfMemMng in each garbage collection interception module is called. Procedures are called in the order of most recently added module (with $addMemMngModule) to least recently added.

Immediately after a garbage collection terminates, $endOfMemMng is called in each garbage collection module, in the same order as for $startOfMemMng.

To prevent an infinite recursion, $collectLock is incremented before calls to $startOfMemMng and $endOfMemMng. Therefore, the user risks an "Insufficient memory: exiting" termination of MAINSAIL if a call to $startOfMemMng or $endOfMemMng allocates any data, either directly or indirectly through any system facilities that allocate data.

A module is removed from the list of garbage collection interception modules by means of a call to $removeMemMngModule.

Example 1.10-3 shows a sample garbage collection interception module. The module shown should be bound from a program, not invoked from MAINEX; otherwise, the final procedure will execute immediately, removing the module from the list of garbage collection interception modules.

```
BEGIN "mmMsg"

MODULE($memMngModule) mmMsg;

INITIAL PROCEDURE;
$addMemMngModule(thisDataSection);



FINAL PROCEDURE;
$removeMemMngModule(thisDataSection);



# Assume ttyWrite cannot trigger a collection (true in
# line-oriented mode in the present version of MAINSAIL):

PROCEDURE $startOfMemMng;
ttyWrite("Doing memory management..." & eol);



PROCEDURE $endOfMemMng;
ttyWrite("Done with memory management." & eol);

END "mmMsg"
```

Example 1.10-3. Garbage Collection Interception Module

1.11. $addToDateAndTime

```
BOOLEAN
PROCEDURE    $addToDateAndTime
                        (LONG INTEGER date,time;
                         LONG INTEGER daysToAdd,
                             secondsToAdd;
                         PRODUCES LONG INTEGER
                             newDate,newTime;
                         OPTIONAL BITS ctrlBits);
```

Table 1.11-1. $addToDateAndTime

$addToDateAndTime performs addition on a date and time. newDate and newTime are the resulting date and time after daysToAdd and secondsToAdd have been added to date and time, respectively.

daysToAdd and secondsToAdd are interpreted as date and time differences (even if they fall outside the normal date and time difference range). The absolute value of secondsToAdd may exceed one day, and daysToAdd and secondsToAdd need not have the same sign.

date and time may be an absolute (GMT or local) date and time, or they may be a date and time difference. newDate and newTime have the same format (GMT, local, or difference) as date and time.

If either date or time is invalid, or if date and time do not have the same format, an error occurs, false is returned, and both newDate and newTime are set to 0L.

The only valid ctrlBits bit is errorOK. Unless it is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual". Adding a number of days to a given date or adding a number of seconds to a given time of day may be accomplished by means of the long integer "+" operator.

1.12. $adrOfFirstElement

```
ADDRESS
<macro>     $adrOfFirstElement
                        (ARRAY a);
```

Table 1.12-1. $adrOfFirstElement

$adrOfFirstElement returns the address of the first element (the element stored at the lowest memory address) of a. If a garbage collection occurs, a may be moved so that the value returned by $adrOfFirstElement before the collection is no longer correct. The effect is undefined if a is nullArray.

1.13. $allYearDigits

```
COMPILETIME
LONG BITS
<macro>     $allYearDigits;
```

Table 1.13-1. $allYearDigits

$allYearDigits is a bit that specifies that all digits of a year are to be included in the output of the procedure to which it is passed. It may be passed to $dateToStr and $dateAndTimeToStr.

1.14. $almostOutOfMemoryExcpt

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

The predefined exception $almostOutOfMemoryExcpt is raised when the maximum allowable memory is exhausted. The maximum allowable memory is the value specified to CONF's "MAXMEMORYSIZE" command when the current bootstrap was built, unless the integer variable $allowedMemoryPercent is in the range 1 to 99, inclusive, in which case the maximum

```
# system variable
STRING $almostOutOfMemoryExcpt;
```

Table 1.14-1. $almostOutOfMemoryExcpt

allowable memory is the specified percentage of the "MAXMEMORYSIZE" value. The user program may set $allowedMemoryPercent as desired.

When the $almostOutOfMemoryExcpt exception is raised, $allowedMemoryPercent is automatically increased to 100 so that the rest of memory is available for MAINSAIL to handle the exception. If MAINSAIL runs out of memory while attempting to handle $almostOutOfMemoryExcpt, it exits to the operating system after printing a message to "TTY".

The user's handler may reduce the value of $allowedMemoryPercent, if desired, but if reduced to a value less than the fraction of memory already consumed by MAINSAIL, memory is not somehow "given back" to the operating system.

Since there is no way at present for the user to ensure that the handling of $almostOutOfMemoryExcpt does not cause MAINSAIL to request more memory from the operating system, repeated catching of $almostOutOfMemoryExcpt is not guaranteed to work.

When $almostOutOfMemoryExcpt is raised, $exceptionStringArg1 is:

```
cvs($allowedMemoryPercent)
```

and $exceptionStringArg2 is:

```
cvs(<the number of pages needed>)
```

These strings may be located in scratch space instead of string space, so they must be copied if they are to be remembered after any subsequent raises of $almostOutOfMemoryExcpt.

1.15. alterOK

```
COMPILETIME
BITS
<macro>      alterOK;
```

Table 1.15-1. alterOK

alterOK is a bit that specifies that the target file may be deleted without prompting. It may be passed to $createUniqueFile, open, $rename, and $reOpen.

1.16. append

```
COMPILETIME
BITS
<macro>      append;
```

Table 1.16-1. append

append is a bit that specifies that a scan breaking character is to be appended to the scan result. It may be passed to scan.

1.17. $areaOf

```
$ALWAYSINLINE
POINTER($area)
PROCEDURE    $areaOf      (POINTER p);
```

Table 1.17-1. $areaOf (Generic) (continued)

```
$ALWAYSINLINE
POINTER($area)
PROCEDURE    $areaOf       (STRING s);
```

Table 1.17-1. $areaOf (Generic) (end)

$areaOf returns a pointer to the area containing the chunk pointed to by p or containing the text of s. The result is nullPointer if p or s is Zero and undefined if p or s is dangling.

1.18. $arithmeticExcpt

```
# system variable
STRING $arithmeticExcpt;
```

Table 1.18-1. $arithmeticExcpt

$arithmeticExcpt is a predefined exception that is raised when a (long) integer or (long) real overflow, underflow, or division by zero is detected by MAINSAIL. It may also be raised for certain other conditions, such as invalid floating point formats. On many systems, some or all of these conditions go undetected by MAINSAIL, so $arithmeticExcpt is not raised. On some systems, conditions that go undetected by default can be checked for by compiling modules with the "ACHECK" compiler subcommand.

1.19. aSin

```
REAL
PROCEDURE    aSin          (REAL r);

LONG REAL
PROCEDURE    aSin          (LONG REAL r);
```

Table 1.19-1. aSin (Generic)

aSin returns the arcsine of its argument, which is in radians.

1.20. $assembleDate

```
LONG INTEGER
PROCEDURE   $assembleDate
                          (INTEGER year;
                           OPTIONAL INTEGER month,day;
                           OPTIONAL BITS ctrlBits);
```

Table 1.20-1. $assembleDate

$assembleDate returns a MAINSAIL date given the year, month, and day.

year must not be 0, and must include the century (e.g., a value of 84 refers to the year 84 A.D., not to 1984 A.D.). If month and day are zero, they default to 1; otherwise, month must be between 1 (January) and 12 (December), inclusive, and day must be between 1 and 31, inclusive. An error occurs if a non-existent day of the month is specified, e.g., the 31st of June or the 29th of February in a non-leap year. If such an input value is detected, 0L is returned.

The valid ctrlBits bits are $localTime, $gmt, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the input values are interpreted as a local date and returned in local date format. If $gmt is specified, a GMT format date is returned.

Unless errorOK is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.21. $assembleDateAndTime

```
PROCEDURE    $assembleDateAndTime
                          (INTEGER year;
                           OPTIONAL INTEGER month,day,
                               hour,minute,second;
                           PRODUCES LONG INTEGER date,time;
                           OPTIONAL BITS ctrlBits);
```

Table 1.21-1. $assembleDateAndTime

$assembleDateAndTime produces a MAINSAIL date and time given a year, month, day, hour, minute and second. If month and day are not specified, they default to 1. The restrictions on the year, month, day, hour, minute, and second are the same as for $assembleDate and $assembleTime; date and time are set to 0L if erroneous values are detected.

Valid ctrlBits are errorOK, $localTime, $gmt, $localTimeToGMT, and $GMTtoLocalTime. Of the latter four bits, at most one can be specified; they are interpreted as shown in Table 1.21-2.

Bit	Input Parameters Interpreted as	Output Format for date and time
$localTime	Local time	Local format
$localTimeToGMT	Local time	GMT format
$GMTtoLocalTime	GMT	Local format
$gmt	GMT	GMT format

Table 1.21-2. $assembleDateAndTime ctrlBits Bits

If none of these four bits is specified, $localTime is assumed. The caveats described in Section 19.3 of part I of the "MAINSAIL Language Manual" regarding conversion between local time and GMT apply if $localTimeToGMT is set.

Unless errorOK is specified, an error message is generated for erroneous input values.

1.22. $assembleTime

```
LONG INTEGER
PROCEDURE    $assembleTime
                          (INTEGER hour;
                           OPTIONAL INTEGER minute,second;
                           OPTIONAL BITS ctrlBits;
                           PRODUCES OPTIONAL BOOLEAN
                               success);
```

Table 1.22-1. $assembleTime

$assembleTime returns a MAINSAIL time given the hour, minute, and second.

hour must be between 0 and 23, inclusive, and minute and second between 0 and 59, inclusive. All other values generate an error. If an erroneous input value is detected, 0L is returned and success is set to false. Note that 0L is also a valid return value if $timeDifference is set in ctrlBits and the time difference is zero seconds.

The valid ctrlBits bits are $localTime, $gmt, $timeDifference, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the input values interpreted as a local time and returned in local time format. If $gmt is specified, a GMT format time is returned. If $timeDifference is specified, hour, minute, and second are treated as a time difference; i.e., the value returned is:

```
cvli(hour) * 3600L + cvli(minute) * 60L + cvli(second)
```

Unless errorOK is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.23. aTan

```
REAL
PROCEDURE    aTan          (REAL r);

LONG REAL
PROCEDURE    aTan          (LONG REAL r);
```

Table 1.23-1. aTan (Generic)

aTan returns the arctangent of its argument, which is in radians.

1.24. $atan2

```
REAL
PROCEDURE    $atan2        (REAL y,x);

LONG REAL
PROCEDURE    $atan2        (LONG REAL y,x);
```

Table 1.24-1. $atan2 (Generic)

$atan2 returns the angle, in radians, with respect to the positive x-axis of a ray from the origin to a point with coordinates (x,y). Angles increase counterclockwise. The value returned is between minus pi and pi. If x and y are both positive, then "$atan2(y,x)" returns the same value as "atan(y / x)".

The effect is undefined if both x and y are zero, since such values do not define a ray with respect to the origin.

1.25. $attributes

```
COMPILETIME
LONG BITS
<macro>      $attributes;
```

Table 1.25-1. $attributes

Bits are set in $attributes depending on characteristics of the target processor/operating system combination. Bits of interest to the user are described in Appendix D of part I of the "MAINSAIL Language Manual".

1.26. binary

```
COMPILETIME
BITS
<macro>      binary;
```

Table 1.26-1. binary

binary is a bit that specifies that a binary string representation is input to or output from the procedure to which it is passed. It may be passed to cvb, cvlb, cvs, and $removeBits.

1.27. bind

```
SPECIAL
POINTER
PROCEDURE    bind          (MODULE m;
                            OPTIONAL BITS ctrlBits;
                            OPTIONAL POINTER($area) area);
```

Table 1.27-1. bind (Generic) (continued)

```
POINTER
PROCEDURE    bind          (STRING moduleName;
                            OPTIONAL BITS ctrlBits;
                            OPTIONAL POINTER($area) area);
```

Table 1.27-1. bind (Generic) (end)

If a bound data section is not already allocated for the module indicated by its argument, "bind" allocates a bound data section for it. In any case, bind returns a pointer to the bound data section. The control section is brought into memory, if necessary; it is found as described in Section 12.2 of part I of the "MAINSAIL Language Manual".

The pointer returned is of the class associated with the module m.

In the string form of bind, moduleName is the name of the module to be bound. This allows for those cases in which the name of the module to be bound has not been declared in the program (e.g., it may be obtained from the user). The returned pointer is unclassified since the compiler does not know the class of the anonymous module.

area is the area in which the bound data section is allocated.

Inaccessible bound data sections are not reclaimed by the garbage collector; bound data sections must be explicitly disposed. Inaccessible nonbound data sections, however, are collected.

The predefined valid bits constants for ctrlBits are shown in Table 1.27-2.

The procedure "new" (see Section 1.236) may be used to allocate non-bound data sections.

A related procedure, unBind, is described in Section 1.370.

1.28. $bitsPerChar

```
COMPILETIME
INTEGER
<macro>      $bitsPerChar;
```

Table 1.28-1. $bitsPerChar

Bit	Meaning
$programInterface	Cause the boolean macro $useProgramInterface, if called in the bound module's initial procedure before any other procedure is called, to return true (see Section 1.376).
errorOK	Return nullPointer if the module's control section cannot be bound instead of issuing an error message.

Table 1.27-2. Valid Bits for bind ctrlBits

$bitsPerChar is the number of bits in a character unit. It is always 8.

1.29. $bitsPerStorageUnit

```
COMPILETIME
INTEGER
<macro>     $bitsPerStorageUnit;
```

Table 1.29-1. $bitsPerStorageUnit

$bitsPerStorageUnit is the system-dependent number of bits in a storage unit.

1.30. bMask

```
COMPILETIME
BITS
PROCEDURE   bMask       (INTEGER lowBit,highBit);
```

Table 1.30-1. bMask

bMask makes a "bit mask", which is a contiguous sequence of 1-bits embedded within 0-bits.

bMask returns a bits that consists of 1-bits in the bit positions from lowBit to highBit, and 0-bits everywhere else. The bits are numbered from right to left starting with 0.

If b is a bits with all bits equal to 1-bit, then the result is the same as "(b SHL lowBit) CLR (b SHL (highBit + 1))". The result is undefined if either lowBit or highBit is less than 0 or greater than or equal to the number of bits in a bits, and the result is '0 if 0 LEQ highBit < lowBit.

lbMask (see Section 1.211) provides the same function for long bits.

A garbage collection cannot occur during a call to bMask.

```
bmask(3,8) = 'B111111000
bmask(0,0) = 'B1
bmask(1,0) = '0
```

Example 1.30-2. Use of bMask

1.31. break

```
COMPILETIME
BITS
<macro>     break;
```

Table 1.31-1. break

break is a bit that specifies that the scanning is to stop when one of the scan control characters is reached. It may be passed to scan.

1.32. $briefFormat

```
COMPILETIME
BITS
<macro>      $briefFormat;
```

Table 1.32-1. $briefFormat

$briefFormat is a bit that specifies that a brief date or time string is to be output. It may be passed to $dateAndTimeToStr, $dateToStr, and $timeToStr.

1.33. $canFindModule

```
BOOLEAN
PROCEDURE    $canFindModule
                          (STRING modName);
```

Table 1.33-1. $canFindModule

$canFindModule returns true if a call to "bind" or "new" would succeed for modName.

1.34. $cannotFallOut

```
COMPILETIME
BITS
<macro>      $cannotFallOut;
```

Table 1.34-1. $cannotFallOut

$cannotFallOut is a bit that specifies that a handler for the exception in question is not allowed to fall out. It may be passed to $raise and tested in $exceptionBits.

1.35. $cannotReturn

```
COMPILETIME
BITS
<macro>     $cannotReturn;
```

Table 1.35-1. $cannotReturn

$cannotReturn is a bit that specifies that a handler for the exception in question is not allowed to call $raiseReturn. It may be passed to $raise and tested in $exceptionBits. It is set in a call to $raise made from errMsg if the fatal bit is set in the call to errMsg.

1.36. $caseIndexExcpt

```
# system variable
STRING $caseIndexExcpt;
```

Table 1.36-1. $caseIndexExcpt

$caseIndexExcpt is a predefined exception that is raised when a Case Statement index does not match any of the Case Statement's selectors.

1.37. ceiling

```
INTEGER
PROCEDURE   ceiling     (REAL v);

LONG INTEGER
PROCEDURE   ceiling     (LONG REAL v);
```

Table 1.37-1. ceiling (Generic)

ceiling returns the smallest (long) integer greater than or equal to v.

See Table 1.169-3 for a table contrasting ceiling, cvi, floor, and truncate.

```
ceiling(10.5)  =  11
ceiling(-10.5) = -10
```

Example 1.37-2. Use of ceiling

1.38. $characterRead

```
LONG INTEGER
PROCEDURE   $characterRead
                          (POINTER(textFile) f;
                           LONG INTEGER numCharacters;
                           CHARADR memCharadr;
                           OPTIONAL BITS ctrlBits);
```

Table 1.38-1. $characterRead

$characterRead is the text file counterpart to $storageUnitRead, which works only on data files.

$characterRead reads up to numCharacters characters into memory at memCharadr. $characterRead does not filter out null characters; i.e., it acts as if the file had been opened with the keepNul bit set. If end-of-file occurs before numCharacters characters have been read, fewer than numCharacters are read; the number actually read is returned. The effect is undefined if at least numCharacters characters' worth of scratch memory is not allocated at memCharadr. If the file is opened for PDF I/O, the characters by default are translated from the PDF to the host character set.

The only valid ctrlBits bit is $noTranslate. If it is set and f is open for PDF I/O, no character translation from PDF to the host character set is done.

$characterRead may be called for an unbuffered file (a file opened with the $unbuffered bit set).

1.39. $characterWrite

```
PROCEDURE    $characterWrite
                        (POINTER(textFile) f;
                         LONG INTEGER numCharacters;
                         CHARADR memCharadr);
```

Table 1.39-1. $characterWrite

$characterWrite is the text file counterpart to $storageUnitWrite, which works only on data files.

$characterWrite writes numCharacters characters from memory starting at memCharadr to f. The effect is undefined if inaccessible memory is specified. If the file is opened for PDF I/O, the characters by default are translated from the host to the PDF character set.

The only valid ctrlBits bit is $noTranslate. If it is set and f is open for PDF I/O, no character translation from the host to the PDF character set is done.

$characterWrite may be called for an unbuffered file (a file opened with the $unbuffered bit set).

1.40. $charSet

```
COMPILETIME
INTEGER
<macro>      $charSet;
```

Table 1.40-1. $charSet

$charSet is the operating-system-dependent character set. Predefined values for $charSet are described in Appendix E of part I of the "MAINSAIL Language Manual".

1.41. $charsPerPage

```
COMPILETIME
INTEGER
<macro>      $charsPerPage;
```

Table 1.41-1. $charsPerPage

$charsPerPage returns the operating-system-dependent number of character units per page (equal to "$charsPerStorageUnit * $pageSize").

1.42. $charsPerStorageUnit

```
COMPILETIME
INTEGER
<macro>      $charsPerStorageUnit;
```

Table 1.42-1. $charsPerStorageUnit

$charsPerStorageUnit returns the operating-system-dependent number of character units per storage unit.

1.43. $checkConsistency

```
STRING
PROCEDURE    $checkConsistency;
```

Table 1.43-1. $checkConsistency

$checkConsistency traverses memory in the same way a garbage collection would, but it performs no collection. If it finds any inconsistencies in MAINSAIL's data structures, it returns a string error message. If it finds no inconsistency, it returns the null string.

Inconsistencies are frequently introduced when a pointer is used to modify the field of a disposed object (see Section 1.127), although other things can cause inconsistencies as well.

Strategically located calls to $checkConsistency may be used to determine where an inconsistency first appeared. The MAINSAIL utility MM provides a way to invoke $checkConsistency; see the "MAINSAIL Utilities User's Guide".

1.44. $classDscrFor

```
SPECIAL
POINTER($classDscr)
PROCEDURE    $classDscrFor
                          (CLASS c);
```

Table 1.44-1. $classDscrFor

$classDscrFor returns the class descriptor for c. It is currently implemented by looking in a table to see if there is already a class descriptor for c, and if not it creates one for it and stores it in the table. Since $classDscrFor involves a search, if a given class descriptor pointer is to be used often, it is more efficient to do "p := $classDscrFor(c)" and reuse p than to call $classDscrFor repeatedly. This procedure returns the same pointer as "$dscrPtr(new(c))", but no new record of the class needs to be allocated.

1.45. $classInfo

```
BOOLEAN
PROCEDURE    $classInfo   (POINTER p;
                           PRODUCES OPTIONAL STRING
                               className,fieldNames,
                               fieldTypes;
                           PRODUCES OPTIONAL
                               POINTER($classDscr) q;
                           OPTIONAL POINTER($area) area);
```

Table 1.45-1. $classInfo

p is a pointer to a record, class descriptor, or data section. If p is invalid, false is returned; otherwise, true is returned, and the produces parameters are set. className is set to the name (all upper case) of the associated class (i.e., to the same value returned by "$className(p)"). fieldNames is set to the non-procedure field names (all upper case) separated by eol's, fieldTypes to the field type codes separated by eol's. q is set to the class descriptor for p (the class descriptor for p may be obtained more efficiently by calling $dscrPtr). A unique class descriptor exists for each class currently in use. area specifies the destination area for any string text generated.

```
    CLASS c
        (INTEGER i; REAL x; STRING s; POINTER(c) link);
    ...
    p := new(c);
    $fieldInfo(p,className,fieldNames,fieldTypes);

The produces parameters are set as follows:

className = "C"
fieldNames = "I<eol>X<eol>S<eol>LINK"
fieldTypes = "2<eol>4<eol>8<eol>11"

(2 = integerCode, 4 = realCode, 8 = stringCode,
 11 = pointerCode)
<eol> indicates an embedded end-of-line string.
```

Example 1.45-2. Use of $classInfo

1.46. $className

```
STRING
PROCEDURE    $className   (POINTER p);
```

Table 1.46-1. $className

p is a pointer to a record, class descriptor, or data section. The name of the associated class is returned. If there is no associated class (e.g., p is nullPointer), the null string is returned. The effect if p points to a data section is described in Section 1.46.1.

Class descriptor pointers may be obtained with $createClassDscr, $classInfo, or $dscrPtr.

Since class names need not be unique within a MAINSAIL program, the correct way to determine whether two records are of the same class is to compare their class descriptor pointers (as returned by $dscrPtr).

1.46.1. $className of a Data Section

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

The behavior of "$className(p)" if p points to a data is complicated and subject to change.

Let p point to a data section for some module M:

- If M has no interface data fields, $className returns the null string.
- Otherwise:
    - If M has interface data fields that were not declared in some prefix class for M, then $className returns the module name, i.e., "M". What really happens is that the compiler creates a "dummy" class with name "M".
    - Otherwise, M must have a prefix class that contributed the data field(s), say a class C; $className returns the name of the prefix class, i.e., "C".

See Example 1.46.1-1.

It is not obvious that $className should behave this way, so the behavior is subject to change. The reasoning behind this approach is that $className returns the name of the class that describes the interface data fields of the data section. If there are no interface data fields, then it returns null string; otherwise, it returns the name of the actual class if all the data fields came from a class; otherwise, it returns the name of the dummy class created by the compiler to describe the fields declared for the module, and the compiler happens to give this class the name of the module.

Given the declarations:

```
MODULE m (PROCEDURE foo);

MODULE n (INTEGER i; ...);

CLASS c (INTEGER j; ...);

MODULE(c) o (PROCEDURE foo);

MODULE(c) p (INTEGER i; ...);

CLASS d (PROCEDURE bar);

MODULE(d) q (PROCEDURE foo);
```

then:

if p points to a data section for:	then "$className(p)" returns:
M	""
N	"N"
O	"C"
P	"P"
Q	""

Example 1.46.1-1. Behavior of $className with Data Section Arguments

1.47. clear

```
$BUILTIN
PROCEDURE   clear       (ADDRESS dst;
                         INTEGER n);

$BUILTIN
PROCEDURE   clear       (CHARADR dst;
                         INTEGER n);
```

Table 1.47-1. clear (Generic) (continued)

```
$BUILTIN
PROCEDURE    clear        (ADDRESS dst;
                           LONG INTEGER n);

$BUILTIN
PROCEDURE    clear        (CHARADR dst;
                           LONG INTEGER n);

PROCEDURE    clear        (POINTER p);

PROCEDURE    clear        (LONG ARRAY dst;
                           OPTIONAL INTEGER n);

PROCEDURE    clear        (LONG ARRAY dst;
                           OPTIONAL LONG INTEGER n);
```

Table 1.47-1. clear (Generic) (end)

"clear" is used to clear storage units, characters, a record, a data section, or an array.

The address forms of clear set the contents of n storage units starting with dst to Zero. dst must be an aligned address and n must be a multiple of the size of a MAINSAIL data type; otherwise, the effect is undefined.

The charadr forms of clear set the n character positions starting with dst to the character code 0.

The effect of the address and charadr forms of clear is undefined if dst is Zero.

The pointer form of clear clears the record or data section pointed to by p; nothing happens if p is nullPointer. Each field of the record pointed to by p is set to Zero. Clearing a data section has the effect of clearing the interface, outer, and own variables. It is not specified whether implicit module pointers are cleared (see Section 10.7 of part I of the "MAINSAIL Language Manual").

The array forms of clear set the first n elements of the array dst to Zero (nothing happens if dst is Zero). n is determined as follows:

```
m := IF NOT n .MAX 0 THEN <number of elements in array>
     ELSE n MIN <number of elements in array>
```

If n is Zero in an address or charadr form of clear or negative in any form of clear, nothing is cleared.

A garbage collection cannot occur during a call to clear unless the object to be cleared is a data section.

```
INTEGER i;
INTEGER ARRAY(0 TO 1000) ary;
...
FOR i := 0 UPTO 1000 DO ary[i] := 0;

# could be written as clear(ary)
```

Example 1.47-2. Use of clear for an Array

1.48. $clearArea

```
PROCEDURE   $clearArea  (REPEATABLE POINTER($area) area);
```

Table 1.48-1. $clearArea

$clearArea clears area, i.e., sets it to its state immediately after allocation, thereby freeing most of the memory occupied by area.

1.49. $clearStrSpc

```
PROCEDURE   $clearStrSpc
                        (POINTER($area) area);
```

Table 1.49-1. $clearStrSpc

$clearStrSpc clears only the string space part of area. area's string space is cleared (converted to empty string space, not free pages); the effects of subsequently accessing the text of strings referencing area at the time $clearStrSpc was called are undefined.

One use of $clearStrSpc is shown in Example 1.49-2. The contents of a large array of real numbers are to be written to a file in a particular format (no exponent, six digits after the decimal point). Calling cvs many times to write a number in this format uses a lot of string space, but each string is used just once; it is created, written to the file, and then never used again. The code in Example 1.49-2 clears the string space of the area "Foo" after every hundredth string written, so that no more than a hundred numbers' worth of string space is in use at any time. $clearStrSpc executes fairly quickly, so it is not unreasonable to call it after every hundredth string is computed.

```
INTEGER i;
LONG INTEGER ii;
POINTER($area) myArea;
REAL LONG ARRAY(1L TO 100000L) rAry;
POINTER(textFile) f;
...
area := $newArea("Foo",$noCollectablePtrs!
    $noCompactablePtrs!$noCollectableStrs,2000L);
i := 0;
FOR ii := 1L UPTO 100000L DOB
    write(f,cvs(rAry[ii],fixed!'6,myArea),eol);
    IF i .+ 1 = 100 THENB
        i := 0; $clearStrSpc(myArea) END END;
$disposeArea(myArea);
```

Example 1.49-2. Use of $clearStrSpc

1.50. $clearFileCache

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

$clearFileCache removes some or all of f's buffers from the file cache LRU list, optionally writes dirty buffers, and optionally uncaches f. f's current buffer is altered if necessary, as described below. If f is nullPointer, an error occurs and $clearFileCache returns FALSE. No action is taken and $clearFileCache returns true if f is not cached.

$clearFileCache removes from the LRU list all of f's buffers of which the buffer boundary is greater than or equal to pos. Dirty buffers removed from the LRU list are written if the delete bit is not set in ctrlBits.

```
BOOLEAN
PROCEDURE    $clearFileCache
                          (POINTER(file) f;
                           OPTIONAL BITS ctrlBits;
                           OPTIONAL BOOLEAN unCacheFile;
                           OPTIONAL LONG INTEGER pos);
```

Table 1.50-1. $clearFileCache

If pos is not on a buffer boundary and falls within a cached buffer, the remainder of the buffer is cleared and it is marked as dirty. If pos is equal to the bufPos of the current buffer, the buffer is cleared and marked as not dirty. In either case, if the buffer is dirty and the delete bit is not set in ctrlBits, it is written before it is altered. The buffer remains cached.

If pos is nonZero, then unCacheFile is ignored. Otherwise, if unCacheFile is true, f is uncached (the current buffer is removed from the cache and f's cache is disposed of). If unCacheFile is false, the current buffer remains cached.

Valid ctrlBits are errorOK and delete. An error message is generated if an error occurs and errorOK is not specified.

1.51. cLoad

```
$BUILTIN
INTEGER
PROCEDURE    cLoad          (CHARADR c;
                             OPTIONAL INTEGER dspl);
```

Table 1.51-1. cLoad

cLoad loads a character from a charadr. Another form of cLoad, which loads a charadr from an address, is described in Section 1.216.

cLoad loads a character from the location given by "displace(c,dspl)", where dspl is a displacement in characters.

The effect is undefined if c is Zero or if "displace(c,dspl)" is undefined. See Example 1.51-2.

```
INTEGER i;
ADDRESS a;
CHARADR c;

c := cLoad(a); # loads a charadr from location given by a
i := cLoad(c); # loads a character from character location
               # given by c
```

Example 1.51-2. Use of cLoad

1.52. close

```
PROCEDURE   close       (MODIFIES POINTER(file) f;
                         OPTIONAL BITS closeBits);
```

Table 1.52-1. close

"close" closes (and optionally deletes) a file.

f is closed according to directives in closeBits, and then set to nullPointer.

The only valid predefined bits constants for closeBits are delete and errorOK. If delete is specified, it indicates that the file should be deleted. The delete bit can also be specified to open (Section 1.259), in which case the file is deleted when closed regardless of whether the delete bit is specified to close.

The errorOK bit suppresses an error message if delete is specified and the file cannot be deleted.

A file that was opened at some point during program execution should always be closed before execution is complete. MAINSAIL automatically closes all files that are open at the end of program execution; however, closing a file may free up resources (e.g., memory or operating-system-dependent file handles) associated with an open file, so it is best to close a file as soon as it is no longer needed.

Closing cmdFile or logFile has the effect of reopening it to "TTY".

```
POINTER(dataFile) f;
...
open(f,"Input file: ",create!random!output!prompt);
...
close(f,delete);     # never again need f
```

Example 1.52-2. Use of close

1.53. $closedFile

```
BOOLEAN
<macro>     $closedFile     (POINTER(file) f);
```

Table 1.53-1. $closedFile

The macro $closedFile returns true if and only if the file f has been closed by a call to close.

1.54. closeLibrary

```
PROCEDURE   closeLibrary
                          (STRING fileName);
```

Table 1.54-1. closeLibrary

closeLibrary closes the library file (see Chapter 12 of part I of the "MAINSAIL Language Manual") with the name fileName, thereby eliminating it from taking part in module searches (unless opened again).

Modules already obtained from the library are not affected by the library's being closed. Thus, it is possible to open a library, obtain a module from the library, close the library, and then continue to use the module. The runtime system preserves a copy of the module until it is disposed.

Example 1.54-2 calls a procedure p in a module m, assuming that the file "myLib" is an objmod library that contains m.

```
MODULE m (PROCEDURE p);
...
openLibrary("myLib");
bind(m);
closeLibrary("myLib");
p;
```

Example 1.54-2. Use of closeLibrary

1.55. $clrConfigurationBit

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
<macro>     $clrConfigurationBit
                          (BITS b);
```

Table 1.55-1. $clrConfigurationBit

$clrConfigurationBit clears various bits that control MAINSAIL execution. The bits are documented in detail under $setConfigurationBit.

1.56. $clrSystemBit

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

$clrSystemBit clears various bits that control MAINSAIL execution. The bits are documented in detail under $setSystemBit.

```
<macro>     $clrSystemBit
                        (BITS b);
```

Table 1.56-1. $clrSystemBit

1.57. cmdFile

```
# system variable
POINTER(textFile) cmdFile;
```

Table 1.57-1. cmdFile

cmdFile is MAINSAIL's standard input file. cmdFile and logFile are described in Section 18.12 of part I of the "MAINSAIL Language Manual".

1.58. $cmdFileEofExcpt

```
# system variable
STRING $cmdFileEofExcpt;
```

Table 1.58-1. $cmdFileEofExcpt

$cmdFileEofExcpt is a predefined exception that is raised when the end of cmdFile is reached unless the configuration bit $noAutoCmdFileSwitching is set, as described in Section 18.12 of part I of the "MAINSAIL Language Manual".

1.59. cmdMatch

```
INTEGER
PROCEDURE    cmdMatch     (STRING ARRAY(*) cmds;
                           OPTIONAL STRING promptString;
                           OPTIONAL BITS ctrlBits;
                           PRODUCES OPTIONAL STRING s);
```

Table 1.59-1. cmdMatch

cmdMatch is a standard means of presenting a menu of commands to the user and recognizing the user's choice or displaying the choices in response to a "?" prompt.

cmdMatch matches a string (the match string) against the elements of cmds, and returns the index of the element that matches. The default case (in which ctrlBits is '0) is described first; it can be altered by setting various bits in ctrlBits.

promptString is written to logFile. A string is then read from cmdFile into s. The match string is s.

A caseless comparison (see Section 4.8.1 of part I of the "MAINSAIL Language Manual") is done between the match string and the elements of cmds (starting with the first) until either the match string is matched or all elements have been examined.

If the match string exactly (ignoring upper and lower case distinctions) matches a command, then that command is taken as the target command, and no further commands are examined.

If the match string matches an initial part of exactly one command, then that command is taken as the target command.

If the match string matches an initial part of more than one command, "...ambiguous" is written to logFile and a new match string is read from cmdFile.

If the match string matches no command, "...invalid" is written to logFile unless the first character of the match string is "?", in which case the valid commands are written, one per line (the null string is written as "<eol>"). A new match string is read from cmdFile, and the matching process begins again.

If the ctrlBits parameter to cmdMatch has both noResponse and errorOK set, and promptString is "?", the list of possible responses is written to logFile, as usual, and the value for "ambiguous" (one greater than the upper bound of the cmds array) is returned.

The valid predefined bits constants shown in Table 1.59-2 may be set in ctrlBits to alter the behavior of cmdMatch.

Example 1.59-3 shows how a comment may be put after the part of the command that is to take part in the match.

```
STRING ARRAY(1 TO 3) commands;
...
IF NOT commands THENB
    new(commands);
    INIT commands
        (
        "NO HERALD  do not put a herald on each page",
        "NO PAGES   do not print page numbers",
        "COUNT      do not print: just count pages",
        );
    END;

commandIndex := cmdMatch(commands,"command: ");

"command: " is written to logFile, and cmdMatch gets a
reply from cmdFile.

If "c", "C", "co", or "count" is typed, commandIndex is
set to 3.

If "no" is typed, "...ambiguous" is written to logFile,
and a new command is read from cmdFile.

If "pages" is typed, "...invalid" is written to logFile,
and a new command is read from cmdFile.

If "?" is typed, the commands are typed, each on a new
line, then a new command is read from cmdFile.
```

Example 1.59-3. Use of cmdMatch

errorOK	If the match string is ambiguous or invalid, no message is written to logFile. An index one greater than the upper bound (if the match string is ambiguous) or one less than the lower bound (if it is invalid) of cmds is returned.
noResponse	Do not write the promptString to logFile or read the match string from cmdFile; instead use the promptString as the match string.
useKeyWord	The first word (non-blank, non-tab characters delimited by blanks or tabs) is removed from the match string and is used as the match string; the remainder of the match string, leading blanks and tabs removed, is put into s. Matches are attempted with the first word of each element of cmds. If no match occurs and errorOk (or noResponse) is set, the original match string is produced in s. This bit might be used, for example, for match strings of the form: <keyword><blanks><parameters> for which <keyword> is used as the match string, and <parameters> is produced in s.
upperCase	The cases of letters in the commands in the cmds array are to be used to determine the minimum unambiguous abbreviations for commands. The part of the command sufficient to cause a string to match (if it would otherwise be ambiguous) should be uppercase, and the remainder of the command lowercase.

Table 1.59-2. Predefined Bits Constants for the cmdMatch ctrlBits Parameter

Example 1.59-4 shows a use of the useKeyWord option. If the user types the line "pages 5", cmdIndex is set to 1 and s is "5". Thus the "5" does not take part in the matching process, but is produced in the final argument.

```
STRING ARRAY(1 TO 2) cmds;
INTEGER n;
STRING  s;
...
new(cmds); INIT cmds ("PAGES","COPIES");
...
cmdIndex := cmdMatch(cmds,"command: ",useKeyWord,s);
n := cvi(s);
```

Example 1.59-4. Use of useKeyWord Option with cmdMatch

As an example of the use of the upperCase bit, if two cmds elements are:

```
Closelibrary
CLOSEINTLIB
```

then "C" matches "CLOSELIBRARY" instead of being ambiguous, as it would be if upperCase were not set. The effect of following lowercase letters with uppercase letters in a command element (e.g., "ClosELibRARY") is undefined.

In detail, the rules for upperCase are:

- if the match string is exactly the same as a cmds element, ignoring case, that element is taken to match, as usual, else
- if the match string is a prefix of only one cmds element, ignoring case, that element is taken to match, as usual, else
- the cmds element is chosen of which the match string matches at least the entire uppercase prefix; if several such such elements exist, the first one with the longest uppercase prefix is chosen from them;
- if no element matches according to the above criteria, the match string is considered ambiguous or invalid according to the usual rules.

So, in the case of the cmds elements "Closelibrary" and "CLOSEINTLIB", "CL", "CLO", "CLOS", and "CLOSE" also all match "CLOSELIBRARY". In the case of the cmds elements:

```
ABc
Abc
```

"A" matches "Abc" and "AB" matches "ABc". In the case of the cmds elements:

```
CHeck
CHECKALL
```

"C" is ambiguous; "CH", "CHE", "CHEC", and "CHECK" all match "CHeck".

1.60. $collect

```
PROCEDURE    $collect     (OPTIONAL INTEGER
                               kindOfCollection);
```

Table 1.60-1. $collect

The procedure $collect causes a garbage collection to occur.

If kindOfCollection is equal to stringCode, only strings are collected. If equal to pointerCode, only non-strings (arrays, records, and data sections) are collected. If kindOfCollection is Zero, all collectable data are collected.

1.61. $collectableChkSpc

```
COMPILETIME
LONG BITS
<macro>      $collectableChkSpc;
```

Table 1.61-1. $collectableChkSpc

$collectableChkSpc is a bit that specifies that an area's chunks are to be collected in automatic garbage collections. It may be passed to $newArea.

1.62. $collectableStrSpc

```
COMPILETIME
LONG BITS
<macro>     $collectableStrSpc;
```

Table 1.62-1. $collectableStrSpc

$collectableChkSpc is a bit that specifies that an area's strings are to be collected in automatic garbage collections. It may be passed to $newArea.

1.63. $collectLock

```
# system variable
INTEGER $collectLock;
```

Table 1.63-1. $collectLock

No automatic collection occurs if $collectLock is non-zero (collections may still be triggered explicitly with $collect). Customarily, $collectLock is incremented by one before a section of code in which collections are not to occur, then decremented by one afterwards. Assignment of a specific value to $collectLock without saving the previous value is undefined, since $collectLock may have a non-zero value even in ordinary user code.

If the programmer is not careful to redecrement $collectLock, collections may be locked out indefinitely, which may cause the MAINSAIL execution to run out of memory.

1.64. $compactableChkSpc

```
COMPILETIME
LONG BITS
<macro>     $compactableChkSpc;
```

Table 1.64-1. $compactableChkSpc

$compactableChkSpc is a bit that specifies that an area's chunks may be compacted (i.e., moved around) in automatic garbage collections. It may be passed to $newArea.

1.65. compare

```
INTEGER
PROCEDURE   compare     (STRING r,s;
                         OPTIONAL BITS ctrlBits);
```

Table 1.65-1. compare

"compare" returns an integer that represents the comparison of r and s.

compare returns -1 if r is less than s, 0 if r is equal to s, or 1 if r is greater than s. String comparison is discussed in Section 4.8.1 of part I of the "MAINSAIL Language Manual".

A single valid bits constant, upperCase, may be set in ctrlBits. If set, it means to ignore distinctions between upper- and lowercase letters in the strings when doing the comparison. "compare(r,s,upperCase)" returns the same value as "compare(cvu(r),cvu(s))" but is more efficient.

In Example 1.65-2, the first form is more efficient since only a single comparison takes place. In the second form two implicit calls to compare are generated, one for "<" and one for "=".

```
    CASE compare(r,s) OFB [-1] s1; [0] s2; [1] s3 END

is equivalent to

    IF r < s THEN s1
    EF r = s THEN s2
    EL s3
```

Example 1.65-2. Use of compare

1.66. $compareIntmods

Two intmods can be compared using the procedure $compareIntmods. This feature is documented in detail under INTCOM in the "MAINSAIL Utilities User's Guide".

1.67. $compareObjmods

Two objmods can be compared using the procedure $compareObjmods. This feature is documented in detail under OBJCOM in the "MAINSAIL Utilities User's Guide".

1.68. $compile

$compile is a system procedure that invokes the MAINSAIL compiler. This feature is documented in detail in the "MAINSAIL Compiler User's Guide".

1.69. $compileTimeValue

```
SPECIAL COMPILETIME
STRING
PROCEDURE   $compileTimeValue
                          (STRING valueName);
```

Table 1.69-1. $compileTimeValue

The argument of the compiletime procedure $compileTimeValue must be a string constant that consists of a keyword, possibly followed by arguments if the keyword takes arguments. Case is not distinguished in the keywords.

For each of the following keywords, the result is "TRUE" if the compiler option is in effect, else the null string:

```
ALIST DEBUG FLDXREF GENCODE GENINLINES INCREMENTAL ITFXREF
MODTIME PERMOD PERPROC PERSTMT PROCS PROCTIME RECOMPILE
RESPONSE SAVEON SLIST UNBOUND
```

The effects of the following keywords are described in Chapter 15 of part I of the "MAINSAIL Language Manual":

```
OPTIMIZE CHECKINGSTATUS LOCALCHECKINGSTATUS ACHECKINGSTATUS
LOCALACHECKINGSTATUS
```

$compileTimeValue("MONITOR") is "TRUE" if and only if:

```
$compileTimeValue("PERMOD") AND
$compileTimeValue("PERPROC") AND
$compileTimeValue("PERSTMT") AND
$compileTimeValue("PROCTIME") AND
$compileTimeValue("MODTIME")
```

$compileTimeValue("VERSION") returns a string in the form:

```
<majorVersion>.<minorVersion>
```

representing the current MAINSAIL version, e.g., "12.10".

$compileTimeValue("FLI") is the FLI specification given to the "FLI" subcommand (e.g., "TC", "FP") if the current compilation is for an FLI, or the null string if the current compilation is not an FLI compilation.

$compileTimeValue("RPC") returns:

```
""          if the compilation is not an RPC compilation
"C"         if the compilation is a C RPC compilation
"MAINSAIL"  if the compilation is a MAINSAIL RPC compilation
```

$compileTimeValue("ERRORS") is "TRUE" if any errors have occurred in the current compilation, the null string otherwise.

$compileTimeValue("THISFILENAME") is the name of the current source file, i.e., is equivalent to $thisFileName.

$compileTimeValue("SOURCEFILE") returns the name of the file that sourcefiled the current file, if it was sourcefiled; otherwise, it returns the null string.

$compileTimeValue("SOURCEFILE <file name>") returns "TRUE" if the named file can be opened as a text file; otherwise, it returns the null string. This is useful to know whether or not the current file is being used as a sourcefile, and could be used for deciding, e.g., whether to compile a header (if being sourcefiled) or a module (if being compiled as a top-level file).

$compileTimeValue("THISMODULENAME") is the uppercase name of the module being compiled, or the null string if the initial "BEGIN" and module name of a module have not yet been encountered.

$compileTimeValue("THISPROCEDURENAME") is the name of the procedure currently being compiled, or the null string if a procedure body is not being compiled.

$compileTimeValue("THISPAGENUMBER") is the current source file page number (where pages are delimited by eop characters).

$compileTimeValue("THISLINENUMBER") is the current source file line number (relative to the start of the current page).

$compileTimeValue("DATEANDTIME") is the current date and time in the format:

```
dd-mm-yy hh:mm:ss
```

$compileTimeValue("DATEANDTIME") recomputes the date and time for each call.

$compileTimeValue("RESTOREFROM <file or module name>") returns "TRUE" if it is possible to perform a restorefrom from the named file or module, otherwise the null string. No restorefrom is actually done.

$compileTimeValue("HASBODY <procedure>") is "TRUE" if <procedure> is the name of a procedure of which the body has been parsed during this compilation, the null string otherwise. The test:

```
NOT $compileTimeValue("HASBODY <procedure>")
```

differs from:

```
NEEDBODY(<procedure>)
```

in that the latter is true only if the procedure has been declared as forward and called (or is an interface procedure of the current module), but not given a body; the former is true if and only the procedure has not been given a body, regardless of whether it has been declared as forward or called.

1.70. $concat

```
COMPILETIME
STRING
PROCEDURE    $concat      (STRING r,s;
                           OPTIONAL POINTER($area) area);
```

Table 1.70-1. $concat

The procedure $concat performs the operation specified by the MAINSAIL string concatenation operator, "&"; i.e.:

```
$concat(r,s) = r & s
```

area is the destination area for the resulting string. $concat needs to be used instead of "&" only if area is specified.

1.71. confirm

```
BOOLEAN
PROCEDURE    confirm      (OPTIONAL STRING msg,val);
```

Table 1.71-1. confirm

confirm gets a yes-no confirmation from cmdFile.

msg is first written to logFile. If val is not "", it is written to logFile, preceded and followed by a blank. Then "(Yes or No):" is written to logFile. The user may respond with "yes" (or "y") for yes, "no" (or "n") for no (case is not distinguished), or "?" for a help message. confirm returns true if the response is yes, false if the response is no, and reprompts if the response is "?".

In Example 1.71-2, if the user types "y<eol>" or "Y<eol>", true is returned. If the user types "n<eol>" or "N<eol>", false is returned. Otherwise, a message is written to logFile, and the user is reprompted until a valid response is obtained.

```
    IF confirm("OK to delete",f.name) THEN ...

The following is written to logFile
if f.name = "WORKSHEET":

    OK to delete WORKSHEET (Yes or No):
```

Example 1.71-2. Use of confirm

1.72. $convertDateAndTime

```
BOOLEAN
PROCEDURE   $convertDateAndTime
                        (LONG INTEGER inputDate,inputTime;
                         PRODUCES LONG INTEGER
                             outputDate,outputTime;
                         OPTIONAL BITS ctrlBits);
```

Table 1.72-1. $convertDateAndTime

$convertDateAndTime converts a local date and time to GMT or vice versa, depending on the format of the input date and time. A date or time difference is not a valid input to $convertDateAndTime. True is returned if and only if no error occurs.

If inputDate and inputTime are in local time format, $convertDateAndTime converts a local date and time, represented by inputDate and inputTime, to a Greenwich Mean Time date and time, represented by outputDate and outputTime. It takes into account whether daylight savings time is, was, or will be in effect at the local date and time; however, it is not specified which GMT date and time is returned for the (ambiguous) local date and time during the transition between daylight and standard time.

If inputDate and inputTime are in GMT format, $convertDateAndTime converts a Greenwich Mean Time date and time, represented by inputDate and inputTime, to a local date and time,

represented by outputDate and outputTime. It takes into account whether daylight savings time is, was, or will be in effect locally at the resulting local date and time.

The conversion may make an incorrect adjustment for daylight savings time if the current algorithm for daylight savings time is not, was not, or will not be that in effect at the time represented by inputDate and inputTime (e.g., due to incorrect values specified in the MAINSAIL bootstrap or a change in statutes governing daylight savings time). The conversion may fail if $timeSubcommandsSet is false and the date is too close to the earliest or latest day the operating system can represent (e.g., 1 January 1970 on UNIX).

errorOK may be set in ctrlBits. If errorOK is set, any error messages that might be generated by invalid input values are suppressed.

1.73. copy

```
$BUILTIN
PROCEDURE    copy        (ADDRESS src,dst;
                          INTEGER n);

$BUILTIN
PROCEDURE    copy        (CHARADR src,dst;
                          INTEGER n);

$BUILTIN
PROCEDURE    copy        (ADDRESS src,dst;
                          LONG INTEGER n);

$BUILTIN
PROCEDURE    copy        (CHARADR src,dst;
                          LONG INTEGER n);

PROCEDURE    copy        (POINTER src,dst);

PROCEDURE    copy        (LONG ARRAY src,dst;
                          OPTIONAL INTEGER n);

PROCEDURE    copy        (LONG ARRAY src,dst;
                          OPTIONAL LONG INTEGER n);
```

Table 1.73-1. copy (Generic)

copy is used to copy storage units, characters, a record, or an array.

The pointer form of copy copies the record pointed to by src to the record pointed to by dst. The number of storage units copied is determined by the smaller of the two records. The effect is undefined if src and dst are not class compatible. It is an error if src or dst is a pointer to a data section.

The array forms of copy copy the first m elements of the src array to the first m elements of the dst array. The two arrays must be of the same data type. m is determined as follows:

```
m1 := elementsInSrcArray MIN elementsInDstArray;
m := IF NOT n .MAX 0(L) THEN m1 ELSE n MIN m1;
```

The address forms of copy copy n storage units from memory starting at src to memory starting at dst. src and dst must be aligned addresses and n must be a multiple of the size of a MAINSAIL data type; otherwise, the effect is undefined.

The charadr forms of copy copy n characters from memory starting at src to memory starting at dst.

Neither src nor dst may be Zero for any form of copy. For the pointer and array forms, this situation generates an error; for the address and charadr forms, the result is undefined.

If n is Zero or negative in any form of copy, no data movement takes place.

In the address and charadr forms, the effect is undefined if the source and destination areas of memory overlap.

A garbage collection cannot occur during a call to copy.

```
CLASS c (BITS b,c; STRING s);
POINTER(c) p,q;

p := new(c); q := new(c);
...
p.b := q.b; p.c := q.c; p.s := q.s;

# could be accomplished with: copy(q,p)
```

Example 1.73-2. Use of copy

1.74. $copyFile

```
PROCEDURE   $copyFile   (POINTER(file) src,dst;
                         OPTIONAL LONG INTEGER copyLen);
```

Table 1.74-1. $copyFile

$copyFile copies a portion of the file src to the file dst. It is an error if src is not open for input or dst is not open for output. No operation is performed if src equals dst.

The amount of data to be copied is measured in characters if src and dst are text files and in storage units if they are data files. It is computed as "IF NOT copyLen THEN <end-of-file position of src> - <current position of src> ELSE copyLen MIN (<end-of-file position of src> - <current position of src>)". $copyFile copies the amount of data to be copied, starting at the current positions of src and dst. It is more efficient than calling $storageUnitRead and $storageUnitWrite or $characterRead and $characterWrite to store the data to be copied in a temporary area.

If the files are text files and keepNul was not set in the open bits for src, null characters are discarded during the copy. In this case, the actual number of characters copied to dst is copyLen minus the number of null characters discarded. $copyFile of a text file is more efficient if keepNul was set when src was opened.

1.75. $coroutineExcpt

```
# system variable
STRING $coroutineExcpt;
```

Table 1.75-1. $coroutineExcpt

$coroutineExcpt is a predefined exception that is raised when the end of a coroutine's initializing procedure is reached without calling $resumeCoroutine.

1.76. cos

```
REAL
PROCEDURE    cos          (REAL r);

LONG REAL
PROCEDURE    cos          (LONG REAL r);
```

Table 1.76-1. cos (Generic)

cos returns the cosine of its argument, which is in radians.

1.77. cosh

```
REAL
PROCEDURE    cosh         (REAL r);

LONG REAL
PROCEDURE    cosh         (LONG REAL r);
```

Table 1.77-1. cosh (Generic)

cosh returns the hyperbolic cosine of its argument, which is in radians.

1.78. $cot

```
REAL
PROCEDURE    $cot          (REAL r);

LONG REAL
PROCEDURE    $cot          (LONG REAL r);
```

Table 1.78-1. $cot (Generic)

$cot returns the cotangent of its argument, which is in radians.

1.79. $cpuID

```
STRING
PROCEDURE    $cpuID;
```

Table 1.79-1. $cpuID

The CPU ID of the machine on which MAINSAIL is running is returned as a string, if it is available. If unavailable, the null string is returned. The CPU ID is unavailable on many operating systems; see the appropriate operating-system-specific user's guide for details.

On some operating systems, a configuration bit can be set to cause MAINSAIL to call a user-defined procedure for $cpuID. The user defines a procedure $alternateCpuID in an FLI module called $aCpuID; this mechanism is documented in detail in the operating-system-specific user's guides for the systems on which it is supported.

1.80. $cpuTime

```
LONG INTEGER
PROCEDURE    $cpuTime;
```

Table 1.80-1. $cpuTime

$cpuTime returns the number of CPU time units used by the current process. $cpuTimeResolution is the number of $cpuTime units per second and is operating-system-dependent. If the operating system has no notion of per-process CPU time, this procedure returns the elapsed (wall clock) time.

It is unspecified whether this measurement includes operating system overhead on behalf of the process. However, if the operating system provides measures of CPU time both including and excluding time spent by the operating system, the former value is returned.

Programs may not assume that the first call to $cpuTime returns the value 0L. In addition, it is possible that the CPU timer may wrap around during the execution. On most systems such wraparound is quite rare.

The accuracy of $cpuTime is not guaranteed to be better than one second.

1.81. $cpuTimeResolution

```
LONG INTEGER
<macro>      $cpuTimeResolution;
```

Table 1.81-1. $cpuTimeResolution

$cpuTimeResolution returns the operating-system-dependent number of CPU time units per second. The value returned by $cpuTime is measured in CPU time units.

1.82. cRead

```
$BUILTIN
INTEGER
PROCEDURE    cRead        (MODIFIES STRING s);

INTEGER
PROCEDURE    cRead        (POINTER(textFile) f);

$BUILTIN
INTEGER
PROCEDURE    cRead        (MODIFIES CHARADR c);

INTEGER
PROCEDURE    cRead        (POINTER(dataFile) f);
```

Table 1.82-1. cRead (Generic)

cRead returns the character code of either the current character in an input file, or the first character of a string, or the character addressed by a charadr. After the character is obtained, the file is positioned to the next character, or the character is removed from the string, or the charadr is displaced by one character. In the case of a data file, characters are stored one per character unit, as in a text file.

In the file forms, if the file is opened for PDF I/O, the character may be translated from the PDF to the host character set.

-1 is returned by the string and file forms if the string is "" or the end of the file has been reached. cRead from nullCharadr is undefined.

Example:

```
i := cRead(f)
```

reads the current character from textFile f and puts its code into i. f is then positioned to the next character.

1.83. create

```
COMPILETIME
BITS
<macro>      create;
```

Table 1.83-1. create

create is a bit that specifies that the output file is new, i.e., is to be created (completely replacing any existing file by the same name). It may be passed to $createUniqueFile, open, and $reOpen.

1.84. $createClassDscr

```
POINTER($classDscr)
PROCEDURE    $createClassDscr
                        (STRING className,fieldNames,
                            fieldTypes;
                         OPTIONAL BITS ctrlBits);
```

Table 1.84-1. $createClassDscr

$createClassDscr creates a class descriptor from the information given by the arguments, and returns a pointer to it. This class descriptor pointer can be used to allocate records with $createRecord. The fields of the class $classDscr are not documented. className is the name of the class (the effect is undefined if it is not a valid MAINSAIL identifier), fieldNames is the field names separated by eol's, and fieldTypes is the field data types separated by eol's, just like the corresponding values returned by $classInfo; the effect is undefined if more than one eol occurs between values, or at the start or end of either string. className and fieldNames are converted to upper case befor being stored in the class descriptor. className and fieldNames may be the null string, but fieldTypes must consist of a valid sequence of data type codes. Possible errors are an invalid fieldType and too many fieldNames. If an error occurs, nullPointer is returned and an error message is generated unless ctrlBits has the predefined bit errorOK set. errorOK is the only valid ctrlBits bit.

The class descriptor of a pointer is returned by $dscrPtr.

Example 1.84-2 shows how a new class descriptor can be created. In this case, a new class descriptor is created from an existing one by changing the first field to be of the integer data type.

```
STRING              s;
POINTER($classDscr) cd;
POINTER             p,q;
# assume p points to the class descriptor
$classInfo(p,className,fieldNames,fieldTypes);
read(fieldTypes,s); # discard first type code
cd := $createClassDscr(    # give it a different
    "INTEGER" & className, # name to distinguish it
    fieldNames,
        # assume that we know it has more than one field
    cvs(integerCode) & eol & fieldTypes);
q := $createRecord(cd); # allocate record of new class
```

Example 1.84-2. Use of $createClassDscr

1.85. $createCoroutine

```
POINTER($coroutine)
PROCEDURE   $createCoroutine
                        (POINTER initDataSec;
                         STRING initProcName;
                         OPTIONAL STRING coroutineName;
                         OPTIONAL INTEGER stackPages;
                         OPTIONAL BITS ctrlBits;
                         OPTIONAL POINTER($coroutine)
                             parent);
```

Table 1.85-1. $createCoroutine

$createCoroutine is used to create a new coroutine. If successful, it allocates a stack and a $coroutine record and returns a pointer to the record. The class $coroutine is described in Chapter 17 of part I of the "MAINSAIL Language Manual". The $coroutine pointer can be used to resume or kill the coroutine or access the fields of the $coroutine record. $createCoroutine does not start the coroutine executing; it just allocates and initializes the

coroutine. A subsequent call to $resumeCoroutine is used to transfer control to it and have it start executing.

initDataSec points to the data section to be used when the coroutine is first resumed. This pointer can be obtained from the MAINSAIL system procedure bind, new, or thisDataSection.

initProcName is the name of the procedure where execution is to start when the coroutine is first resumed. The procedure must be typeless and parameterless, and must be in a module for which initDataSec is a data section. MAINSAIL does not check whether the named procedure is in fact typeless and parameterless; violation of this rule results in undefined behavior.

coroutineName is the name (which is converted to upper case) to be given to the coroutine. It can be any string, except that it must not be the same as the name for any existing coroutine. If the argument is omitted, the next name from the sequence "coroutine1", "coroutine2", "coroutine3", ... is used.

The name of the root coroutine is "MAINSAIL".

stackPages is the number of pages to be allocated for the stack. If omitted, the default specified in the bootstrap is used. It is difficult to estimate how large the stack should be; if a coroutine involves a deeply nested set of procedure calls, or a deep level of recursion, then the default stack size may not be sufficient. If a stack does overflow, the program may fail in some undefined manner.

The only valid ctrlBits bit is errorOK, which indicates that if an error is detected (e.g., coroutineName is already assigned), then no error message is to be generated. In any case, an error causes nullPointer to be returned.

parent specifies the parent coroutine of the new coroutine. If parent is Zero, the coroutine is created as a child of the current coroutine.

1.86. $createRecord

```
POINTER
PROCEDURE    $createRecord
                          (POINTER p;
                           OPTIONAL POINTER($area) area);
```

Table 1.86-1. $createRecord

p is a pointer to a record, class descriptor, or data section. If p is invalid, an error message is generated and nullPointer returned. Otherwise, a record is allocated with the data fields described by the associated class descriptor, and a pointer to the record is returned. The record is allocated in area.

An example of the use of $createRecord is to allocate a record of the same class as another record of which the class is not known at compiletime, as shown in Example 1.86-2.

```
POINTER p,q;
... p gets set to a record of an unknown class ...
q := $createRecord(p);
```

Example 1.86-2. Use of $createRecord

1.87. $createUniqueFile

```
BOOLEAN
PROCEDURE    $createUniqueFile
                        (PRODUCES POINTER(textFile) f;
                         BITS openBits);

BOOLEAN
PROCEDURE    $createUniqueFile
                        (PRODUCES POINTER(dataFile) f;
                         BITS openBits);
```

Table 1.87-1. $createUniqueFile (Generic)

$createUniqueFile creates a uniquely named file. This is useful for temporary files created by several instances of the same program on the same file directory. If successful, it returns true, with f equal to the newly opened file (opened with openBits, as described for the system procedure "open", except that create is always set and prompt is ignored); otherwise, it returns false, and f is Zero.

Between the time a unique file name is found and the time that the file is created, it is possible for some other process to create a file with the same name. The probability of this happening is small but increases with the number of processes creating files on the same directory.

Unique file names are presently of the form "z<number>.tmp", where <number> is an integer of up to five digits. This format is subject to change.

1.88. $currentDirectory

```
STRING
PROCEDURE    $currentDirectory
                          (OPTIONAL BITS ctrlBits;
                           PRODUCES OPTIONAL STRING msg);
```

Table 1.88-1. $currentDirectory

If the operating system has a notion of a current working or connected directory or catalog, this procedure returns its name. If there is no such notion, or if it is not possible to determine the current working directory from a program, "" is returned. Consult the appropriate operating-system-specific MAINSAIL user's guide for details. If an error occurs, msg is set to a string describing the error, and if errorOK is not set in ctrlBits, an error message is issued. errorOK is the only valid bit in ctrlBits.

1.89. cva

```
$BUILTIN
ADDRESS
PROCEDURE    cva          (POINTER p);

$BUILTIN
ADDRESS
PROCEDURE    cva          (CHARADR c);

$BUILTIN
ADDRESS
PROCEDURE    cva          (LONG INTEGER l);
```

Table 1.89-1. cva (Generic) (continued)

```
$BUILTIN
ADDRESS
PROCEDURE    cva          (LONG BITS b);

$BUILTIN COMPILETIME
ADDRESS
PROCEDURE    cva          (ADDRESS a);
```

Table 1.89-1. cva (Generic) (end)

cva converts to address.

The long integer and long bits forms return the address represented by l or b. This is a machine-dependent conversion; the same integer or long bits value may not correspond to a valid address on every machine.

The charadr form returns the address of the storage unit containing the character addressed by c. The address may be rounded down to the next lower data-type-aligned address (i.e., "cvc(cva(c))" may return a charadr less than c). The bit pattern of the resulting address is not necessarily the same as that of c, since the formats of charadrs and addresses may differ. If c is nullCharadr, cva returns nullAddress.

The pointer form returns the address of the object referenced by p. If p is nullPointer, the result is nullAddress. The object referenced by p is subject to garbage collection; its address may change after the call to cva.

```
ADDRESS a;
POINTER p;

a := displace(cva(p),size(integerCode))
```

Example 1.89-2. Use of cva

Example 1.89-2 assigns to a the address that is displaced by the number of storage units in an integer from the address of the object referenced by p.

The address form of cva returns its argument.

1.90. cvAry

```
$BUILTIN
PROCEDURE    cvAry        (POINTER p;
                           PRODUCES ARRAY a);

$BUILTIN
PROCEDURE    cvAry        (LONG ARRAY a;
                           PRODUCES ARRAY b);
```

Table 1.90-1. cvAry (Generic)

cvAry converts a long array, or a pointer to an array element chunk, to an array.

The pointer form converts an element chunk (as produced by cvp) back into an array. The conversion is syntactic only; no actual copying or moving of elements is performed.

The long array form converts a long array into a short array. No actual copying or moving of elements is performed. The effect is undefined if a does not satisfy the short-array rule, as described in Section 7.9 of part I of the "MAINSAIL Language Manual".

A short array may be converted to a long array by means of an assignment statement; i.e., a short array is implicitly converted to a long array.

1.91. cvb

```
$BUILTIN COMPILETIME
BITS
PROCEDURE    cvb          (INTEGER i);

BITS
PROCEDURE    cvb          (STRING s;
                           OPTIONAL BITS radix);
```

Table 1.91-1. cvb (Generic) (continued)

```
$BUILTIN COMPILETIME
BITS
PROCEDURE    cvb          (LONG BITS b);

$BUILTIN COMPILETIME
BITS
PROCEDURE    cvb          (LONG INTEGER i);

$BUILTIN COMPILETIME
BITS
PROCEDURE    cvb          (BITS b);
```

Table 1.91-1. cvb (Generic) (end)

cvb converts to bits.

The (long) integer forms convert i to the bit pattern for its binary representation, truncated on the left if necessary; i.e., if i is nonnegative, "cvs(cvb(i),binary)" produces the string that is the standard mathematical base-two representation for i's value. If i is a negative constant, the compiler issues an error message; but this is subject to change.

The long bits form discards high-order bits if there are fewer bits in a bits than in a long bits on the target system.

The bits form returns its argument.

"b := cvb(s)" produces the same result as "r := s; read(r,b)", where r is a temporary string variable. The valid bits for radix are binary, hex, and octal (octal is assumed if radix is not specified); s is assumed to contain a value in the specified radix unless s contains an explicit radix specifier ("'B", "'H", "'O", or "'" (the latter is equivalent to "'O")), in which case the explicit radix specifier overrides the radix bit.

```
cvb(97)                          = '141
cvb('723L)                       = '723
cvb("Location '134 in error.")   = '134
```

Example 1.91-2. Use of cvb

1.92. $cvbo

```
BOOLEAN
PROCEDURE    $cvbo          (STRING s);
```

Table 1.92-1. $cvbo

$cvbo scans s for the string representation "TRUE" or "FALSE". Case is ignored. As soon as one of these string representations is found or there are no more characters in the source string, the scan stops. $cvbo returns true if the characters "TRUE" were found in the source string; otherwise, it returns false.

The characters "TRUE" or "FALSE" need not be preceded or followed by a blank, tab, or end of line.

1.93. cvc

```
$BUILTIN
CHARADR
PROCEDURE    cvc            (STRING s);

$BUILTIN
CHARADR
PROCEDURE    cvc            (ADDRESS a);

$BUILTIN
CHARADR
PROCEDURE    cvc            (LONG BITS bb);

$BUILTIN
CHARADR
PROCEDURE    cvc            (CHARADR c);
```

Table 1.93-1. cvc (Generic)

cvc converts to charadr.

The string form returns the charadr that addresses the first character of s. If s is in string space and a garbage collection occurs, the charadr that addresses the first character of s may change. If s is Zero, nullCharadr is returned.

The address form returns the charadr of the first character in the storage unit addressed by a. If a is nullAddress, nullCharadr is returned.

The long bits form returns a charadr with the same bit pattern as bb. The effect is undefined if bb is not a valid charadr bit pattern. The effect is undefined on machines where the size of a charadr and the size of a long bits differ.

The charadr form returns its argument.

```
STRING s;
CLASS stringDscr (CHARADR c; INTEGER l);
POINTER(stringDscr) p;
...
p := new(stringDscr);
p.c := cvc(s);
p.l := length(s);
```

Example 1.93-2. Use of cvc

In Example 1.93-2, the string variable s contains the same information as the record pointed to by p, though s's representation is machine-dependent and may be more compact than stringDscr records. s may also be subject to garbage collection if its characters are located in string space; the fields of the record are not updated if s is moved.

1.94. cvcs

```
STRING
PROCEDURE    cvcs          (INTEGER char;
                            OPTIONAL POINTER($area) area);
```

Table 1.94-1. cvcs

cvcs converts a character code to the string that consists of the single character with the code char. area specifies the destination area for the resulting string.

```
cvcs(65) = "A" # assuming the ASCII character set
               # in which 65 is the code for "A"
```

Example 1.94-2. Use of cvcs

1.95. cvi

```
$BUILTIN COMPILETIME
INTEGER
PROCEDURE   cvi          (BITS b);

INTEGER
PROCEDURE   cvi          (STRING s);

$BUILTIN
INTEGER
PROCEDURE   cvi          (REAL x);

$BUILTIN COMPILETIME
INTEGER
PROCEDURE   cvi          (LONG INTEGER i);

$BUILTIN COMPILETIME
INTEGER
PROCEDURE   cvi          (LONG BITS b);

$BUILTIN
INTEGER
PROCEDURE   cvi          (LONG REAL x);

$BUILTIN COMPILETIME
INTEGER
PROCEDURE   cvi          (INTEGER i);
```

Table 1.95-1. cvi (Generic)

cvi converts to integer.

The long integer form converts to a different internal format, if necessary. Overflow is not necessarily detected.

The integer form returns its argument.

The (long) real forms round x to the nearest integer. If x is exactly halfway between two integers (i.e., has a fraction of 0.5), the direction of rounding is unspecified.

The (long) bits forms convert b to the integer with the corresponding bit pattern, truncated on the left if necessary. If b is a constant bit pattern representing a negative number, the compiler issues an error message; but this is subject to change.

If s is a string, "i := cvi(s)" has the same result as "r := s; read(r,i)", where r is a temporary string variable.

See Table 1.169-3 for a table contrasting ceiling, cvi, floor, and truncate.

```
cvi(1876L)                    = 1876
cvi(10.4)                     = 10
cvi(-10.6)                    = -11
cvi(10.5)                     = 10 or 11 (unspecified)
cvi('234)                     = 156
cvi("There are 10 errors")    = 10
```

Example 1.95-2. Use of cvi

1.96. cvl

```
$BUILTIN
INTEGER
PROCEDURE    cvl          (INTEGER char);

COMPILETIME
STRING
PROCEDURE    cvl          (STRING s;
                           OPTIONAL POINTER($area) area);
```

Table 1.96-1. cvl (Generic)

cvl converts a character or string to lower case.

The string form converts a string to all lower case (i.e., the uppercase letters "A" through "Z" are converted to the corresponding lowercase letters "a" through "z", and other characters are unchanged).

If char is the code of an uppercase letter, then the integer form returns the code of the corresponding lowercase letter. Otherwise, "cvl(char)" is equal to char.

area specifies the destination area for the resulting string.

```
cvl("ABc") = "abc"
cvl("A2$") = "a2$"

cvl('M')   = 'm'
cvl('n')   = 'n'
```

Example 1.96-2. Use of cvl

1.97. cvlb

```
$BUILTIN COMPILETIME
LONG BITS
PROCEDURE   cvlb        (BITS b);

$BUILTIN COMPILETIME
LONG BITS
PROCEDURE   cvlb        (INTEGER i);

$BUILTIN COMPILETIME
LONG BITS
PROCEDURE   cvlb        (LONG INTEGER i);

LONG BITS
PROCEDURE   cvlb        (STRING s;
                         OPTIONAL BITS radix);
```

Table 1.97-1. cvlb (Generic) (continued)

```
$BUILTIN
LONG BITS
PROCEDURE    cvlb          (ADDRESS a);

$BUILTIN
LONG BITS
PROCEDURE    cvlb          (CHARADR c);

$BUILTIN COMPILETIME
LONG BITS
PROCEDURE    cvlb          (LONG BITS bb);
```

Table 1.97-1. cvlb (Generic) (end)

cvlb converts to long bits.

The (long) integer forms convert i to the bit pattern for its binary representation, sign-extending on the left if necessary. If i is a negative constant, the compiler issues an error message; but this is subject to change.

The bits form converts to a different format, if necessary. If there are more bits in the long bits data type than in the bits data type, the extra leftmost bits are cleared.

The charadr form returns a long bits with the same bit pattern as c. The effect is undefined on machines where the size of a charadr and the size of a long bits differ (at present, the sizes of charadr and long bits are the same on all MAINSAIL implementations).

The long bits form returns its argument.

If s is a string, "b := cvlb(s)" has the same result as "r := s; read(r,b)", where r is a string temporary variable. The valid bits for radix are binary, hex, and octal (octal is assumed if radix is not specified); s is assumed to contain a value in the specified radix unless s contains an explicit radix specifier ("'B", "'H", "'O", or "'" (the latter is equivalent to "'O")), in which case the explicit radix specifier overrides the radix bit.

The address form returns the bit pattern of a.

```
cvlb(123)                    =       '173L
cvlb('123)                   =       '123L
cvlb("The bits are '456")    =       '456L
```

Example 1.97-2. Use of cvlb

1.98. cvli

```
$BUILTIN COMPILETIME
LONG INTEGER
PROCEDURE   cvli        (INTEGER i);

LONG INTEGER
PROCEDURE   cvli        (STRING s);

$BUILTIN COMPILETIME
LONG INTEGER
PROCEDURE   cvli        (BITS b);

$BUILTIN
LONG INTEGER
PROCEDURE   cvli        (REAL x);

$BUILTIN COMPILETIME
LONG INTEGER
PROCEDURE   cvli        (LONG BITS b);

$BUILTIN
LONG INTEGER
PROCEDURE   cvli        (LONG REAL x);

$BUILTIN
LONG INTEGER
PROCEDURE   cvli        (ADDRESS a);
```

Table 1.98-1. cvli (Generic) (continued)

```
$BUILTIN COMPILETIME
LONG INTEGER
PROCEDURE    cvli          (LONG INTEGER ii);
```

Table 1.98-1. cvli (Generic) (end)

cvli converts to long integer.

The integer form converts to a different internal format, if necessary; the result has the same mathematical value as i.

The long integer form returns its argument.

The (long) real forms round x to the nearest long integer. If x is exactly halfway between two long integers (i.e., has a fraction of 0.5), the direction of rounding is unspecified.

The (long) bits forms convert b to the long integer with the corresponding bit pattern, zero-filling on the left if necessary. If b is a constant bit pattern representing a negative number, the compiler issues an error message; but this is subject to change.

If s is a string, "i := cvli(s)" has the same result as "r := s; read(r,i)", where r is a temporary string variable.

The address form returns the long integer corresponding to the bit pattern of a.

```
cvli(10)               = 10L
cvli(310.5)            = 310L or 311L (unspecified)
cvli(-310.5)           = -310L or -311L (unspecified)
cvli('130)             = 88L
cvli("Result: 1087")   = 1087L
cvli(NULLADDRESS)      = 0L
```

Example 1.98-2. Use of cvli

1.99. cvlr

```
$BUILTIN COMPILETIME
LONG REAL
PROCEDURE    cvlr          (INTEGER x);

$BUILTIN COMPILETIME
LONG REAL
PROCEDURE    cvlr          (REAL x);

LONG REAL
PROCEDURE    cvlr          (STRING s);

$BUILTIN COMPILETIME
LONG REAL
PROCEDURE    cvlr          (LONG INTEGER x);

$BUILTIN COMPILETIME
LONG REAL
PROCEDURE    cvlr          (LONG REAL x);
```

Table 1.99-1. cvlr (Generic)

cvlr converts to long real.

The (long) integer forms convert x to a long real with an equivalent mathematical value, provided that the value of x can be represented exactly as a long real value; otherwise, the result is rounded or truncated in an unspecified direction.

The real form converts to a different internal format, if necessary.

The long real form returns its argument.

If s is a string, "x := cvlr(s)" has the same effect as "r := s; read(r,x)", where r is a temporary string variable.

```
cvlr(1)                                = 1.L
cvlr("The value is 123.1234E-23") = 123.1234E-23L
```

Example 1.99-2. Use of cvlr

1.100. cvp

```
$BUILTIN
POINTER
PROCEDURE    cvp          (ADDRESS a);

$BUILTIN
POINTER
PROCEDURE    cvp          (LONG ARRAY a);

$BUILTIN COMPILETIME
POINTER
PROCEDURE    cvp          (POINTER p);
```

Table 1.100-1. cvp (Generic)

cvp converts to pointer.

The address form converts an address to a pointer. The result is undefined unless the address either is nullAddress or points to a chunk allocated by the MAINSAIL runtime system; in the former case the result is nullPointer.

The array form returns a pointer to a's element chunk. The form of an element chunk is not documented and is subject to change without notice. If a is nullArray, nullPointer is returned.

The pointer form returns its argument.

cvAry may be used to convert a pointer to an element chunk to an array.

```
INTEGER ARRAY(1 TO 8) a;
POINTER p;
...
p := cvp(a); # p points to a's element chunk
```

Example 1.100-2. Use of cvp

1.101. cvr

```
$BUILTIN COMPILETIME
REAL
PROCEDURE    cvr          (INTEGER x);

$BUILTIN COMPILETIME
REAL
PROCEDURE    cvr          (LONG REAL x);

REAL
PROCEDURE    cvr          (STRING s);

$BUILTIN COMPILETIME
REAL
PROCEDURE    cvr          (LONG INTEGER x);

$BUILTIN COMPILETIME
REAL
PROCEDURE    cvr          (REAL x);
```

Table 1.101-1. cvr (Generic)

cvr converts to real.

The (long) integer forms convert x to a real with an equivalent mathematical value, provided that the value of x can be represented exactly as a real value; otherwise, the result is rounded or truncated in an unspecified direction.

The long real form converts x to a real with an equivalent mathematical value, provided that the value of x can be represented exactly as a real value. If x has too great a magnitude to be represented as a real, the result is undefined; overflow is not necessarily detected. If x does not have too great a magnitude to be represented as a real, but cannot be represented exactly as a real, it is rounded or truncated in an unspecified direction.

The real form returns its argument.

If s is a string, "x := cvr(s)" has the same effect as "r := s; read(r,x)", where r is a temporary string variable.

1.102. cvs

```
STRING
PROCEDURE    cvs          (BOOLEAN v;
                           OPTIONAL POINTER($area) area);

COMPILETIME
STRING
PROCEDURE    cvs          (INTEGER i;
                           OPTIONAL POINTER($area) area);

STRING
PROCEDURE    cvs          (BITS b;
                           OPTIONAL BITS form;
                           OPTIONAL POINTER($area) area);

STRING
PROCEDURE    cvs          (REAL x;
                           OPTIONAL BITS form;
                           OPTIONAL POINTER($area) area);

COMPILETIME
STRING
PROCEDURE    cvs          (LONG INTEGER i;
                           OPTIONAL POINTER($area) area);
```

Table 1.102-1. cvs (Generic) (continued)

```
STRING
PROCEDURE    cvs          (LONG REAL x;
                           OPTIONAL BITS form;
                           OPTIONAL POINTER($area) area);

STRING
PROCEDURE    cvs          (LONG BITS b;
                           OPTIONAL BITS form;
                           OPTIONAL POINTER($area) area);

$BUILTIN COMPILETIME
STRING
PROCEDURE    cvs          (STRING s);
```

Table 1.102-1. cvs (Generic) (end)

cvs converts to string.

The boolean form of cvs returns "TRUE" if the boolean value is true and "FALSE" if it is false.

The (long) integer forms convert to the string that is the constant representation of i, except that the long integer form does not append "L".

The (long) real forms create the string that is the constant representation of x, except that the long real form does not append "L". The optional "form" argument gives the programmer some control over the format. The rightmost 8 bits of form, i.e., "cvi(form MSK 'HFF)", specify the number of digits to follow the decimal point. Roundoff or addition of trailing zeros is used to make the proper number of fraction digits. The two valid predefined bits constants that may be set in form are shown in Table 1.102-2.

Bit Name	Meaning
fixed	do not use an exponent
exponent	do use an exponent

Table 1.102-2. Valid Bits for form in the (Long) Real Form of cvs

The exponent is always the letter "E" followed by at least two digits (with a leading "0" if the magnitude of the exponent is less than 10). A nonnegative exponent is separated from "E" with

"+". All digits go after the decimal point in the exponent form if the rightmost 8 bits of form are not set.

If neither fixed nor exponent is specified (this is the default), an attempt is made to give the simplest representation; "form MSK 'HFF" is not used. The width of the resulting strings is not the same for every possible value.

When forming the default representation, roundoff occurs at the last significant digit if there are any fraction digits (this helps prevent values like ".9999999" for 1). No exponent appears in the result string if the decimal point would fall immediately before, within, or immediately after the significant digits; otherwise, an exponent is used.

If "form MSK 'HFF" is Zero, and exponent is specified, the number of fraction digits is taken to be the number of significant digits in x, minus one.

The (long) bits forms create the string that is the constant representation of b, except that the long bits form does not append "L". The form argument gives the programmer some control over the format. The four valid bits constants predefined for form are shown in Table 1.102-3. The octal format is used if neither binary nor hex is specified.

Bit Name	Meaning
binary	output in base 2
octal	output in base 8
hex	output in base 16
formatted	precede constant with single quote and base letter, as in program text

Table 1.102-3. Valid Bits for form in the (Long) Bits Forms of cvs

The string form returns its argument.

area specifies the destination area for the resulting string.

```
cvs(123)                          = "123"
cvs(456L)                         = "456"

cvs(123.456)                      = "123.456"
cvs(123.456,exponent!'4)          = ".1235E+03"
cvs(.123456E3,fixed!'4)           = "123.4560"

cvs('H123)                        = "443"
cvs('H123,formatted)              = "'O443"
cvs('H123,binary)                 = "100100011"
cvs('H123,formatted!binary)       = "'B100100011"
cvs('H123,hex)                    = "123"
cvs('H123,formatted!hex)          = "'H123"
```

Example 1.102-4. Use of cvs

1.103. cvu

```
$BUILTIN
INTEGER
PROCEDURE    cvu          (INTEGER char);

COMPILETIME
STRING
PROCEDURE    cvu          (STRING s;
                           OPTIONAL POINTER($area) area);
```

Table 1.103-1. cvu (Generic)

cvu converts a character or string to upper case.

The string form converts a string to all upper case (i.e., the lowercase letters "a" through "z" are converted to the corresponding uppercase letters "A" through "Z", and other characters are unchanged). area specifies the destination area for the resulting string.

If char is the code of a lowercase letter, then the integer form returns the code of the corresponding uppercase letter. Otherwise, "cvu(char)" is equal to char.

```
cvu("aBc") = "ABC"
cvu("a2$") = "A2$"
cvu('a')   = 'A'
cvu('M')   = 'M'

IF r and s are string variables, then:

    IF cvu(r) = cvu(s) THEN ...
    IF cvu(r) < cvu(s) THEN ...

are more efficiently written as:

    IF equ(r,u,upperCase) THEN ...
    IF compare(r,u,upperCase) < 0 THEN ...
```

Example 1.103-2. Use of cvu

1.104. cWrite

```
PROCEDURE   cWrite      (MODIFIES STRING s;
                         REPEATABLE INTEGER char);

PROCEDURE   cWrite      (POINTER(textFile) f;
                         REPEATABLE INTEGER char);

$BUILTIN
PROCEDURE   cWrite      (MODIFIES CHARADR c;
                         REPEATABLE INTEGER char);

PROCEDURE   cWrite      (MODIFIES STRING s;
                         POINTER($area) area;
                         REPEATABLE INTEGER char);

PROCEDURE   cWrite      (POINTER(dataFile) f;
                         REPEATABLE INTEGER char);
```

Table 1.104-1. cWrite (Generic)

cWrite writes a character to a file, a string, or a charadr.

The file forms put a character into the current character position in an output file f and update the current position to be the next position in the file. In the case of a data file, characters are stored one per character unit, as in a text file. If the file is opened for PDF I/O, the character may be translated to the PDF character set.

The string forms append the character to the end of s. In the area form, area specifies the destination area for the resulting string.

```
    cWrite(f,'a',' ',98)

has the same effect as "write(f,"a b")" if the character
set is ASCII, since 98 is ASCII for "b".  Portable
programs cannot assume the ASCII character set.

    cWrite(s,'a',' ',98)

has the same effect as s .& "a b".
```

Example 1.104-2. Use of the File and String Forms of cWrite

The charadr form puts the character with code char into the character location given by its charadr argument. The charadr is then positioned to the next character location. The effect is undefined if the charadr is nullCharadr. Example 1.104-3 writes the character "b" to the character location c.

```
CHARADR c;
...
cWrite(c,'b')
```

Example 1.104-3. Use of the Charadr Form of cWrite

1.105. $date

```
LONG INTEGER
PROCEDURE    $date        (OPTIONAL BITS ctrlBits);
```

Table 1.105-1. $date

$date returns the current date.

The valid predefined bits constants for ctrlBits are $localTime, $gmt, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the local date is returned, if available. If $gmt is specified, the GMT date is returned, if available.

If errorOK is specified and the date is not provided by the operating system and has not been set with $setTheDate, 0L is returned. If errorOK is not specified, the user is prompted for the date if not provided by the operating system and not set with $setTheDate.

$dateAndTime should be used if both date and time are to be obtained for the same instant. Otherwise, a wraparound can occur at midnight.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.106. $dateAndTime

```
PROCEDURE    $dateAndTime
                          (PRODUCES LONG INTEGER date,time;
                           OPTIONAL BITS ctrlBits);
```

Table 1.106-1. $dateAndTime

$dateAndTime returns the current date and the time. If possible, it obtains both the date and time at the same instant to guard against the wraparound at midnight that might occur if they were obtained individually.

The valid predefined bits constants for ctrlBits are $localTime, $gmt, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the local date and time are returned, if available. If $gmt is specified, the GMT date and time are returned, if available.

If errorOK is specified and the date or time is not available, 0L is returned for the unavailable value or values. If errorOK is not specified and the date and/or time is unavailable, the user is prompted for the date and/or time.

1.107. $dateAndTimeCompare

```
INTEGER
PROCEDURE    $dateAndTimeCompare
                          (LONG INTEGER d1,t1,d2,t2;
                           OPTIONAL BITS ctrlBits);
```

Table 1.107-1. $dateAndTimeCompare

$dateAndTimeCompare compares two dates and times.

If the two dates and times are absolute (GMT or local, but they need not be the same format; i.e., the start time may be GMT and the stop time local or vice versa, provided time conversion is available), $dateAndTimeCompare returns -1 if the time represented by the date d1 and the time t1 is before the time represented by the date d2 and the time t2, 0 if the two times are the same, and 1 if the second time is after the first. If the two dates and times are differences, $dateAndTimeCompare returns -1 if the interval represented by the number of days d1 and the number of seconds t1 is less than the interval represented by the number of days d2 and the number of seconds t2, 0 if the two intervals are the same, and 1 if the second interval is greater than the first.

If input values are invalid or of incompatible formats, -2 is returned and, unless the ctrlBits bit errorOK is set, an error message is issued.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.108. $dateAndTimeDifference

```
BOOLEAN
PROCEDURE    $dateAndTimeDifference
                        (LONG INTEGER startDate,startTime;
                         LONG INTEGER stopDate,stopTime;
                         PRODUCES LONG INTEGER
                             dateDif,timeDif;
                         OPTIONAL BITS ctrlBits);
```

Table 1.108-1. $dateAndTimeDifference

$dateAndTimeDifference produces the difference between two MAINSAIL date-time pairs.

startDate, startTime, stopDate, and stopTime must be valid MAINSAIL dates and times; they may be absolute (GMT or local) dates and times or date and time differences.

If the times are absolute (they need not be the same format; i.e., the start time may be GMT and the stop time local or vice versa, provided time conversion is available), the start time is startTime on startDate, and the stop time is stopTime on stopDate. If the times are time differences, the start time is simply subtracted from the stop time. The difference between the start time and the stop time is dateDif days plus timeDif seconds in time difference format (so that the magnitude of timeDif is always less than one day). dateDif and timeDif have the same signs, unless one is zero and the other is not. If the start time is before the stop time (if inputs are absolute) or less than the stop time (if inputs are differences), the difference is positive; if after or greater, it is negative.

$dateAndTimeDifference returns false if any of the argument dates and times is invalid or in an incompatible format, true otherwise.

The only valid ctrlBits bit is errorOK. Unless it is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual". Subtracting one date from another in the same format or subtracting one time of day from another in the same format may be accomplished by means of the long integer "-" operator.

1.109. $dateAndTimeToStr

```
STRING
PROCEDURE    $dateAndTimeToStr
                          (LONG INTEGER date,time;
                           OPTIONAL BITS ctrlBits;
                           PRODUCES OPTIONAL STRING
                               dateStr,timeStr,zoneStr;
                           OPTIONAL POINTER($area) strArea;
                           OPTIONAL LONG BITS ctrlBits2);
```

Table 1.109-1. $dateAndTimeToStr

$dateAndTimeToStr converts a MAINSAIL date and time to a string. date and time may be in local time, GMT time, or time difference format. By default, a local time string is returned.

If ctrlBits is not specified, the output format of $dateAndTimeToStr is the same as the output formats of $dateToStr and $timeToStr, separated by a space; e.g.:

```
$dateAndTimeToStr(d,t) = $dateToStr(d) & " " & $timeToStr(t)
```

except that if a time zone name appears in the string, $dateAndTimeToStr appends it only once to the end of the string, or if a plus or minus precedes a time difference string, it is included in the string only once.

dateStr, timeStr, and zoneStr are the date, time, and time zone name substrings, respectively, of the returned string. Some settings of ctrlBits may cause dateStr and timeStr to be different from those that would be returned by $dateToStr and $timeToStr, and these strings may not be correctly parsed if passed to $strToDate or $strToTime.

ctrlBits and ctrlBits2 bits valid for $dateToStr and $timeToStr are also valid for $dateAndTimeToStr, with the same effects. In addition, $localTime and $gmt may be specified in ctrlBits. If $localTime or $gmt is specified, a local time or GMT string is returned, respectively. The caveats described in Section 19.3 of part I of the "MAINSAIL Language Manual" regarding conversion between local time and GMT apply if date and time are in local format and $gmt is set or vice versa.

1.110. $dateFormat

```
BITS
PROCEDURE    $dateFormat (LONG INTEGER date);
```

Table 1.110-1. $dateFormat

$dateFormat returns $gmt if its argument is a GMT date, $localTime if its argument is a local date, $timeDifference if its argument is a date difference, or '0 if its argument is not a valid date value.

1.111. $dateToStr

```
STRING
PROCEDURE    $dateToStr  (LONG INTEGER date;
                          OPTIONAL BITS ctrlBits;
                          OPTIONAL POINTER($area) area;
                          OPTIONAL LONG BITS ctrlBits2);
```

Table 1.111-1. $dateToStr

$dateToStr produces a string from a MAINSAIL date, which may be an absolute (local or GMT) date or a date difference. area specifies the destination area for the resulting string.

The default format for $dateToStr if date is an absolute date is "<date> <month> <year> <B.C. if applicable> <GMT if applicable>", e.g., "4 July 1776", "15 March 44 B.C.", "29 February 1988 GMT". The string "A.D." is appended to the output string if the year is between 1 A.D. and 99 A.D., inclusive, so that the string is not mistaken for an abbreviation of a year in the current century. "GMT" is added to the string if date is in GMT format unless the ctrlBits bit $doNotIncludeTimeZone is set.

Date differences are converted by default to the format:

```
{-}<d> day{s}
```

The "-" is included if date is negative.

The null string is returned if an invalid input value is detected.

The predefined bits constants shown in Table 1.111-2 are valid in ctrlBits; those shown in Table 1.111-3 in ctrlBits2.

Bit	Meaning
$includeWeekday	The day of the week precedes the date and is separated from it by a comma and a space.
$reverseDateAndMonth	The month field precedes the date. The date is separated from the year by a comma and a blank unless $hyphenateDate is set.
$hyphenateDate	The month field is abbreviated to three letters, and only the last two digits of the year are given (unless $allYearDigits is specified in ctrlBits2). The date, month, and year fields are separated from each other by a hyphen (minus) character rather than a space.
$doNotIncludeTimeZone	If date is in GMT format, suppress the default addition of "GMT" to the returned string.
$includeTimeZone	If date is in local format, append the local time zone name, if known.
$briefFormat	If date is a date difference, convert it to "[+|-]<d>d", e.g., "+23d" (23 days), "-4d" (4 days, negative). A zero difference has a plus sign ("+0d").
errorOK	No error message is given if an invalid input value is detected.

Table 1.111-2. Predefined Bits Constants for $dateToStr ctrlBits

Example 1.111-4 shows the possible output string formats.

Bit	Meaning
$allYearDigits	The number of digits in the year is always exactly as many digits as required to represent the year, regardless of the value of ctrlBits.
$twoYearDigits	The year is always displayed with two digits (the last two digits of the year), padding or truncating as necessary, regardless of the value of ctrlBits.

Table 1.111-3. Predefined Bits Constants for $dateToStr ctrlBits2

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.112. $debugExec

$debugExec invokes MAINDEBUG from a program (if MAINDEBUG is installed). This feature is documented in detail in the "MAINDEBUG User's Guide".

1.113. $defaultArea

```
# system variable
POINTER($area) $defaultArea;
```

Table 1.113-1. $defaultArea

$defaultArea is the default area for all operations involving storage allocation or use; see Chapter 20 of part I of the "MAINSAIL Language Manual" for details.

```
For a sample date of 9 August 1982, Greenwich Mean Time,
the following string representations are possible:

$includeWeekday
      $hyphenateDate
            $reverseDateAndMonth
                  $doNotIncludeTimeZone
                        Resulting string

 clear clear clear clear "9 August 1982 GMT"
 clear clear clear set   "9 August 1982"
 clear clear set   clear "August 9, 1982 GMT"
 clear clear set   set   "August 9, 1982"
 clear set   clear clear "9-Aug-82 GMT"
 clear set   clear set   "9-Aug-82"
 clear set   set   clear "Aug-9-82 GMT"
 clear set   set   set   "Aug-9-82"
 set   clear clear clear "Monday, 9 August 1982 GMT"
 set   clear clear set   "Monday, 9 August 1982"
 set   clear set   clear "Monday, August 9, 1982 GMT"
 set   clear set   set   "Monday, August 9, 1982"
 set   set   clear clear "Monday, 9-Aug-82 GMT"
 set   set   clear set   "Monday, 9-Aug-82"
 set   set   set   clear "Monday, Aug-9-82 GMT"
 set   set   set   set   "Monday, Aug-9-82"

For a sample date of 8 March 1989, local time, the
following string representations are possible if
$dateToStr is called in the Pacific Standard Time zone
(PST):

$includeWeekday
      $hyphenateDate
            $reverseDateAndMonth
                  $includeTimeZone
                        Resulting string
```

Example 1.111-4. Sample $dateToStr Output Formats (continued)

```
clear clear clear clear "8 March 1989"
clear clear clear set   "8 March 1989 PST"
clear clear set   clear "March 8, 1989"
clear clear set   set   "March 8, 1989 PST"
clear set   clear clear "8-Mar-89"
clear set   clear set   "8-Mar-89 PST"
clear set   set   clear "Mar-8-89"
clear set   set   set   "Mar-8-89 PST"
set   clear clear clear "Wednesday, 8 March 1989"
set   clear clear set   "Wednesday, 8 March 1989 PST"
set   clear set   clear "Wednesday, March 8, 1989"
set   clear set   set   "Wednesday, March 8, 1989 PST"
set   set   clear clear "Wednesday, 8-Mar-89"
set   set   clear set   "Wednesday, 8-Mar-89 PST"
set   set   set   clear "Wednesday, Mar-8-89"
set   set   set   set   "Wednesday, Mar-8-89 PST"

For a time difference of 23L (23 days), the following
string representations are possible:

$briefFormat              Resulting string

 clear                    "23 days"
 set                      "+23d"
```

Example 1.111-4. Sample $dateToStr Output Formats (end)

1.114. delete

```
COMPILETIME
BITS
<macro>     delete;
```

Table 1.114-1. delete

delete is a bit that specifies that a file is to be deleted or a coroutine to be killed. It may be passed to close, $createUniqueFile, open, $resumeCoroutine, and $reOpen.

1.115. $delete

```
BOOLEAN
PROCEDURE    $delete      (STRING fileName;
                           OPTIONAL BITS ctrlBits);
```

Table 1.115-1. $delete

$delete deletes the file named fileName.

If the file cannot be deleted, $delete writes an error message to logFile and requests a new file name from cmdFile. If a blank line is read from cmdFile, $delete returns false. Otherwise, it again tries to delete the named file. If the bit errorOK is set in ctrlBits and the file cannot be deleted, $delete returns false without writing an error message or reading a new file name. If a file is successfully deleted, $delete returns true. If $useOriginalFileName is set in ctrlBits, no logical name lookup or application of searchpaths is done; the file name specified is used.

The effect of $delete is undefined if the specified file is open (by MAINSAIL or some other program).

1.116. $deregisterException

```
PROCEDURE    $deregisterException
                        (REPEATABLE STRING exceptionName);
```

Table 1.116-1. $deRegisterException

$deregisterException removes the exception denoted by exceptionName from the list of exceptions registered by means of $registerException. If no exception by that name is currently registered, an error message is issued. Distinctions between upper- and lowercase letters are ignored when comparing exceptionName to the strings denoting the registered exceptions.

1.117. $descendantKilledExcpt

```
# system variable
STRING $descendantKilledExcpt;
```

Table 1.117-1. $descendantKilledExcpt

$descendantKilledExcpt is a predefined exception that is raised in a coroutine's ancestors when the coroutine is killed to inform the coroutines that their descendant has died. It is described in more detail under $killCoroutine.

1.118. $devModBrk

```
COMPILETIME
INTEGER
<macro>      $devModBrk;
```

Table 1.118-1. $devModBrk

$devModBrk is the character used to separate a device module name from the rest of a file name, if the file name contains an explicit device module specification (see Section 18.11 of part I of the "MAINSAIL Language Manual"). Its value varies from operating system to operating system, but is usually '>'; consult the appropriate operating-system-dependent MAINSAIL user's guide.

1.119. $devModBrkStr

```
COMPILETIME
STRING
<macro>      $devModBrkStr;
```

Table 1.119-1. $devModBrkStr

$devModBrkStr is the string constant consisting of the single character $devModBrk; see Section 1.118.

```
To open a memory file using the standard device module MEM
(as described in the "MAINSAIL Utilities User's Guide"):

open(f,"MEM" & $devModBrkStr,create!input!output!random);
```

Example 1.119-2. Use of $devModBrkStr

1.120. $directory

```
BOOLEAN
PROCEDURE    $directory  (PRODUCES STRING ARRAY(1 TO *)
                              dir;
                         OPTIONAL STRING directoryName;
                         OPTIONAL BITS ctrlBits);
```

Table 1.120-1. $directory

$directory finds the list of files contained in the directory named directoryName and places one file name in each element of dir. directoryName is expected to be in the same format as returned by $currentDirectory. If it is the null string, $currentDirectory is used. $directory returns false if it find or read the directory named directoryName, true otherwise. The array dir is nullArray if there are no files in the directory. directoryName may contain a device module prefix, if appropriate.

If the operating system considers that subdirectories of a directory are files in that directory, the names of the subdirectories are also included in dir.

Valid ctrlBits are errorOK, $reportAllVersions, $fullPathNames, and $useOriginalFileName. If $fullPathNames is set, dir contains full path names, i.e., file names that may be used from any directory; normally it contains only relative path names, i.e., file names that may be used when directoryName is the current directory. The $reportAllVersions bit is ignored except on operating systems or device modules that maintain multiple numbered versions of files. On such systems or devices, only the most recent version of a file is included by default in the dir array, and the version number is not returned as part of a file name. If $reportAllVersions is

set, however, all existing versions of a file are included, and the version number is included in the file name. If $useOriginalFileName is set, no logical name lookup or application of searchpaths is done; directoryName is used as specified. errorOK suppresses any system-dependent error message that might otherwise occur.

1.121. $disassembleDate

```
PROCEDURE   $disassembleDate
                        (LONG INTEGER date;
                         PRODUCES INTEGER year;
                         PRODUCES OPTIONAL INTEGER
                             month,day;
                         OPTIONAL BITS ctrlBits);
```

Table 1.121-1. $disassembleDate

$disassembleDate returns the year, month, and day given an absolute (local or GMT) date.

If an illegal input value is detected, 0 is returned for year, month, and day.

The only valid ctrlBits bit is errorOK. Unless it is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.122. $disassembleDateAndTime

```
PROCEDURE   $disassembleDateAndTime
                       (LONG INTEGER date,time;
                        PRODUCES INTEGER year;
                        PRODUCES OPTIONAL INTEGER
                            month,day,hour,minute,second;
                        OPTIONAL BITS ctrlBits);
```

Table 1.122-1. $disassembleDateAndTime

$disassembleDateAndTime produces a year, month, day, hour, minute, and second given a MAINSAIL date and time. date and time may be in local time or GMT format. By default, local values are produced if date and time are in local format, and GMT values if date and time are in GMT format.

If invalid values are passed for date and time, the output values are all set to 0.

Valid ctrlBits bits are errorOK, $localTime, and $gmt. errorOK suppresses any error messages that might be printed. $localTime and $gmt cause the produced values to be local time and GMT values, respectively, regardless of the input value format. The caveats described in Section 19.3 of part I of the "MAINSAIL Language Manual" regarding conversion between local time and GMT apply if date and time are in local format and $gmt is set or vice versa.

1.123. $disassembleTime

```
PROCEDURE    $disassembleTime
                          (LONG INTEGER time;
                           PRODUCES INTEGER hour;
                           PRODUCES OPTIONAL INTEGER
                               minute,second;
                           OPTIONAL BITS ctrlBits);
```

Table 1.123-1. $disassembleTime

$disassembleTime returns the hour, minute, and second if given an absolute (local or GMT) time, or the number of hours, minutes, and seconds if given a time difference.

If an illegal input value is detected, -1 is returned for hour, minute, and second.

The only valid ctrlBits bit is errorOK. Unless it is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.124. discard

```
COMPILETIME
BITS
<macro>      discard;
```

Table 1.124-1. discard

discard is a bit that specifies that characters are to be discarded in various sorts of text scan. It may be passed to $removeBits, $removeDateAndTime, $removeInteger, $removeReal, and scan.

1.125. displace

```
$BUILTIN
ADDRESS
PROCEDURE    displace     (POINTER p;
                           INTEGER n);

$BUILTIN
ADDRESS
PROCEDURE    displace     (POINTER p;
                           LONG INTEGER n);

$BUILTIN
ADDRESS
PROCEDURE    displace     (ADDRESS a;
                           INTEGER n);

$BUILTIN
CHARADR
PROCEDURE    displace     (CHARADR c;
                           INTEGER n);
```

Table 1.125-1. displace (Generic) (continued)

```
$BUILTIN
ADDRESS
PROCEDURE   displace     (ADDRESS a;
                          LONG INTEGER n);

$BUILTIN
CHARADR
PROCEDURE   displace     (CHARADR c;
                          LONG INTEGER n);
```

Table 1.125-1. displace (Generic) (end)

"displace" computes an address or charadr as a displacement from a pointer, address, or charadr.

The address forms return an address that is displaced n storage units from a.

The pointer forms return an address that is displaced n storage units from p (i.e., "displace(p,n)" is equivalent to "displace(cva(p),n)").

The charadr forms return a charadr that is displaced n characters from c.

n may be positive or negative. If the resulting address or charadr would be less than the lowest representable address or charadr, or greater than the highest representable address or charadr, the result is undefined; i.e., a program may not assume that addresses "wrap around" at Zero.

A garbage collection cannot occur during a call to displace.

```
INTEGER i;
STRING s;
CHARADR c;

s := "xyz";
c := displace(cvc(s),2);
i := cLoad(c); # i is 'z'
```

Example 1.125-2. Use of displace

1.126. displacement

```
$BUILTIN
INTEGER
PROCEDURE   displacement
                          (ADDRESS a,b);

$BUILTIN
INTEGER
PROCEDURE   displacement
                          (CHARADR a,b);
```

Table 1.126-1. displacement (Generic)

"displacement" computes the distance between two addresses or charadrs.

The address form returns the number of storage units from address a to address b.

The charadr form returns the number of character units from charadr a to charadr b.

If a is beyond b, the result is negative. If there is a possibility that the distance is larger than can be represented as an integer, lDisplacement should be used; the use of displacement when the distance between a and b is larger than can be represented as an integer is undefined.

```
INTEGER i;
ADDRESS a,b;
INTEGER ARRAY(1 TO 100) ary;
...
IF NOT ary THEN new(ary);
# assume a is the address of the first element of ary
b := a;
FOR i := 1 UPTO 100 DO read(b,ary[i]);
i := displacement(a,b);

# i = 100 * size(integerCode)
```

Example 1.126-2. Use of displacement

1.127. dispose

```
PROCEDURE   dispose     (MODIFIES REPEATABLE POINTER p);

PROCEDURE   dispose     (MODIFIES REPEATABLE ARRAY a);

PROCEDURE   dispose     (MODIFIES REPEATABLE LONG ARRAY
                            a);

PROCEDURE   dispose     (REPEATABLE MODULE m);

PROCEDURE   dispose     (REPEATABLE STRING s);
```

Table 1.127-1. dispose (Generic)

"dispose" frees the memory occupied by a record, data section, array, or module so that the storage can be immediately reallocated for some other purpose. This freeing of memory is referred to as "disposing" the data structure that occupies the memory.

The pointer form may be used to dispose a record; the array form disposes an array.

The pointer form may also be used to dispose a data section. If p points to a data section, the final procedure (if any) of the corresponding module is executed before the memory is freed. If p points at the bound data section, it is unbound; i.e., the effect is the same as if unBind had been called for the module.

The module and string forms dispose all of the data sections of the module m or the module named by s. As with the pointer form, the final procedure of the module is executed before the memory associated with each data section is freed. In addition, any runtime system data structures associated with the module are disposed, and the control section of the module is released; i.e., the association of the module with the control section is broken, so that the next time the module is bound or newed, the standard search procedure for modules is followed.

A disposed record, data section, or array must not later be referenced. The argument to the pointer and array forms is modified to Zero to prevent it from being used for future references. If there are other pointers to the disposed object, the results of using them are undefined. Such a bug can be exceedingly difficult to track.

If p, a, or s is Zero, dispose does nothing.

The storage for any data sections, records, and arrays that become inaccessible is eventually reclaimed by the garbage collector. Explicit use of dispose is a more efficient alternative, but may lead to bugs that are difficult to track if a pointer to the disposed object is accidentally used.

1.128. $disposeArea

```
PROCEDURE    $disposeArea
                          (MODIFIES REPEATABLE
                               POINTER($area) area);
```

Table 1.128-1. $disposeArea

$disposeArea disposes area. All memory occupied by chunks or string text in area is freed; subsequent reference to the chunks or text has undefined effects.

1.129. $disposeDataSecsInArea

```
LONG INTEGER
PROCEDURE    $disposeDataSecsInArea
                          (POINTER($area) area);
```

Table 1.129-1. $disposeDataSecsInArea

$disposeDataSecsInArea disposes all data sections in area. It returns the number of data sections disposed. If the final procedure of any of the disposed data sections creates new data sections in the area, the procedure may fail by issuing a fatal error message.

1.130. $disposedDataSecExcpt

```
# system variable
STRING $disposedDataSecExcpt;
```

Table 1.130-1. $disposedDataSecExcpt

$disposedDataSecExcpt is a predefined exception that is raised when a procedure attempts to return to a procedure in an instance of a module that has been disposed.

1.131. $doNotClear

```
COMPILETIME
BITS
<macro>      $doNotClear;
```

Table 1.131-1. $doNotClear

$doNotClear is a bit that specifies that allocated memory is not to be initially cleared. It may be passed to newPage.

1.132. $doNotIncludeTimeZone

```
COMPILETIME
BITS
<macro>      $doNotIncludeTimeZone;
```

Table 1.132-1. $doNotIncludeTimeZone

$doNotIncludeTimeZone is a bit that specifies that a time zone string is not to be included in an output date or time string. It may be passed to $dateAndTimeToStr, $dateToStr, and $timeToStr.

1.133. $doNotMatch

```
COMPILETIME
BITS
<macro>      $doNotMatch;
```

Table 1.133-1. $doNotMatch

$doNotMatch is a bit that specifies that an exception is ignored when errMsg searches for a registered exception to raise. It may be passed to $registerException.

1.134. $doNotRaise

```
COMPILETIME
BITS
<macro>      $doNotRaise;
```

Table 1.134-1. $doNotRaise

$doNotRaise is a bit that specifies that an exception is not to be raised. It may be passed to errMsg.

1.135. $dscrPtr

```
POINTER($classDscr)
<macro>      $dscrPtr     (POINTER p);
```

Table 1.135-1. $dscrPtr

$dscrPtr returns the class descriptor associated with p. A unique class descriptor, of the class $classDscr, exists for each class in memory. The fields of $classDscr are not documented.

If p is nullPointer or dangling, the effect is undefined.

1.136. DSP

The compiletime pseudo-procedure "DSP" returns an integer displacement to a field of a class or module; it is described in detail in Section 14.8 of part I of the "MAINSAIL Language Manual".

1.137. $dup

```
$ALWAYSINLINE COMPILETIME
STRING
PROCEDURE    $dup          (STRING s;
                            INTEGER n;
                            OPTIONAL POINTER($area) area);
```

Table 1.137-1. $dup

$dup returns s concatenated with itself n times (the new text is placed in area if area is specified). For example:

```
$dup("-",50)
```

returns a string of 50 dashes. fldWrite may be used to pad a string to a specified length; see Section 1.168.

If s and n are constants, and area is omitted, $dup is computed at compiletime.

1.138. enterLogicalName

```
PROCEDURE    enterLogicalName
                           (STRING logicalName,trueName);
```

Table 1.138-1. enterLogicalName

enterLogicalName establishes or removes a logical file name. If trueName is not the null string, then after the call to enterLogicalName, whenever logicalName is passed as a file name to the procedure open, the file named trueName is actually opened. If trueName is the null string, any logical file name association for logicalName is removed.

For example, after executing:

```
enterLogicalName("parameters","src:parms.txt")
```

the call to open:

```
open(f,"parameters",...)
```

attempts to open the file "src:parms.txt" instead of the file "parameters".

1.139. eof

```
BOOLEAN
PROCEDURE   eof          (POINTER(file) f);
```

Table 1.139-1. eof

eof (end-of-file) returns true when the file pointer is positioned at or beyond the end of the file f. The preferred method of determining the end-of-file position of a file is $gotValue (see Section 1.184).

The programmer is advised not to rely on eof or $gotValue, but rather to design files that indicate their own end-of-file, e.g., by some special data value; some operating systems do not permit MAINSAIL to ascertain the end-of-file position exactly.

1.140. eol

```
COMPILETIME
STRING
<macro>      eol;
```

Table 1.140-1. eol

eol is the one-character end-of-line string.

1.141. eop

```
COMPILETIME
STRING
<macro>      eop;
```

Table 1.141-1. eop

eop is the string consisting of the end-of-page character.

1.142. equ

```
COMPILETIME
BOOLEAN
PROCEDURE    equ          (STRING r,s;
                           OPTIONAL BITS ctrlBits);
```

Table 1.142-1. equ

equ checks the equality of its two string arguments.

"equ(r,s)" is equivalent to "r = s"; i.e., it returns true if the strings have the same value and false if not.

A single valid bits constant, upperCase, is defined for use with ctrlBits. If present, it means ignore upper/lower case distinctions when checking the arguments for equality. The effect is as if both arguments were converted to upper case before the check. This option is more efficient than first converting to upper case with cvu or scan and then checking for equality. See Example 1.142-2.

```
    IF equ(r,s,upperCase) THEN ...

has the same effect as (but is more efficient than)

    IF cvu(r) = cvu(s) THEN ...
```

Example 1.142-2. Use of equ

1.143. errMsg

```
BOOLEAN
PROCEDURE   errMsg      (OPTIONAL STRING msg,val;
                         OPTIONAL BITS ctrlBits);
```

Table 1.143-1. errMsg

errMsg raises an exception. If no handler handles the exception, a message is written to logFile and a response obtained from cmdFile.

If $doNotRaise is not set in ctrlBits, errMsg raises the predefined exception denoted by $systemExcpt by calling $raise with the arguments shown in Table 1.143-2. In the case of fatal errors, it sets the $cannotReturn bit in the call to $raise; otherwise, if no handler handles the exception, control returns to errMsg and errMsg writes the message specified by its arguments.

errMsg returns false if a handler handling the $systemExcpt exception calls $raiseReturn, in which case no message is written to logFile and no response read from cmdFile. Otherwise, the message is written, and errMsg returns true.

```
$raise($systemExcpt,msg,val,NULLPOINTER,
        $returnIfNoHandler!ctrlBits)
```

Table 1.143-2. Arguments to $raise When Called from errMsg

When the message is written, "ERROR:" is written to logFile, followed by the string msg. If val is not "" it is written after the message, preceded by a blank. Finally, "Error response:" is written on a new line to signify that a response is requested; valid responses appear in Table 1.143-4. Other responses may be shown if the appropriate exceptions have been registered by means of $registerException. Section 16.9 of part I of the "MAINSAIL Language Manual" explains how errMsg responses may be abbreviated.

Valid predefined bits constants for ctrlBits are shown in Table 1.143-3.

Bit	Effect
$doNotRaise	Do not call $raise; just write the message.
warning	Write "WARNING:" instead of "ERROR:" before the message. Do not get a response.
fatal	Write "FATAL:" instead of "ERROR:" before the message. Do not allow execution to continue.
noResponse	Do not get a response.
msgMe	Write the name of the module that called errMsg and the decimal offset within the module's control section at which the call to errMsg occurred. If the invoking coroutine is not the root coroutine "MAINSAIL", its name is written as well.
msgMyCaller	Write the name of the module that called the module that called errMsg and the decimal offset within the module's control section at which the call to the procedure that called errMsg occurred. If the invoking coroutine is not the root coroutine "MAINSAIL", its name is written as well.

Table 1.143-3. Predefined Bits Constants for errMsg ctrlBits

<eol>	Continue execution (invalid if fatal error).
QUIT	Exit MAINSAIL.
MAINSAIL: Abort program	Exit the current program.
EXECUTE m/f	Execute module m or module in file f.
CALLS c	Show (on logFile) a list of the calls (most recent first) in coroutine c. If c is omitted, the call chain for the current coroutine is shown, i.e., the call sequence that led to the call to errMsg. For each procedure call made, the module in which the call was made is shown, followed by the decimal offset of the call in the module.
DEBUG	Enter MAINDEBUG, the MAINSAIL debugger.
@	Enter MAINEDIT, the MAINSAIL editor (no effect if MAINEDIT is already running).
??	Rewrite the error message and the "Error response:" prompt.
?	Show a list of valid responses.

Table 1.143-4. Valid Responses to "Error response:" Prompt

1.144. errorOK

COMPILETIME	
BITS	
<macro>	errorOK

Table 1.144-1. errorOK

errorOK is a bit that indicates that an error message is to be suppressed. It may be passed to most system procedures that accept a controlling bits parameter.

1.145. $exceptionBits

```
BITS
<macro>     $exceptionBits;
```

Table 1.145-1. $exceptionBits

$exceptionBits returns information about the current exception. All bits that can be specified to $raise may be tested in $exceptionBits; see Table 1.282-2. In addition, the bits warning, fatal, and noResponse may be tested; these bits are set if $raise was called from errMsg and the bits were set in the call to errMsg. If there is no current exception, $exceptionBits returns Zero.

1.146. $exceptionCoroutine

```
POINTER($coroutine)
<macro>     $exceptionCoroutine;
```

Table 1.146-1. $exceptionCoroutine

$exceptionCoroutine returns a pointer to the raiser coroutine for the current exception (different from the raisee coroutine only if the exceptionCoroutine argument to $raise denoted a coroutine other than the raiser coroutine). If there is no current exception, $exceptionCoroutine returns nullPointer. In the case of $abortProcedureExcpt, $exceptionCoroutine returns the coroutine in which the original exception occurred.

1.147. $exceptionName

```
STRING
<macro>     $exceptionName;
```

Table 1.147-1. $exceptionName

$exceptionName returns the name of the current exception. If there is no current exception, $exceptionName returns the null string.

1.148. $exceptionPointerArg

```
POINTER
<macro>     $exceptionPointerArg;
```

Table 1.148-1. $exceptionPointerArg

$exceptionPointerArg returns the value that was passed as the argument exceptionPointerArg to the system procedure $raise when the current exception was raised. If there is no current exception, $exceptionPointerArg returns nullPointer.

1.149. $exceptionStringArg1

```
STRING
<macro>     $exceptionStringArg1;
```

Table 1.149-1. $exceptionStringArg1

$exceptionStringArg1 returns the value that was passed as the argument exceptionStringArg1 to the system procedure $raise when the current exception was raised. If there is no current exception, $exceptionStringArg1 returns the null string.

1.150. $exceptionStringArg2

```
STRING
<macro>      $exceptionStringArg2;
```

Table 1.150-1. $exceptionStringArg2

$exceptionStringArg2 returns the value that was passed as the argument exceptionStringArg2 to the system procedure $raise when the current exception was raised. If there is no current exception, $exceptionStringArg2 returns the null string.

1.151. $excludeSeconds

```
COMPILETIME
BITS
<macro>      $excludeSeconds;
```

Table 1.151-1. $excludeSeconds

$excludeSeconds is a bit that specifies that seconds are not to be included in the output string. It may be passed to $dateAndTimeToStr and $timeToStr.

1.152. $executeIntlibCommands

INTLIB can be controlled from a user program by calling the procedure $executeIntlibCommands. This feature is documented in detail under INTLIB in the "MAINSAIL Utilities User's Guide".

1.153. $executeModlibCommands

MODLIB can be controlled from a user program by calling the procedure $executeModlibCommands. This feature is documented in detail under MODLIB in the "MAINSAIL Utilities User's Guide".

1.154. $executeStampCommands

STAMP can be controlled from a user program by calling the procedure $executeStampCommands. This feature is documented in detail under STAMP in the "MAINSAIL Utilities User's Guide".

1.155. exit

```
PROCEDURE    exit          (OPTIONAL STRING msg);
```

Table 1.155-1. exit

"exit" writes msg (if non-Zero) to logFile. The final procedures associated with all data sections are executed (in an unspecified order), and any open files and libraries are closed. MAINSAIL then returns control to the operating system from which it was invoked.

fastExit provides a quicker (less orderly) exit.

1.156. exp

```
REAL
PROCEDURE    exp           (REAL x);

LONG REAL
PROCEDURE    exp           (LONG REAL x);
```

Table 1.156-1. exp (Generic)

exp returns the exponential e to the xth power, where e is the base of the natural logarithms.

1.157. exponent

```
COMPILETIME
BITS
<macro>     exponent;
```

Table 1.157-1. exponent

exponent is a bit that specifies that the output string is to include an exponent. It may be passed to cvs.

1.158. $exponentExcpt

```
# system variable
STRING $exponentExcpt;
```

Table 1.158-1. $exponentExcpt

$exponentExcpt is a predefined exception that is raised when a (long) integer is raised to power less than zero.

1.159. fastExit

```
PROCEDURE   fastExit    (OPTIONAL STRING msg);
```

Table 1.159-1. fastExit

fastExit writes msg (if non-Zero) to logFile, then terminates the MAINSAIL session; i.e., MAINSAIL returns control to the operating system from which it was invoked.

exit provides a more orderly exit.

1.160. fatal

```
COMPILETIME
BITS
<macro>     fatal;
```

Table 1.160-1. fatal

fatal is a bit that specifies that an error message is fatal. It may be passed to errMsg and tested in $exceptionBits. It is set in a call to $raise made from errMsg if the fatal bit is set in the call to errMsg.

1.161. $fieldInfo

```
BOOLEAN
PROCEDURE   $fieldInfo   (POINTER p;
                          STRING fieldName;
                          PRODUCES OPTIONAL INTEGER
                              type,dspl);
```

Table 1.161-1. $fieldInfo

p is a pointer to a record, class descriptor, or data section. fieldName is the name of a field. If p is invalid (e.g., nullPointer), false is returned. Otherwise, the field names in the associated class descriptor are searched, and if fieldName is found (comparison is caseless), type is set to the data type code for the field (e.g., integerCode), dspl is set to the displacement in storage units from the start of the record (first field) to the start of the named field, and true is returned. If there is no field by the name fieldName, false is returned.

$fieldInfo can be used to get or change the value of a field given a pointer and a string with the name of the field, as illustrated in Example 1.161-2.

$fieldInfo makes a linear search of the string that contains the field names to look for the argument fieldName. If there are many fields, and the lookup is done often, it is more efficient to call $classInfo once to get all the required information, and store this information in a more rapidly accessible data structure (e.g., a hash table based on field name).

```
ttyWrite(s," = ");
IF $fieldInfo(p,s,type,dspl) THEN
    CASE type OFB
        [booleanCode]
            ttyWrite(IF boLoad(cva(p),dspl) THEN "TRUE"
                     EL "FALSE");
        [integerCode]
            ttyWrite(iLoad(cva(p),dspl));
        [longIntegerCode]
            ttyWrite(liload(cva(p),dspl));
        ...
        [pointerCode]
            ttyWrite("'",lbload(cva(p),dspl));
        END
EL ttyWrite("<invalid pointer or field name>");
ttyWrite(eol)
```

Example 1.161-2. Use of $fieldInfo

1.162. $fileInfo

```
CLASS $fileInfoCls (
    STRING $fullPathName;
    LONG INTEGER $OSDSize;
    LONG INTEGER $createDate,$createTime,
        $modifyDate,$modifyTime;
);

POINTER($fileInfoCls)
PROCEDURE   $fileInfo   (POINTER(file) f;
                         OPTIONAL BITS ctrlBits;
                         OPTIONAL POINTER($fileInfoCls)
                             fi);
```

Table 1.162-1. $fileInfo (Generic) and $fileInfoCls (continued)

```
POINTER($fileInfoCls)
PROCEDURE    $fileInfo    (STRING fileName;
                           OPTIONAL BITS ctrlBits;
                           OPTIONAL POINTER($fileInfoCls)
                               fi);
```

Table 1.162-1. $fileInfo (Generic) and $fileInfoCls (end)

$fileInfo returns information about a file, given its name or an open file pointer to it. NullPointer is returned if the requested file cannot be found or if no information about the file can be obtained. Fields unavailable from the operating system are Zero, except that $OSDsize is -1L if unavailable, since 0L is a possible file length. The meanings of the fields of $fileInfoCls are as shown in Table 1.162-2.

Field	Meaning
$fullPathName	A full, unambiguous file name for the file. Suitable for passing to the system procedure open. On systems that permit it, the correspondence between full path names and files is one-to-one.
$OSDSize	System-dependent file size.
$createDate, $createTime	Time file was created, if available.
$modifyDate, $modifyTime	Time file was last modified, if available.

Table 1.162-2. $fileInfoCls Fields

If the fileName argument to $fileInfo includes a device prefix, a device prefix is included in $fullPathName; otherwise, no device prefix appears in $fullPathName. If the device prefix in fileName specifies the format of a disk file (e.g., "BS", "VAR", "FIX"), the device prefix in $fullPathName is changed if the actual format of the file is different from that specified; e.g., if fileName is "bs>foo", the prefix for $fullPathName may be "var" if foo is actually a VAR-format file.

If fi is Zero, a new record is allocated and a pointer to it returned (if $fileInfo is successful; otherwise, $fileInfo returns nullPointer). If fi is non-Zero, the record it points to is filled in and returned if $fileInfo is successful.

The exact value of $OSDSize is dependent on the operating system. It does not have any predictable relationship to the MAINSAIL end-of-file position. $OSDSize is intended to be used by programs that need to organize a list of files in approximate order of size.

The valid ctrlBits bits are errorOK, $useOriginalFileName, and $gmt. errorOK suppresses any system-dependent error message that might otherwise occur. If $useOriginalFileName is set, no logical name lookup or application of searchpaths is done; fileName is used as specified. If $gmt is set, $createDate, $createTime, $modifyDate, and $modifyTime are returned in GMT format instead of local time format, if available.

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

In the present release, a long bits field, $fileAttr, is included in $fileInfoCls. The following bits are used in $fileAttr:

```
$bsFormat
$fixFormat
$varFormat
$isDirectory
```

At most one of $bsFormat, $fixFormat, $varFormat, and $isDirectory can be set in $fileAttr. $bsFormat, $fixFormat, and $varFormat indicate byte-stream, fixed-length-record, and variable-length-record text and data files, respectively (for operating systems that support record-structured files, the MAINSAIL facilities used to access them are described in the appropriate system-specific documentation). $isDirectory is set if the file is a directory. If the attributes of the file cannot be determined (or if $fileAttr has not yet been implemented on the host operating system), none of these bits is set in $fileAttr.

1.163. $findArea

```
POINTER($area)
PROCEDURE   $findArea   (STRING title);
```

Table 1.163-1. $findArea

$findArea returns the area with title title. If more than one such area exists, it is not specified which area is returned. If no such area exists, it returns nullPointer.

1.164. $findCoroutine

```
POINTER($coroutine)
PROCEDURE    $findCoroutine
                          (STRING coroutineName);
```

Table 1.164-1. $findCoroutine

$findCoroutine returns a pointer to the $coroutine record for the coroutine with the indicated name (case is ignored). NullPointer is returned if there is no such coroutine.

1.165. first

```
$BUILTIN COMPILETIME
INTEGER
PROCEDURE    first        (STRING s);
```

Table 1.165-1. first

first returns the character code for the first character of a string.

If s is "", -1 is returned. -1 is not a valid character code.

```
first("abc") = 'a' = 97 # assuming the ASCII character set
first("") = -1
```

Example 1.165-2. Use of first

1.166. fixed

```
COMPILETIME
BITS
<macro>      fixed;
```

Table 1.166-1. fixed

fixed is a bit that specifies that no exponent is to appear in the output string. It may be passed to cvs.

1.167. fldRead

```
STRING
PROCEDURE    fldRead      (POINTER(textFile) f;
                           INTEGER width;
                           OPTIONAL POINTER($area) area);

STRING
PROCEDURE    fldRead      (POINTER(dataFile) f;
                           INTEGER width;
                           OPTIONAL POINTER($area) area);

STRING
PROCEDURE    fldRead      (MODIFIES STRING s;
                           INTEGER width);
```

Table 1.167-1. fldRead (Generic)

fldRead reads a field from an input file or a string. A field is a string with the specified width.

If width is less than one, "" is returned.

If requested characters lie beyond the end of the file or string, only those characters (if any) obtained before the end of the file or string are returned.

The textFile form may be called for an unbuffered file (a file opened with the $unbuffered bit set). fldRead does not filter out null characters from an unbuffered file; i.e., it acts as if the file had been opened with the keepNul bit set. If the file is opened for PDF I/O, the characters may be translated from the PDF to the host character set.

For example, "s := fldRead(inFile,15)" reads the next 15 characters from inFile. If only 10 characters remain in the file, then the string consisting of those 10 characters is returned.

In the textFile and dataFile forms, area specifies the destination area for the resulting string.

Characters in a data file are stored as described in Sections 1.82 and 1.104.

1.168. fldWrite

```
PROCEDURE   fldWrite    (POINTER(textFile) dst;
                         STRING s;
                         INTEGER w,fillChar);

PROCEDURE   fldWrite    (POINTER(dataFile) dst;
                         STRING s;
                         INTEGER w,fillChar);

PROCEDURE   fldWrite    (MODIFIES STRING dst;
                         STRING s;
                         INTEGER w,fillChar;
                         OPTIONAL POINTER($area) area);
```

Table 1.168-1. fldWrite (Generic)

fldWrite writes a string of the specified width w to a file or string destination dst. The string written is composed of s and enough fill characters to make the length of the string written equal to w. The character code of the fill character is given by fillChar.

Normally, fill characters are put before s, so that when fldWrite is used to produce columns, the right margin of the column is aligned. For example, a fillChar of ' ' results in s right-justified in a field of blanks.

Use the negative of the desired fill character to have fill characters written after s.

If s exceeds the field width w, a string consisting of w asterisks (the "*" character) is written.

In the string form, area specifies the destination area for the resulting string. In the file forms, if the file is opened for PDF I/O, the characters may be translated from the host to the PDF character set.

Characters in a data file are stored as described in Sections 1.82 and 1.104.

$dup may be used to create a string by concatenating a given string several times; see Section 1.137.

```
If s = "ABCDEF", then

    fldWrite(f,s,10,' ')

writes "    ABCDEF" to f.

    fldWrite(f,s,12,- '.')

writes "ABCDEF......" to f.
```

Example 1.168-2. Use of fldWrite

1.169. floor

```
INTEGER
PROCEDURE   floor       (REAL x);

LONG INTEGER
PROCEDURE   floor       (LONG REAL x);
```

Table 1.169-1. floor (Generic)

"floor" returns the largest (long) integer less than or equal to x.

Table 1.169-3 shows the directions on the real number line in which the conversion procedures from (long) real to (long) integer "move" their arguments (there is no movement if the argument is an integral value).

```
floor(10.5)  = 10
floor(-10.5) = -11
```

Example 1.169-2. Use of floor

```
floor          <------        |        <------
                              |
ceiling        ------>        |        ------>
                              |
truncate       ------>        |        <------
                              |
cvi            <----->        |        <----->
                              |
------------------------------+------------------------------
              negative        0        positive

floor(-10.5)     = -11          floor(10.5)     = 10
ceiling(-10.5)   = -10          ceiling(10.5)   = 11
truncate(-10.5)  = -10          truncate(10.5)  = 10
cvi(-10.4)       = -10          cvi(10.4)       = 10
cvi(-10.6)       = -11          cvi(10.6)       = 11

The values of "cvi(-10.5)" and "cvi(10.5)" are
unspecified; they may be -10 or -11, and 10 or 11,
respectively.
```

Table 1.169-3. Rounding Directions for (Long) Real to (Long) Integer Conversion Procedures

1.170. formatted

```
COMPILETIME
BITS
<macro>     formatted;
```

Table 1.170-1. formatted

formatted is a bit that specifies that an input or output string representation of a (long) bits begins with a single quote, as in program text. It may be passed to cvs and $removeBits.

1.171. $formParagraph

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
STRING
PROCEDURE   $formParagraph  (STRING s;
                             OPTIONAL INTEGER
                                rightMargin,
                                firstLineIndent;
                             OPTIONAL STRING ctrlChars;
                             OPTIONAL BITS ctrlBits);
```

Table 1.171-1. $formParagraph

$formParagraph returns a "filled" form of a string s, i.e., an s with as many words as possible on each line subject to the constraint that no line be longer than a specified maximum number of characters.

"Blank characters" are blanks, tabs, and end-of-line and end-of-page characters. A "word" is a sequence of non-blank characters. Lines are formed from the words in s (retaining the order of the words); words are separated from each other by a single blank character, except as noted below.

The first line starts with (firstLineIndent MAX 0) blanks. Subsequent lines start with (-firstLineIndent MAX 0) blanks. If firstLineIndent is less than zero, all but the first line are indented.

Lines (counting indentation) can have at most rightMargin characters. Each line contains as many words as can fit. If a word is longer than a line can be, it is put on a line by itself. The lines are separated by <eol> characters.

If not specified, rightMargin defaults to 72.

If append is set in ctrlBits, the "two-blank heuristic" is used:

- if two consecutive words occupy the same line,
- and the first word ends in a period,
- and the second word does not start with a lowercase letter,
- then the two words are separated by two blanks rather than one blank (this may result in the second word being pushed onto the next line).

The first character of ctrlChars is an alias for space that is used to force two words to be on the same line. For example, if the first character in ctrlChars is '@' then "United@States" results in "United States" being on the same line. If the first character of ctrlChars is space (or ctrlChars is Zero), then there is no space alias.

The second character of ctrlChars is an alias for period that is used to circumvent the two-blank heuristic in a specific instance. For example, if the second character in ctrlChars is '%', then for the string "i.e% John", $formParagraph replaces the '%' with '.' and puts one space before John, whereas for the string "i.e. John", $formParagraph would have put two spaces before John. If the second character of ctrlChars is space (or ctrlChars is shorter than two characters), then there is no period alias.

The behavior is undefined if the space alias and the period alias are the same (non-blank) character.

This procedure is considered a temporary feature and may be changed or enhanced in the future.

1.172. $fullPathNames

```
COMPILETIME
BITS
<macro>      $fullPathNames;
```

Table 1.172-1. $fullPathNames

$fullPathNames is a bit that specifies that full path names of files are to be included in the output. It may be passed to $directory.

1.173. generateMultipleQuickSort

generateMultipleQuickSort is a macro provided by the sorting package, SRTMOD, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.174. generateQuickSort

generateQuickSort is a macro provided by the sorting package, SRTMOD, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.175. $getCommandLine

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
BOOLEAN
PROCEDURE   $getCommandLine
                          (PRODUCES STRING s);
```

Table 1.175-1. $getCommandLine

Many operating systems allow the user to invoke a program using a command line, which can specify information other than just the program name. For example, a command line "foo a b c" could mean to execute the program foo with arguments a, b, and c. $getCommandLine gives a program access to command arguments, which may have been set by the operating system or by a MAINSAIL program.

A command line can be specified in three ways:

- At the operating system level, when MAINSAIL is invoked
- At the MAINEX "*" prompt
- During program execution, using $setCommandLine

In the first case, MAINSAIL alters the command line if necessary to put it into a more portable form:

- On some operating systems, the program name is part of the command line, i.e., "foo" in the example above. In this case, MAINSAIL removes the program name (and any blank space after it) from the command line. In the example, the command line would be altered to be just "a b c". Thus the name of the program is not available in the command line formed by MAINSAIL (it may be available as $programName).

- On some operating systems, the command line is parsed and broken into a sequence of arguments before MAINSAIL has access to it. In this case, MAINSAIL concatenates the arguments, separated by a single blank, into a string that becomes the command line. For example, if the command line typed to the operating system were "foo a b c", i.e., with several blanks separating the arguments, and the operating system broke this into a program name and three arguments a, b, and c, MAINSAIL would form the string "a b c" as the command line.

- The exact rules for the formation of the command line information provided by the operating system into a single command line string by MAINSAIL is provided in each system-specific MAINSAIL user's guide if not covered by the above points.

- The resulting command line is concatenated onto the end of the "COMMANDSTRING" value specified in the MAINSAIL bootstrap file. This value is the null string unless a value was specified for the "COMMANDSTRING" command when CONF was used to make the bootstrap. If the resulting concatenated string is non-Zero, MAINEX executes module(s), one for each line in the command string (interpreting the first word of the line as the module name, and the rest of the line as the module's arguments), until the command string is exhausted and then returns to the operating system. In this case, the MAINSAIL banner and the MAINEX "*" prompt are not displayed.

In the second case, MAINSAIL removes the first word (presumably the module or file name to be invoked), and any blank space that follows it, and sets the command line to what remains. This means that the user cannot specify to the "*" prompt a file name containing embedded blanks, since the components of the file name after the first word would be treated as part of the command line rather than as part of the file name.

The MAINSAIL runtime system makes the operating system's command line arguments visible to a MAINSAIL program by calling the system procedure $setCommandLine. $setCommandLine sets the command line to its argument (which may be the null string), with leading and trailing blanks and tabs removed, and also sets an internal boolean variable to true to indicate that the command line is set.

$getCommandLine is used by a program to examine the command line. $getCommandLine examines the internal boolean variable maintained by $setCommandLine. If it is true, then $getCommandLine sets its argument to the command line, sets the boolean variable to false to indicate that the command line is no longer set, and returns true. If the boolean variable is

false, then $getCommandLine sets its argument to the null string and returns false. The new procedure $removeWord may be useful in parsing command lines.

If several calls are made to $getCommandLine without any intervening calls to $setCommandLine, the first one will obtain the command line, and subsequent ones will not. If a particular access to the command line does not need to process all of it, call $setCommandLine with the unprocessed part to make it available to the next call to $getCommandLine.

$getCommandLine should be called first thing in a module's initial procedure. MAINSAIL system calls may cause an arbitrary amount of work to be done, possibly including the invocation of other modules that change the remembered command line.

Example 1.175-2 shows some examples of the command line mechanism.

```
(1) mainsa foo<eol>

    MAINSAIL is invoked and executes the module FOO. The
    command line is the null string. The MAINSAIL herald
    is not displayed. When module FOO terminates,
    MAINSAIL exits to the operating system.

(2) mainsa foo a b c<eol>

    Same as (1), except that the command line is "a b c".

(3) mainsa<eol>
    *foo a b c<eol>

    Same as (2) except that the MAINSAIL herald is
    displayed and MAINSAIL returns to the "*" prompt if
    FOO terminates normally.
```

Example 1.175-2. Examples of the Use of Command Line

$invokeModule uses only the first word of its string argument as the name of the module to invoke; the remainder of the string is used to set the command line.

The definition of $getCommandLine is intended to allow a program to distinguish between an operating system that does not support command line arguments (the first call to $getCommandLine returns false) and an operating system that does support command line

arguments, but where no arguments were provided for the current program ($getCommandLine returns true but sets its argument to the null string).

Several XIDAK utility programs examine the command line, as desribed in the "MAINSAIL Utilities User's Guide" and the "MAINSAIL Compiler User's Guide". The command line syntaxes described therein are subject to change.

1.176. $getEofPos

```
LONG INTEGER
PROCEDURE   $getEofPos  (POINTER(file) f);
```

Table 1.176-1. $getEofPos

$getEofPos returns a value greater than or equal to the current end-of-file position of f (in character units if f is a text file or a file open for PDF I/O, storage units otherwise). The result of $getEofPos is undefined if f is not a byte stream file (a file that can be opened for random output) with a definite ending position.

1.177. $getInArea

```
STRING
PROCEDURE   $getInArea  (STRING s;
                         OPTIONAL POINTER($area) area);

POINTER
PROCEDURE   $getInArea  (POINTER p;
                         OPTIONAL POINTER($area) area);
```

Table 1.177-1. $getInArea (Generic)

$getInArea does nothing if s or p is Zero or if $inArea is true of its arguments; otherwise, it copies the characters of s or the chunk pointed to by p into area, and returns the string descriptor or pointer referencing the copied data. If s is a string, then "$getInArea(s,a)" is equivalent to:

```
IF $inArea(s,a) THEN s EL $getToTop(s,a)
```

If area is omitted, $defaultArea is used. This is useful, for example, when a string has been created in static space and the user wishes to have the string collected like a normal MAINSAIL string, or if the static space is to be reused to allocate a new string. For example, if a foreign language procedure returns a string with length len at charadr ch, $getInArea can be used in conjunction with the procedure newString as in Example 1.177-2. The resulting string s is a string in MAINSAIL string space that is subject to MAINSAIL string collection.

```
INTEGER len; CHARADR ch; STRING s;
...
# foreignProcedure creates a string at ch with length len
foreignProcedure(ch,len);
s := $getInArea(newString(ch,len));
```

Example 1.177-2. Use of $getInArea

1.178. getPos

```
LONG INTEGER
PROCEDURE   getPos      (POINTER(file) f);
```

Table 1.178-1. getPos

getPos returns the current position of f (in character units if f is a text file or a file open for PDF I/O, storage units otherwise).

1.179. $getSubcommands

$getSubcommands allows a program to process a series of MAINEX subcommands. $getSubcommands is documented in detail in the "MAINSAIL Utilities User's Guide".

```
INTEGER              c,j,k;
LONG INTEGER         pos,savePos;
POINTER(textFile)    f;

open(f,"results",create!random);

# Suppose you are writing characters to f and you come to
# a point where you know another character is needed,
# but you don't yet know its value and you need to
# write out other characters before you will know its
# value.  You can save the position at which this
# character belongs,

savePos := getPos(f);

# temporarily write out an "x", say, as a place holder,

cWrite(f,'x');

# and then continue writing out other characters, e.g.,

write(f,"abc");

. . .

# until you know the value of the original character,
# say c, so you can replace the "x" written in its place
# with the proper value.  Save the current position in f,
# position back to the position for the character c, write
# out the desired character, and position back to where
# you were to continue writing more characters:

pos := getPos(f);
setPos(f,savePos); cWrite(f,c);
setPos(f,pos);
```

Example 1.178-2. Use of getPos

1.180. $getToTop

```
STRING
PROCEDURE    $getToTop    (STRING s;
                           OPTIONAL POINTER($area) area);
```

Table 1.180-1. $getToTop

The procedure $getToTop copies the characters of a string the top of area's string space. A better way to ensure that a string is in a given area (whether at the top of string space or not) is $getInArea (see Table 1.177-1).

1.181. The Global Symbol Table Procedures

MAINSAIL supports a method of establishing records that are visible to every module in the current execution. The records are established using a string key. A key should be chosen to be unique; it should be long and descriptive, including at least the name of the program or system using it.

Each record established by a user program is prefixed by the class $globalSymbol:

```
CLASS $globalSymbol (STRING $key);
```

The procedures shown in Table 1.181-1 are used to manipulate the global symbol table. $globalLookup returns the record with $key equal to key, or nullPointer if no such record exists. $globalEnter enters the record pointed to by p into the global symbol table. The effect is undefined if p.$key is the key of a record already in the global symbol table; call $globalLookup before entering p to ensure that p's key is unique. $globalRemove removes and returns the record with $key equal to key; if no such record exists, it returns nullPointer.

Suppose a module M has a large number of data sections that wish to share some data under the symbol:

```
M: shared data
```

In the initial procedure of M, declare:

```
CLASS($globalSymbol) mSharedData (... extra fields...);
POINTER(mSharedData) p;
```

```
POINTER($globalSymbol)
PROCEDURE    $globalLookUp
                          (STRING key);

PROCEDURE    $globalEnter
                          (POINTER($globalSymbol) p);

POINTER($globalSymbol)
PROCEDURE    $globalRemove
                          (STRING key);
```

Table 1.181-1. Global Symbol Table Procedures

If the first data section of M is to establish the field values, and all subsequent data sections to use those values, the code would look something like:

```
IF NOT p := $globalLookup("M: shared data") THENB
    p := new(mSharedData); p.$key := "M: shared data";
    ... set other fields of p...
    $globalEnter(p) END;
... use fields of p...
```

Interface fields of a bound data section provide a more efficient (but sometimes less convenient) repository for data shared among many modules.

The symbol table manipulated by the global symbol table procedures is not related to the global symbol table in which entries are made by the directive "$GLOBALREDEFINE".

1.182. $gmt

```
COMPILETIME
BITS
<macro>      $gmt;
```

Table 1.182-1. $gmt

$gmt is a bit that specifies that Greenwich Mean Time (GMT) date(s) and/or time(s) are input to or output from the procedure to which it is passed. It may be passed to $assembleDate,

$assembleTime, $date, $dateAndTime, $dateAndTimeToStr, $strToDate, $strToDateAndTime, $strToTime, and $time. It may be returned by $dateFormat and $timeFormat to indicate the format of the long integer date or time argument.

1.183. $GMTtoLocalTime

```
COMPILETIME
BITS
<macro>      $GMTtoLocalTime;
```

Table 1.183-1. $GMTtoLocalTime

$GMTtoLocalTime is a bit that specifies that a conversion from Greenwich Mean Time (GMT) to local time is to be performed. It may be passed to $assembleDateAndTime and $strToDateAndTime.

1.184. $gotValue

```
BOOLEAN
<macro>      $gotValue          (POINTER(file) f);
```

Table 1.184-1. $gotValue

$gotValue returns true if the last read from the file f returned a value; i.e., if end-of-file was not encountered during the last read from the file f.

$gotValue is a less ambiguous (and more efficient) test for end-of-file than eof. $gotValue returns false only when a read is attempted beyond end-of-file; eof may return true when a read is attempted beyond end-of-file or immediately before such a read is attempted. Both $gotValue and eof suffer from the drawback that some operating systems do not permit MAINSAIL to ascertain the end-of-file position exactly. Where possible, the programmer should design files that indicate their own end-of-file, e.g., by some special data value.

$gotValue may become true or false after a call to the textFile or dataFile form of read. It is not affected by other input procedures (even if end-of-file is encountered) since those

procedures (which include fldRead, cRead, scan, $storageUnitRead, and $pageRead) return a distinctive value when they encounter end-of-file.

1.185. $hash

```
INTEGER
PROCEDURE    $hash        (STRING key;
                           INTEGER buckets);
```

Table 1.185-1. $hash

$hash is a general-purpose hash function that generates an integer in the range 0 to buckets - 1 based on key (the effect is undefined if buckets is less than one). The algorithm differs from that used by HSHMOD, and is subject to change from release to release of MAINSAIL.

$hash is not a module interface procedure, so calls to it may be usefully prefixed with "INLINE" or "$ALWAYSINLINE"; this may be advisable when fast hashing is important. The code for $hash is not completely trivial, however, so this expansion should not be made too many times per module, or it will take up a lot of space.

1.186. hex

```
COMPILETIME
BITS
<macro>      hex;
```

Table 1.186-1. hex

hex is a bit that specifies that a hexadecimal string representation is input to or output from the procedure to which it is passed. It may be passed to cvb, cvlb, cvs and $removeBits. It may be returned by $preferredRadix.

1.187. $homeDirectory

```
STRING
PROCEDURE    $homeDirectory
                          (OPTIONAL BITS ctrlBits;
                           PRODUCES OPTIONAL STRING msg);
```

Table 1.187-1. $homeDirectory

If the operating system defines a notion of "home directory", $homeDirectory returns its name; consult the appropriate system-specific MAINSAIL user's guide for details. If an error occurs, the null string is returned, msg is set to a string describing the error, and if errorOK is not set in ctrlBits, an error message is issued. errorOK is the only valid bit in ctrlBits.

1.188. HSHMOD Procedures

The HSHMOD procedures provide a set of facilities for constructing hash tables of records. HSHMOD is documented in detail in the "MAINSAIL Utilities User's Guide".

1.189. $hyphenateDate

```
COMPILETIME
BITS
<macro>      $hyphenateDate;
```

Table 1.189-1. $hyphenateDate

$hyphenateDate is a bit that specifies that a hyphenated date string is to be output. It may be passed to $dateAndTimeToStr and $dateToStr.

1.190. $inArea

```
BOOLEAN
PROCEDURE    $inArea      (STRING s;
                           OPTIONAL POINTER($area) area);

BOOLEAN
PROCEDURE    $inArea      (POINTER p;
                           OPTIONAL POINTER($area) area);
```

Table 1.190-1. $inArea (Generic)

$inArea returns true if and only if the text referenced by s or the chunk pointed to by p is in area ($defaultArea if area is not specified). The effect is undefined if s or p is dangling.

1.191. $includeTimeZone

```
COMPILETIME
BITS
<macro>      $includeTimeZone;
```

Table 1.191-1. $includeTimeZone

$includeTimeZone is a bit that specifies that a time zone string is to be included in the output of the procedure to which it is passed. It may be passed to $dateAndTimeToStr and $dateToStr.

1.192. $includeWeekday

```
COMPILETIME
BITS
<macro>      $includeWeekday;
```

Table 1.192-1. $includeWeekday

$includeWeekday is a bit that specifies that the day of the week is to be included in the output string. It may be passed to $dateAndTimeToStr and $dateToStr.

1.193. $initRand

$initRand is used to initialize one of the pseudo-random number generation algorithms provided by $ranMod, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.194. $initsRand

$initsRand is used to initialize one of the pseudo-random number generation algorithms provided by $ranMod, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.195. input

```
COMPILETIME
BITS
<macro>      input;
```

Table 1.195-1. input

input is a bit that specifies that input operations are to be allowed on the file that is being opened. It may be passed to $createUniqueFile, open, and $reOpen.

1.196. $insertLeft

```
COMPILETIME
BITS
<macro>     $insertLeft;
```

Table 1.196-1. $insertLeft

$insertLeft is a bit that specifies that a coroutine is to be inserted into a coroutine tree to the left of another coroutine. It may be passed to $moveCoroutine.

1.197. $insertRight

```
COMPILETIME
BITS
<macro>     $insertRight;
```

Table 1.197-1. $insertRight

$insertRight is a bit that specifies that a coroutine is to be inserted into a coroutine tree to the right of another coroutine. It may be passed to $moveCoroutine.

1.198. $intmodInfo

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

$intmodInfo is analogous to $moduleInfo, except it works on intmods or intmod libraries instead of objmods or objmod libraries, and cmdLine has the form of arguments to INTLIB's "DIRECTORY" command instead of MODLIB's. See the description of $moduleInfo for more details.

The $legalNoticeStr bit is never set in $moduleRec.$cmpBits for an intmod.

```
BOOLEAN
PROCEDURE    $intmodInfo (STRING cmdLine;
                          PRODUCES POINTER($moduleRec)
                              modList;
                          OPTIONAL BITS ctrlBits);
```

Table 1.198-1. $intmodInfo

1.199. $invokeModule

```
BOOLEAN
PROCEDURE    $invokeModule
                         (STRING moduleOrFileNameAndArgs;
                          OPTIONAL BITS ctrlBits;
                          PRODUCES OPTIONAL LONG INTEGER
                              bindCpuTime);
```

Table 1.199-1. $invokeModule

$invokeModule invokes the module (i.e., binds then disposes its data section, like MAINEX) named by the first (blank- or tab-delimited) word of moduleOrFileNameAndArgs (or contained in the file named by the same word) with the arguments composing the remainder of moduleOrFileNameAndArgs (arguments are discussed in more detail under the entry for $getCommandLine). It returns false if a bound data section already exists for the module or if the module cannot be invoked.

The first word of moduleOrFileNameAndArgs is assumed to be a file name if it is not a valid module identifier. The file name may contain a device module specification. The first word of moduleOrFileNameAndArgs may be terminated with a comma, in which case MAINEX subcommands are read from cmdFile (see the "MAINSAIL Utilities User's Guide").

The valid ctrlBits bits are the same as for bind, with the same meanings.

If the call to bind was successful (i.e., if $invokeModule returns true), bindCpuTime is set to the number of CPU time units (see the description of $cpuTimeResolution) used by the call to bind; this time includes the execution of the initial procedure of the module.

When an unhandled exception causes the initial procedure of a module invoked with $invokeModule to be aborted, $invokeModule unbinds the module.

1.200. $ioSize

```
INTEGER
<macro>     $ioSize      (POINTER(file) f;
                          INTEGER typ);
```

Table 1.200-1. $ioSize

"$ioSize(f,x)", where x is a MAINSAIL data type code, returns the size of x based on the format of the data in f. For example, if f contains host data, "$ioSize(f,x)" returns the same value as "size(x)", but if f contains PDF data, "$ioSize(f,x)" returns the same value as "pdfChars(x)" (see the description of PDFMOD in the "MAINSAIL Utilities User's Guide").

$ioSize returns 0 if f is a text file since there is no fixed size for the string representation of a data type.

The result of $ioSize is undefined if the type code is not boolean, (long) integer, (long) real, or (long) bits.

1.201. isAlpha

```
$BUILTIN
BOOLEAN
PROCEDURE   isAlpha      (INTEGER char);
```

Table 1.201-1. isAlpha

isAlpha returns true if and only char is the character code for an alphabetic character, i.e., one of the uppercase letters "A" through "Z" or one of the lowercase letters "a" through "z". isAlpha is independent of the underlying character set.

```
isAlpha('b') = TRUE
isAlpha('M') = TRUE
isAlpha('#') = FALSE
isAlpha('9') = FALSE
```

Example 1.201-2. Use of isAlpha

1.202. $isArray

```
INLINE
BOOLEAN
PROCEDURE   $isArray    (POINTER p);
```

Table 1.202-1. $isArray

$isArray returns true if and only if its argument is an array pointer (e.g., as returned by the array form of cvp).

1.203. $isBound

```
BOOLEAN
PROCEDURE   $isBound    (STRING modName);
```

Table 1.203-1. $isBound

$isBound returns true if a bound data section exists for the module named modName.

modName is a true module name, not a dummy module name as established by setModName; e.g., if the dummy name "ABC" has been established for a module DEF, and the module DEF is bound but no module named ABC is bound, "$isBound("ABC")" returns false.

1.204. isLowerCase

```
$BUILTIN
BOOLEAN
PROCEDURE   isLowerCase (INTEGER char);
```

Table 1.204-1. isLowerCase

isLowerCase returns true if and only if char is the character code for a lowercase letter, i.e., one of the lowercase letters "a" through "z". isLowerCase is independent of the underlying character set.

```
isLowerCase('b') = TRUE
isLowerCase('M') = FALSE
isLowerCase('#') = FALSE
isLowerCase('9') = FALSE
```

Example 1.204-2. Use of isLowerCase

1.205. isNul

```
$BUILTIN
BOOLEAN
PROCEDURE   isNul       (INTEGER char);
```

Table 1.205-1. isNul

isNul returns true if and only if char is the null character $nulChar. See Section 1.259 for further information about the treatment of null characters in an input file.

1.206. isUpperCase

```
$BUILTIN
BOOLEAN
PROCEDURE   isUpperCase (INTEGER char);
```

Table 1.206-1. isUpperCase

isUpperCase returns true if and only if char is the character code for an uppercase letter, i.e., one of the uppercase letters "A" through "Z". isUpperCase is independent of the underlying character set.

```
isUpperCase('b') = FALSE
isUpperCase('M') = TRUE
isUpperCase('#') = FALSE
isUpperCase('9') = FALSE
```

Example 1.206-2. Use of isUpperCase

1.207. keepNul

```
COMPILETIME
BITS
<macro>     keepNul;
```

Table 1.207-1. keepNul

keepNul is a bit that specifies that null characters are not to be discarded from input operations on a file. It may be passed to $createUniqueFile, open, and $reOpen.

1.208. $killCoroutine

```
BOOLEAN
PROCEDURE    $killCoroutine
                          (POINTER($coroutine) p;
                           OPTIONAL BITS ctrlBits);

BOOLEAN
PROCEDURE    $killCoroutine
                          (STRING coroutineName;
                           OPTIONAL BITS ctrlBits);
```

Table 1.208-1. $killCoroutine (Generic)

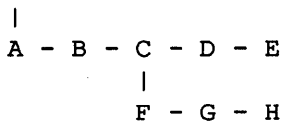
$killCoroutine is used to "kill" (deallocate) a coroutine and, by default, all of its descendants (i.e., all the coroutines that make up the subtree rooted at the argument coroutine). In each coroutine to be killed, $abortProcedureExcpt is first raised to allow the coroutine to clean up after itself. Then each dying coroutine's stack is deallocated and its $coroutine record is taken off all lists and marked as killed. An attempt to resume a killed coroutine is an error. The $coroutine record is reclaimed by the garbage collector when it becomes inaccessible.

$descendantKilledExcpt is raised in the ancestors of a killed coroutine to inform the coroutines that their descendant has died. The exception must be propagated with $raise; it may not be handled with $raiseReturn or by falling out of a handler. $exceptionPointerArg points to the $coroutine record of the dead coroutine, in which $abortProcedureExcpt has already been raised, but of which the $coroutine record has not been unlinked from the coroutine tree.

The root coroutine to be killed can be specified either by name or by a pointer to its $coroutine record.

It is an error to kill the invoking coroutine or any of its ancestors; $resumeCoroutine must be used to kill the invoking coroutine.

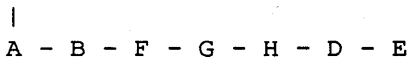
Valid ctrlBits bits are errorOK and $nonRecursive. errorOK suppresses any error messages. If an error occurs, false is returned, otherwise true. If $nonRecursive is specified, $killCoroutine does not kill the specified coroutine's children, but replaces the dying coroutine in the coroutine tree with its children (the left-to-right order of the children is preserved). For example, if the coroutine tree looks like (in part):



then:

```
$killCoroutine("C",$nonRecursive)
```

produces the tree:



1.209. $killedCoroutine

```
BOOLEAN
<macro>     $killedCoroutine
                        (POINTER($coroutine) p);
```

Table 1.209-1. $killedCoroutine

$killedCoroutine returns true if and only if the coroutine record pointed to by p represents a coroutine that has been killed with $killCoroutine.

1.210. last

```
$BUILTIN COMPILETIME
INTEGER
PROCEDURE   last          (STRING s);
```

Table 1.210-1. last

"last" returns the character code for the last character of a string.

If s is "", -1 is returned. -1 is not a valid character code.

```
last("abc") = 'c' = 99 # assuming the ASCII character set
last("") = -1
```

Example 1.210-2. Use of last

1.211. lbMask

```
COMPILETIME
LONG BITS
PROCEDURE   lbMask      (INTEGER lowBit,highBit);
```

Table 1.211-1. lbMask

lbMask makes a "bit mask", which is a contiguous sequence of 1-bits embedded within 0-bits. lbMask is analogous to bMask, except that its result is a long bits instead of a bits. See Section 1.30 for a description of bMask.

A garbage collection cannot occur during a call to lbMask.

1.212. lDisplacement

```
$BUILTIN
LONG INTEGER
PROCEDURE   lDisplacement
                        (ADDRESS a,b);

$BUILTIN
LONG INTEGER
PROCEDURE   lDisplacement
                        (CHARADR a,b);
```

Table 1.212-1. lDisplacement (Generic)

lDisplacement computes the distance between two addresses or charadrs.

The address form returns the number of storage units from address a to address b.

The charadr form returns the number of characters from charadr a to charadr b.

If a is beyond b, the result is negative.

A garbage collection cannot occur during a call to lDisplacement.

1.213. length

```
$BUILTIN COMPILETIME
INTEGER
PROCEDURE   length      (STRING s);
```

Table 1.213-1. length

length returns the number of characters in a string.

```
length("abc") = 3
length("") = 0
```

Example 1.213-2. Use of length

1.214. $length

```
INTEGER
PROCEDURE   $length     (BOOLEAN v);

INTEGER
PROCEDURE   $length     (INTEGER v);
```

Table 1.214-1. $length (Generic) (continued)

```
INTEGER
PROCEDURE    $length      (LONG INTEGER v);

INTEGER
PROCEDURE    $length      (REAL v;
                           OPTIONAL BITS format);

INTEGER
PROCEDURE    $length      (LONG REAL v;
                           OPTIONAL BITS format);

INTEGER
PROCEDURE    $length      (BITS v;
                           OPTIONAL BITS format);

INTEGER
PROCEDURE    $length      (LONG BITS v;
                           OPTIONAL BITS format);
```

Table 1.214-1. $length (Generic) (end)

$length returns the length of the string representation of v, as specified by format, if applicable. Specifically:

```
$length(v)
```

returns the same value as:

```
length(cvs(v))
```

and:

```
$length(v,format)
```

the same as:

```
length(cvs(v,format))
```

The difference is that $length does not put characters into string space, and so is more efficient than the equivalent forms calling length and cvs. However, if the string is actually needed later, it is more efficient to call cvs; i.e., instead of:

```
$length(v,...); ...; s := cvs(v,...); <use s>
```

do:

```
length(s := cvs(v,...)); ...; <use s>
```

1.215. ln

```
REAL
PROCEDURE    ln          (REAL x);

LONG REAL
PROCEDURE    ln          (LONG REAL x);
```

Table 1.215-1. ln (Generic)

ln returns the logarithm base e of x, where e is the base of the natural logarithms.

It is an error if x is less than or equal to zero.

1.216. The Load Procedures

```
$BUILTIN
BOOLEAN
PROCEDURE    boLoad      (ADDRESS a;
                          OPTIONAL INTEGER dspl);

$BUILTIN
INTEGER
PROCEDURE    iLoad       (ADDRESS a;
                          OPTIONAL INTEGER dspl);
```

Table 1.216-1. The Load Procedures (continued)

```
$BUILTIN
LONG INTEGER
PROCEDURE    liLoad       (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
REAL
PROCEDURE    rLoad        (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
LONG REAL
PROCEDURE    lrLoad       (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
BITS
PROCEDURE    bLoad        (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
LONG BITS
PROCEDURE    lbLoad       (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
STRING
PROCEDURE    sLoad        (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
POINTER
PROCEDURE    pLoad        (ADDRESS a;
                           OPTIONAL INTEGER dspl);

$BUILTIN
ADDRESS
PROCEDURE    aLoad        (ADDRESS a;
                           OPTIONAL INTEGER dspl);
```

Table 1.216-1. The Load Procedures (continued)

```
$BUILTIN
CHARADR
PROCEDURE    cLoad        (ADDRESS a;
                           OPTIONAL INTEGER dspl);
```

Table 1.216-1. The Load Procedures (end)

load is used to load a value from a memory address.

"v := xLoad(a,d)" loads a value of type x from the memory location given by "displace(a,d)", where d is a displacement in storage units. If "displace(a,d)" is undefined, then "xLoad(a,d)" is also undefined. The string form loads only a string descriptor; the characters of the string are not referenced.

The effect is undefined if a is nullAddress.

Another form of cLoad, which loads a character from a charadr, is described in Section 1.51.

```
REAL y;
CLASS c (REAL x; POINTER(c) link);
POINTER(c) p,q;

p := new(c);

...

y := rLoad(cva(p));              # same as y := p.x
q := pLoad(cva(p),DSP(c.link)) # same as q := p.link
```

Example 1.216-2. Use of the Load Procedures

1.217. $localTime

```
COMPILETIME
BITS
<macro>      $localTime;
```

Table 1.217-1. $localTime

$localTime is a bit that specifies that local date(s) and/or time(s) are input to or output from the procedure to which it is passed. It may be passed to $assembleDate, $assembleTime, $date, $dateAndTime, $dateAndTimeToStr, $strToDate, $strToDateAndTime, $strToTime, and $time. It may be returned by $dateFormat and $timeFormat to indicate the format of the long integer date or time argument.

1.218. $localTimeToGMT

```
COMPILETIME
BITS
<macro>      $localTimeToGMT;
```

Table 1.218-1. $localTimeToGMT

$localTimeToGMT is a bit that specifies that a conversion from local time to Greenwich Mean Time (GMT) is to be performed. It may be passed to $assembleDateAndTime and $strToDateAndTime.

1.219. log

```
REAL
PROCEDURE    log          (REAL x);

LONG REAL
PROCEDURE    log          (LONG REAL x);
```

Table 1.219-1. log (Generic)

log returns the logarithm base ten of x.

It is an error if x is less than or equal to zero.

1.220. $log2

```
COMPILETIME
INTEGER
<macro>      $log2        (INTEGER x);

COMPILETIME
LONG INTEGER
<macro>      $log2        (LONG INTEGER x);
```

Table 1.220-1. $log2

$log2 returns the logarithm base 2, truncated if necessary to the next lower whole number, of x. If x is not a constant, a compiletime error occurs. The effect of $log2 is undefined if x is not positive.

1.221. logFile

```
# system variable
POINTER(textFile) logFile;
```

Table 1.221-1. logFile

logFile is MAINSAIL's standard output file. cmdFile and logFile are described in Section 18.12 of part I of the "MAINSAIL Language Manual".

1.222. lookUpLogicalName

```
STRING
PROCEDURE   lookUpLogicalName
                          (STRING logicalName);
```

Table 1.222-1. lookUpLogicalName

lookUpLogicalName returns the true file name associated with the logical name logicalName. A logical name association may be established by means of enterLogicalName. The null string is returned if no true file name is associated with logicalName.

1.223. $mainsailExec

The MAINSAIL executive, MAINEX, can be invoked from a user program by calling the procedure $mainsailExec. This feature is documented in detail under MAINEX in the "MAINSAIL Utilities User's Guide".

1.224. $majorVersion

```
# system variable
INTEGER $majorVersion;
```

Table 1.224-1. $majorVersion

The value of $majorVersion is the major part of the MAINSAIL version number. For example, if running version 12.10 of MAINSAIL, $majorVersion is 12.

The effect of altering $majorVersion is undefined.

1.225. $maxChar

```
COMPILETIME
INTEGER
<macro>      $maxChar;
```

Table 1.225-1. $maxChar

$maxChar is the maximum valid value of a character code, equal to "(2 ^ $bitsPerChar - 1)". It is always equal to 255, since characters occupy eight bits.

1.226. $maxInteger

```
COMPILETIME
INTEGER
<macro>      $maxInteger;
```

Table 1.226-1. $maxInteger

$maxInteger is the operating-system-dependent maximum allowed integer value. Attempting to create an integer value larger than $maxInteger may lead to overflow, which has undefined effects.

1.227. $maxLongInteger

```
COMPILETIME
LONG INTEGER
<macro>     $maxLongInteger;
```

Table 1.227-1. $maxLongInteger

$maxLongInteger is the operating-system-dependent maximum allowed long integer value. Attempting to create a long integer value larger than $maxLongInteger may lead to overflow, which has undefined effects.

1.228. $minInteger

```
COMPILETIME
INTEGER
<macro>     $minInteger;
```

Table 1.228-1. $minInteger

$minInteger is the operating-system-dependent minimum allowed integer value. Attempting to create an integer value smaller than $minInteger may lead to overflow, which has undefined effects.

1.229. $minLongInteger

```
COMPILETIME
LONG INTEGER
<macro>     $minLongInteger;
```

Table 1.229-1. $minLongInteger

$minLongInteger is the operating-system-dependent minimum allowed long integer value. Attempting to create a long integer value smaller than $minLongIntege may lead to overflow, which has undefined effects.

1.230. $minorVersion

```
# system variable
INTEGER $minorVersion;
```

Table 1.230-1. $minorVersion

The value of $minorVersion is the minor part of the MAINSAIL version number. For example, if running version 12.10 of MAINSAIL, $minorVersion is 10.

The effect of altering $minorVersion is undefined.

1.231. $moduleInfo

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

$moduleInfo is used from a program to obtain information about one or more objmods, which may reside either in individual files or in libraries. The information is returned as a linked list of records of the class $moduleRec, described below. The records are sorted in ascending order by module name.

```
BOOLEAN
PROCEDURE    $moduleInfo (STRING cmdLine;
                          PRODUCES POINTER($moduleRec)
                              modList;
                          OPTIONAL BITS ctrlBits);
```

Table 1.231-1. $moduleInfo

cmdLine can have the same form as the arguments to MODLIB's "DIRECTORY" command, namely "libName{=fileName} {modList}". libName can be "*" to indicate that no library is involved, i.e., that modList specifies object modules in individual files. modList is a possibly empty list of module specifications separated by blanks. A module specification can have any one of three forms:

```
moduleName            name of a module
fileName              name of file (if not a valid module name)
moduleName=fileName   module in specified file
```

The last two forms are used when libName = "*". The only reason to use the last form is if the fileName would appear to be a valid module name. If a libName is given and modList is omitted, then information is provided for all modules in the library.

ctrlBits can specify errorOK to suppress error messages, and $noLegalNotice may be set in a call to $moduleInfo to suppress fetching of the legal notice from each module (this speeds up the call significantly). A result of FALSE indicates that an error occurred (e.g., a file could not be opened).

Fields of $moduleRec include:

STRING	$dirName	name recorded in the directory, if in a library
STRING	$modName	actual name of the module, usually the same as $dirName
LONG INTEGER	$startPage	start page if in a library
LONG INTEGER	$numPages	number of pages
LONG INTEGER	$compileDate	date compiled
LONG INTEGER	$compileTime	time compiled
INTEGER	$majorVersion	major version
INTEGER	$minorVersion	minor version
LONG BITS	$cmpBits	see below
STRING	$legalNoticeStr	legal notice string
POINTER($moduleRec)	$next	link to next record

Predefined long bits constants for $modBits are:

$hasInitialProc	the module has an initial procedure
$hasFinalProc	the module has a final procedure
$inlinesHaveBodies	"INLINE" procedures were given bodies
$arithmeticChecked	compiled with the "ACHECK" option
$checked	compiled with the "CHECK" option
$debugBit	compiled with the "DEBUG" option
$optimized	compiled with the "OPTIMIZE" option
$countingPerModule	compiled with the "PERMOD" option
$countingPerProc	compiled with the "PERPROC" option
$countingPerStmt	compiled with the "PERSTMT" option
$timingPerModule	compiled with the "MODTIME" option
$timingPerProc	compiled with the "PROCTIME" option
$unbound	compiled with the "UNBOUND" option

See Example 1.231-2.

```
PROCEDURE printModulesInLibrary (STRING libraryFileName);
BEGIN
POINTER($moduleRec) p,q;
IF NOT $moduleInfo(libraryFileName,p) THEN RETURN;
write(logFile,
    "Modules in library file ",libraryFileName,eol);
WHILE q := p DOB
    write(logFile,p.$modName,eol);
    p := p.$next; dispose(q) END;
END;
```

Example 1.231-2. Use of $moduleInfo

1.232. $moduleName

```
STRING
PROCEDURE    $moduleName (POINTER p;
                          OPTIONAL POINTER($area) area);
```

Table 1.232-1. $moduleName

If p points to a data section, $moduleName returns the uppercase name of the associated module. Otherwise, $moduleName returns the null string. area specifies the destination area for the resulting string.

1.233. $moveCoroutine

```
BOOLEAN
PROCEDURE    $moveCoroutine
                        (POINTER($coroutine)
                             coroutine,newParent;
                         OPTIONAL BITS ctrlBits);

BOOLEAN
PROCEDURE    $moveCoroutine
                        (STRING coroutine,newParent;
                         OPTIONAL BITS ctrlBits);
```

Table 1.233-1. $moveCoroutine (Generic)

$moveCoroutine moves coroutine in the coroutine tree so that newParent becomes its parent (or sibling if $insertLeft or $insertRight is specified) (the coroutines are specified by a pointer to the $coroutine record in the pointer form and by name in the string form). It is an error if either coroutine is Zero. Valid ctrlBits bits are errorOK, $nonRecursive, $insertLeft, and $insertRight. errorOK suppresses error messages. $nonRecursive means that coroutine's children should not be moved along with it, but promoted in the coroutine tree to become children of coroutine's parent (as if $killCoroutine had been called on coroutine with the $nonRecursive bit set; see Section 1.208). If $insertLeft or $insertRight is set (the effect is undefined if both are set), then coroutine is inserted in the tree to the left or right, respectively, of parent, instead of being made a child of parent. $moveCoroutine returns true if successful, false otherwise.

1.234. msgMe

```
COMPILETIME
BITS
<macro>      msgMe;
```

Table 1.234-1. msgMe

msgMe is a bit that specifies that the caller of errMsg is to be indicated along with the error message. It may be passed to errMsg.

1.235. msgMyCaller

```
COMPILETIME
BITS
<macro>     msgMyCaller;
```

Table 1.235-1. msgMyCaller

msgMe is a bit that specifies that the caller of the caller of errMsg is to be indicated along with the error message. It may be passed to errMsg.

1.236. new

```
SPECIAL
POINTER
PROCEDURE   new         (CLASS  c;
                         OPTIONAL POINTER($area) area);

SPECIAL
PROCEDURE   new         (PRODUCES LONG ARRAY(*) a;
                         OPTIONAL LONG INTEGER l1,u1;
                         OPTIONAL POINTER($area) area;
                         OPTIONAL STRING aryName;
                         OPTIONAL INTEGER typeCode);

SPECIAL
PROCEDURE   new         (PRODUCES LONG ARRAY(*,*) a;
                         OPTIONAL LONG INTEGER
                             l1,u1,l2,u2;
                         OPTIONAL POINTER($area) area;
                         OPTIONAL STRING aryName;
                         OPTIONAL INTEGER typeCode);
```

Table 1.236-1. new (Generic) (continued)

```
SPECIAL
PROCEDURE    new          (PRODUCES LONG ARRAY(*,*,*) a;
                           OPTIONAL LONG INTEGER
                               l1,u1,l2,u2,l3,u3;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);

SPECIAL
PROCEDURE    new          (PRODUCES LONG ARRAY(*) a;
                           OPTIONAL INTEGER l1,u1;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);

SPECIAL
PROCEDURE    new          (PRODUCES LONG ARRAY(*,*) a;
                           OPTIONAL INTEGER l1,u1,l2,u2;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);

SPECIAL
PROCEDURE    new          (PRODUCES LONG ARRAY(*,*,*) a;
                           OPTIONAL INTEGER
                               l1,u1,l2,u2,l3,u3;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);

SPECIAL
PROCEDURE    new          (PRODUCES ARRAY(*) a;
                           OPTIONAL INTEGER l1,u1;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);
```

Table 1.236-1. new (Generic) (continued)

```
SPECIAL
PROCEDURE    new          (PRODUCES ARRAY(*,*) a;
                           OPTIONAL INTEGER l1,u1,l2,u2;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);

SPECIAL
PROCEDURE    new          (PRODUCES ARRAY(*,*,*) a;
                           OPTIONAL INTEGER
                              l1,u1,l2,u2,l3,u3;
                           OPTIONAL POINTER($area) area;
                           OPTIONAL STRING aryName;
                           OPTIONAL INTEGER typeCode);

SPECIAL
POINTER
PROCEDURE    new          (MODULE m;
                           OPTIONAL BITS ctrlBits;
                           OPTIONAL POINTER($area) area);

POINTER
PROCEDURE    new          (STRING moduleName;
                           OPTIONAL BITS ctrlBits;
                           OPTIONAL POINTER($area) area);
```

Table 1.236-1. new (Generic) (end)

new is used to allocate new records, arrays, or data sections (collectively referred to as "chunks"). area specifies the area in which the newly allocated chunk is to be put.

The class form creates a new record of the class c and returns a pointer to it. All fields of the record are initialized to Zero. The returned pointer is of class c.

The module and string forms of new create a new nonbound data section for the module m or the module named by the string moduleName and return a pointer to it. All storage within the data section is cleared. The module's initial procedure (if any) is invoked before returning. In the module form, the returned pointer is of the same class as the module; in the string form, it is unclassified. The control section associated with the allocated data section is found as described in Section 12.2 of part I of the "MAINSAIL Language Manual".

The valid bits for ctrlBits in the module and string forms are the same as for the system procedure "bind". They are shown in Table 1.27-2, and apply to a module allocated by new in the same way as to a module allocated by bind.

The (long) array forms create new array elements for a (long) array and initialize the element values to Zero. li and ui are the lower and upper bounds of the ith dimension, respectively. If a bound of the array being allocated was declared as a constant, the compiler checks that the corresponding argument is the same constant. Any bound declared as a constant may be omitted, as long as all subsequent arguments are also omitted; omitted bounds are filled in by the compiler.

aryName gives the name of the array, and typeCode the type of the array. The compiler checks that typeCode is the same as the type of the array argument, unless the array argument is untyped. In practice, the programmer rarely specifies aryName or typeCode. If the array name is omitted, the compiler substitutes the name of the identifier used as the first argument. If typeCode is omitted, the compiler substitutes the type code for the type declared for the array; an error occurs if the array is untyped and typeCode is omitted.

```
CLASS circle (INTEGER xCoord,yCoord,radius);
POINTER(circle) ARRAY(*) ary;
INTEGER i;
...
new(ary,m,n); # create an array with bounds m TO n
FOR i := m UPTO n DOB
    ary[i] := new(circle); # create a circle record
    ary[i].xCoord := ary[i].yCoord := 10 * i;
    ary[i].radius := 100 END;
```

Example 1.236-2. Use of new

1.237. $newArea

```
POINTER($area)
PROCEDURE    $newArea     (STRING title;
                           OPTIONAL LONG BITS attr;
                           OPTIONAL LONG INTEGER
                               strSpcChars);
```

Table 1.237-1. $newArea

$newArea returns a pointer to a new area. As described in Chapter 20 of part I of the "MAINSAIL Language Manual", title is the area's title, and attr the area's attributes; valid attr bits are shown in Table 1.237-2.

attr Bit	Description
$collectableChkSpc	collect area's chunks
$compactableChkSpc	compact area's chunks
$collectableStrSpc	collect area's string text
$noCollectablePtrs	this area contains no pointers into $collectableChkSpc areas
$noCompactablePtrs	this area contains no pointers into $compactableChkSpc areas
$noCollectableStrs	this area contains no string dscrs into $collectableStrSpc areas

Table 1.237-2. $newArea attr Bits

If attr is not specified, the default is that collections and compactions do NOT occur in the allocated area.

strSpcChars specifies the size in characters of string space, in characters, to allocate (if string space is needed); it should be specified only if an unusually small amount of string space (on the order of 2000 characters or less) is expected to be required (specifying a large strSpcChars if a lot of string space is needed has undesirable effects; it is better to take the default in this case). Extra string space is allocated as needed, so strSpcChars need not be exact.

1.238. $newException

```
STRING
PROCEDURE    $newException;
```

Table 1.238-1. $newException

$newException returns a string consisting of a unique decimal integer followed by a colon. The string may be concatenated to the front of another string describing an exception to produce a unique exception name, or may be used as an exception name by itself. To avoid conflicts with names created in these ways, users should avoid choosing an exception name that

begins with a decimal integer and a colon unless that prefix was obtained by calling $newException.

Values returned by $newException may vary from execution to execution of the same program.

1.239. newPage

```
ADDRESS
PROCEDURE    newPage      (OPTIONAL LONG INTEGER numPages;
                           OPTIONAL INTEGER pageCode;
                           OPTIONAL BITS ctrlBits);

ADDRESS
PROCEDURE    newPage      (OPTIONAL INTEGER numPages;
                           OPTIONAL INTEGER pageCode;
                           OPTIONAL BITS ctrlBits);
```

Table 1.239-1. newPage (Generic)

MAINSAIL divides memory into fixed-sized pages. newPage allocates and returns the address of a page for a program's use.

newPage returns the address of the first of numPages consecutive free pages, and marks them busy. If numPages is less than or equal to zero, one page is allocated. pageCode should be 0; other values are for system use, and the effect of their use in a user program is undefined.

The size of a page is the machine-dependent value $pageSize, which is the number of storage units in a page.

The only valid ctrlBits are errorOK and $doNotClear. If the pages cannot be allocated, an error occurs unless ctrlBits has errorOK set, in which case nullAddress is returned. The pages are cleared unless $doNotClear is set (for efficiency), in which case their contents are initially unspecified.

pageDispose is used to release pages.

```
ADDRESS a;
a := newPage;
...
pageDispose(a)
```

Example 1.239-2. Use of newPage

1.240. $newRecords

```
POINTER
PROCEDURE    $newRecords (POINTER p;
                          STRING linkFieldName;
                          LONG INTEGER numRecords;
                          OPTIONAL BITS ctrlBits;
                          OPTIONAL POINTER($area) area);

POINTER
PROCEDURE    $newRecords (POINTER p;
                          INTEGER linkFieldDspl;
                          LONG INTEGER numRecords;
                          OPTIONAL BITS ctrlBits;
                          OPTIONAL POINTER($area) area);


BOOLEAN
PROCEDURE    $newRecords (POINTER p;
                          POINTER LONG ARRAY(*) ary;
                          OPTIONAL LONG INTEGER numRecords;
                          OPTIONAL BITS ctrlBits;
                          OPTIONAL POINTER($area) area);
```

Table 1.240-1. $newRecords (Generic)

$newRecords may be used to allocate more than one record at a time. If there are many records to be allocated, $newRecords is more efficient than repeated calls to new. The pointer forms of $newRecords return a pointer to the first record in the allocated linked list of records. The records allocated are of p's class. numRecords records are allocated; in the array form, if numRecords is less than or equal to 0L, one record is allocated for each element of the array.

The string form links the records together through a pointer link field of name linkFieldName in each record. The array form sets the elements of ary to point to the allocated records, starting at the lower bound of ary. Any unused elements are unaffected. The integer form allows the link field to be specified by its offset in the record rather than by the name of the field. It is more efficient than the string form.

The only valid ctrlBits bit is errorOK; if set, an error message is suppressed if space cannot be allocated for the records. An error occurs and a Zero value is returned if:

- p is Zero.
- p does not point to a valid record.
- linkFieldName is not the name of a pointer field in p's class.
- ary is Zero.
- Space cannot be allocated for the records.
- numRecords is greater than the number of elements in ary.

area specifies the area in which the newly allocated records are put.

1.241. newScratch

```
ADDRESS
PROCEDURE   newScratch   (INTEGER n);

ADDRESS
PROCEDURE   newScratch   (LONG INTEGER n);
```

Table 1.241-1. newScratch (Generic)

newScratch returns the address of an area of memory of n storage units, the contents of which are initially cleared. An error occurs if the space cannot be allocated or if n is not positive.

scratchDispose is used to dispose of scratch space.

```
ADDRESS a;
a := newScratch(2);
    # get two storage units of scratch space
```

Example 1.241-2. Use of newScratch

1.242. $newScratchChars

```
CHARADR
PROCEDURE    $newScratchChars
                        (INTEGER chars);

CHARADR
PROCEDURE    $newScratchChars
                        (LONG INTEGER chars);
```

Table 1.242-1. $newScratchChars (Generic)

$newScratchChars allocates enough scratch memory to contain chars characters. The contents are initially clear. An error occurs if the storage cannot be allocated.

1.243. newString

```
$BUILTIN
STRING
PROCEDURE    newString    (CHARADR c;
                           INTEGER n);
```

Table 1.243-1. newString

newString is used to create a string descriptor.

newString returns a string of length n of which the first character is at the location given by c. The string so created is not subject to garbage collection if it is not in MAINSAIL's string space, e.g., if it is in storage allocated by a call to newPage or newScratch. If it is desired to reuse the storage to create new strings or to make the string subject to garbage collection, the procedure $getInArea should be used.

If c is nullCharadr or n less than or equal to 0, the result is "".

For example, "t := newString(cvc(s),length(s))" is equivalent to "t := s"; i.e., the string descriptor for t is a copy of the string descriptor for s.

1.244. newUpperBound

```
SPECIAL
PROCEDURE    newUpperBound
                          (MODIFIES ARRAY(*) a;
                           INTEGER n;
                           OPTIONAL POINTER($area) area);

SPECIAL
PROCEDURE    newUpperBound
                          (MODIFIES LONG ARRAY(*) a;
                           INTEGER n;
                           OPTIONAL POINTER($area) area);

SPECIAL
PROCEDURE    newUpperBound
                          (MODIFIES LONG ARRAY(*) a;
                           LONG INTEGER n;
                           OPTIONAL POINTER($area) area);
```

Table 1.244-1. newUpperBound (Generic)

newUpperBound adjusts the upper bound of a one-dimensional array.

A new array is allocated with lower bound the same as a's lower bound and upper bound given by n. newUpperBound then copies as many elements from a as will fit into the new array; if the new array is larger, the extra elements are initialized to Zero. The old array a is disposed, and the newly allocated array replaces it.

It is an error if a is nullArray, if n is less than a's lower bound, or if a was declared with a constant upper bound.

area specifies the area in which the new (copied) array is put, which need not be the same as the area in which the array was originally located. If area is not sepecified, the copied array is allocated in the same area as the old one.

```
IF i > currentBound THEN
    newUpperBound(a,currentBound := i)
```

Example 1.244-2. Use of newUpperBound

1.245. $noCollectablePtrs

```
COMPILETIME
LONG BITS
<macro>     $noCollectablePtrs;
```

Table 1.245-1. $noCollectablePtrs

$noCollectablePtrs is a bit that specifies that an area will have no pointers into areas where chunks are collected. This saves time during garbage collections because the collector does not have to examine pointers into other areas, but if there is indeed a pointer into an area that is collected, the result is undefined. This bit may be passed to $newArea.

An area that will contain a data section must not be marked $noCollectablePtrs.

1.246. $noCollectableStrs

```
COMPILETIME
LONG BITS
<macro>     $noCollectableStrs;
```

Table 1.246-1. $noCollectableStrs

$noCollectableStrs is a bit that specifies that an area will have no string descriptors pointing into areas where strings are collected. This saves time during garbage collections because the collector does not have to examine string descriptors into other areas, but if there is indeed a descriptor pointing into an area that is collected, the result is undefined. This bit may be passed to $newArea.

1.247. $noCompactablePtrs

```
COMPILETIME
LONG BITS
<macro>     $noCompactablePtrs;
```

Table 1.247-1. $noCompactablePtrs

$noCompactablePtrs is a bit that specifies that an area will have no pointers into areas where chunks are compacted (i.e., moved around). This saves time during compactions because the compactor does not have to update pointers into other areas, but if there is indeed a pointer into an area that is compacted, the result is undefined. This bit may be passed to $newArea.

An area that will contain a data section must not be marked $noCompactablePtrs.

1.248. nextAlpha

```
$BUILTIN
INTEGER
PROCEDURE    nextAlpha    (INTEGER char);
```

Table 1.248-1. nextAlpha

nextAlpha returns the character code of the alphabetically next character (same case) after that with character code char. It is undefined if char is not the character code for one of the lowercase letters "a" through "y" or for one of the uppercase letters "A" through "Y". nextAlpha is independent of the underlying character set.

```
nextAlpha('y') = 'z'
nextAlpha('M') = 'N'
nextAlpha('Z') is undefined
```

Example 1.248-2. Use of nextAlpha

1.249. $noHandler

```
COMPILETIME
BITS
<macro>      $noHandler;
```

Table 1.249-1. $noHandler

$noHandler is a bit that indicates that an exception returned because $returnIfNoHandler was set in the call to $raise for the exception and no handler handled it. It can be produced by $raise.

1.250. $nonRecursive

```
COMPILETIME
BITS
<macro>      $nonRecursive;
```

Table 1.250-1. $nonRecursive

$nonRecursive is a bit that specifies that the descendants of a coroutine are not killed along with the coroutine. It may be passed to $killCoroutine, $moveCoroutine, and $resumeCoroutine.

1.251. noResponse

```
COMPILETIME
BITS
<macro>      noResponse;
```

Table 1.251-1. noResponse

noResponse is a bit that specifies that no response is to be read from cmdFile. It may be passed to cmdMatch and errMsg. It may be tested in $exceptionBits; it is set if $raise was called from a call to errMsg with the noResponse bit set.

1.252. $noTranslate

```
COMPILETIME
BITS
<macro>      $noTranslate;
```

Table 1.252-1. $noTranslate

$noTranslate is a bit that suppresses translation to or from the PDF character set. It may be passed to $characterRead and $characterWrite.

1.253. $nulChar

```
COMPILETIME
INTEGER
<macro>      $nulChar;
```

Table 1.253-1. $nulChar

$nulChar is the character code for the null character.

1.254. $nullArrayExcpt

```
# system variable
STRING $nullArrayExcpt;
```

Table 1.254-1. $nullArrayExcpt

$nullArrayExcpt is a predefined exception that is raised when a nullArray error (nullArray used for element or pseudo-field access) occurs in code with runtime checking enabled (see Section 15.2 of part I of the "MAINSAIL Language Manual").

1.255. $nullCallExcpt

```
# system variable
STRING $nullCallExcpt;
```

Table 1.255-1. $nullCallExcpt

$nullCallExcpt is a predefined exception that is raised when a nullPointer call error (nullPointer used for procedure field access) occurs in code with runtime checking enabled (see Section 15.2 of part I of the "MAINSAIL Language Manual").

1.256. $nullPointerExcpt

```
# system variable
STRING $nullPointerExcpt;
```

Table 1.256-1. $nullPointerExcpt

$nullPointerExcpt is a predefined exception that is raised when a nullPointer error (nullPointer used for data field access) occurs in code with runtime checking enabled (see Section 15.2 of part I of the "MAINSAIL Language Manual").

1.257. octal

```
COMPILETIME
BITS
<macro>     octal;
```

Table 1.257-1. octal

octal is a bit that specifies that an octal string representation is input to or output from the procedure to which it is passed. It may be passed to cvb, cvlb, cvs, and $removeBits. It may be returned by $preferredRadix.

1.258. omit

```
COMPILETIME
BITS
<macro>      omit;
```

Table 1.258-1. omit

omit is a bit that specifies that no result string is to be returned. It may be passed to scan.

1.259. open

```
BOOLEAN
PROCEDURE    open          (PRODUCES POINTER(textFile) f;
                            STRING fileName;
                            BITS openBits;
                            OPTIONAL LONG INTEGER fileSize);

BOOLEAN
PROCEDURE    open          (PRODUCES POINTER(dataFile) f;
                            STRING fileName;
                            BITS openBits;
                            OPTIONAL LONG INTEGER fileSize);
```

Table 1.259-1. open (Generic)

"open" is used to "open" a file, i.e., to make the file available for input and/or output. The predeclared classes textFile and dataFile are explained in Section 18.2 of part I of the "MAINSAIL Language Manual".

A file with the name fileName is opened in accordance with the bits specified in openBits. If the file is successfully opened, a pointer to the file is produced in f (which is used for later access to the file) and the open procedure returns true. If the file is not successfully opened, f is set to nullPointer, and the procedure returns false.

It is an error if fileName is the null string (unless the "prompt" bit is set in openBits).

fileSize specifies the size of the file as the number of characters if a text file, and the number of storage units otherwise. It is relevant only if the file is being created, and only for certain file formats. The MAINSAIL runtime system extends a file open for output if data are written beyond the end of the file; therefore, fileSize need never be specified.

The bits constants shown in Table 1.259-2 are valid for openBits. PDF I/O is described in detail in Chapter 21 of part I of the "MAINSAIL Language Manual".

If random is specified, but neither input nor output, then both input and output are assumed. If output is specified, but not random, then create is assumed. That is, a sequential file opened for output has to be a file that does not already exist; specifying create would be redundant information.

The permissible combinations of input, output, random, and create (after the default rules have been applied) are shown in Table 1.259-3.

```
        (sequential) input
create  (sequential) output
        random       input
        random       output
create  random       output
        random       input and output
create  random       input and output
```

Table 1.259-3. Possible Combinations of openBits Bits Constants

If errorOK is not set and the file cannot be opened, an error message is issued and the user may type either:

- a new file name to be used, or
- <eol> to specify the same file name, or
- =<file name>, in which case the user is prompted whether or not to enter <file name> as a logical name for the original file name. If the user answers affirmatively, the logical name is established and the original file name is tried again.

create	Create a new file. If this bit is not set, it is an error if the file does not already exist.
random	Allow random access. If this bit is not set, it is an error to attempt to call relPos or setPos for the file. This bit should also be set if the same file is to be closed and reopened for random access in the future.
input	Allow read access.
output	Allow write access.
prompt	fileName is really a prompt to be written to logFile. After writing the prompt, read the fileName from cmdFile.
keepNul	Each implementation has a "null" character that is normally discarded when read from a text file. keepNul means do not discard any null characters from this file.
delete	Delete the file when it is closed.
alterOK	Permission is normally requested from cmdFile when an existing file is deleted or altered. alterOK suppresses the request for permission and performs the operation silently (unless an error occurs and errorOK is not set).
errorOK	If the operating system is unable to carry out the open as requested, an error message is by default written to logFile and a new file name read from cmdFile. If errorOK is set, the error message is suppressed, f is set to nullPointer, and open returns false.

Table 1.259-2. Predefined Bits Constants for openBits (continued)

```
$unbuffered
          Do not allocate a buffer for the file (this bit
          may be ignored for some file types, e.g., memory
          files, which are necessarily buffered).  Input
          and output must be performed by means of
          $pageRead, $pageWrite, $storageUnitRead,
          $storageUnitWrite, $characterRead,
          $characterWrite, fldRead (textFile form only)
          and the MAINSAIL Structure Blaster.  The
          procedures setPos, getPos, relPos, and close may
          also be called for an unbuffered file, but the
          use of other I/O procedures (e.g., read, write,
          scan, etc.) generates an error.  If large
          amounts of data are to be read or written at
          once, the use of the $unbuffered bit may result
          in a substantial speed increase for I/O.  If the
          file is being created, the random bit should be
          set if the $unbuffered bit is set to ensure that
          unbuffered I/O can be performed on the file.

$pdf      Open the file for PDF I/O.

$useOriginalFileName
          Do not look up logical names or use a
          searchpath; use the file name specified.
```

Table 1.259-2. Predefined Bits Constants for openBits (end)

1.260. openLibrary

```
BOOLEAN
PROCEDURE   openLibrary (STRING fileName;
                         OPTIONAL BITS ctrlBits);
```

Table 1.260-1. openLibrary

openLibrary opens the MAINSAIL objmod library file named fileName. After the library has been opened, it takes place in executable objmod searches. The only valid ctrlBits bit is

```
    POINTER(textFile) f;
    ...
    open(f,"notes",input);

Open the file named "notes" for text input (it must
already exist).  Use f for subsequent references to the
file.

    POINTER(textFile) f;
    ...
    open(f,"Output file: ",create!random!output!prompt);

The prompt is written to logFile, and then a fileName is
read from cmdFile.  A new file is created for random text
output.  Only write access is allowed.  If creation of the
new file requires that an existing file be deleted,
permission must be obtained before proceeding.
```

Example 1.259-4. Use of open

errorOK. If errorOK is set, false is returned if the library cannot be opened. Otherwise, a message is written to logFile, and a new library name read from cmdFile.

Libraries are searched in order from most recently opened to least recently opened. If two libraries contain a module with the same name, the one in the more recently opened library is found by a module search.

An open library file remains open during execution until closeLibrary is called for the library. If the same library name is given to a subsequent call to openLibrary, the library file is not actually opened again. On an operating system where file names are not case sensitive (i.e., where "$attributes NTST $fileNamesAreCaseSensitive" is true), library names are compared caselessly.

1.261. output

```
COMPILETIME
BITS
<macro>      output;
```

Table 1.261-1. output

output is a bit that specifies that output operations are to be allowed on the file that is being opened. It may be passed to $createUniqueFile, open, and $reOpen.

1.262. $overheadPercentExitValue and $overheadTooHighExcpt

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

When MAINSAIL runs low on memory, it may perform garbage collection and module swapping more and more frequently, until the program spends virtually all of its time on memory management and none on useful work. This behavior is known as "thrashing". To detect thrashing, the system variable $overheadPercentExitValue is provided.

```
# system variable
INTEGER $overheadPercentExitValue;
```

Table 1.262-1. $overheadPercentExitValue

A program may set $overheadPercentExitValue to any value in the range 1 to 100, inclusive (a value of 0 disables $overheadPercentExitValue checking). When MAINSAIL attempts to allocate memory and the percentage of time spent in memory management exceeds the value, the exception $overheadTooHighExcpt is raised. If the exception is handled, MAINSAIL continues execution (the handler may free up some memory, or modify $overheadPercentExitValue); otherwise, MAINSAIL exits. Before the $overheadTooHighExcpt is raised, MAINSAIL sets the value of $overheadPercentExitValue to 0. The user program must explicitly set $overheadPercentExitValue to a non-zero value to have the exception raised again (assuming the program handles the exception).

```
# system variable
STRING $overheadTooHighExcpt;
```

Table 1.262-2. $overheadTooHighExcpt

When $overheadTooHighExcpt is raised, $exceptionStringArg1 is cvs($overheadPercentExitValue) before it was set to 0; $exceptionStringArg2 is cvs(numPages), where numPages is the number of pages that need to be allocated. If the program handles the exception, and sets $overheadPercentExitValue to a non-zero value, the exception will not be raised again for the particular allocation of numPages that just caused it to be raised, but it could be raised for subsequent allocations.

Once the overhead percent reaches a certain value, it may take some time for it to decrease significantly, so the handler of $overheadTooHighExcpt should probably not set $overheadPercentExitValue back to its original value since this is likely to cause the exception to be raised again very soon (unless the handler freed a large amount of memory, or the program runs for a significant amount of time before needing much more memory).

1.263. pageDispose

```
PROCEDURE   pageDispose (ADDRESS pageAdr;
                         OPTIONAL LONG INTEGER numPages);

PROCEDURE   pageDispose (ADDRESS pageAdr;
                         OPTIONAL INTEGER numPages);
```

Table 1.263-1. pageDispose (Generic)

pageDispose disposes of pages obtained with newPage.

pageDispose releases numPages pages starting at the page that contains pageAdr (i.e., pageAdr need not be the address of the start of the page). Nothing happens if pageAdr is nullAddress.

1.264. $pageRead

```
LONG INTEGER
PROCEDURE    $pageRead    (POINTER(dataFile) f;
                           ADDRESS memAdr;
                           LONG INTEGER startPage;
                           OPTIONAL LONG INTEGER numPages;
                           OPTIONAL BITS ctrlBits);
```

Table 1.264-1. $pageRead

$pageRead reads numPages pages of data from f to the address memAdr, starting at page startPage in the file (the first page is numbered zero). If numPages is less than or equal to zero, one page is read.

Unless f is opened for random access, $pageRead succeeds only if the file is positioned at the start of the page to be read; otherwise, an error message is given.

The only valid ctrlBits bit is errorOK. If not set, an error occurs if $pageRead cannot read the amount of data requested. The value returned by $pageRead is the number of storage units read.

$pageRead may be especially efficient if the file was opened with the $unbuffered bit.

$storageUnitRead and $characterRead are other procedures used to read large amounts of data from a file with a single procedure call.

1.265. $pageSize

```
COMPILETIME
INTEGER
<macro>      $pageSize;
```

Table 1.265-1. $pageSize

$pageSize is the operating-system-dependent number of storage units per page. A page is the amount of memory returned by newPage, or read by $pageRead, or written by $pageWrite.

1.266. $pageWrite

```
PROCEDURE   $pageWrite   (POINTER(dataFile) f;
                          ADDRESS memAdr;
                          LONG INTEGER startPage;
                          OPTIONAL LONG INTEGER numPages);
```

Table 1.266-1. $pageWrite

$pageWrite writes numPages pages of data from the address memAdr to f, starting at page startPage in the file (the first page is numbered zero). If numPages is less than or equal to zero, one page is written.

Unless f is opened for random access, $pageWrite succeeds only if the file is positioned at the start of the page to be written; otherwise, an error message is given.

$pageWrite may be especially efficient if the file was opened with the $unbuffered bit.

$storageUnitWrite and $characterWrite are other procedures used to write large amounts of data to a file with a single procedure call.

1.267. PDF Low-Level Procedures

The following procedures are provided by the PDFMOD package, which allows low-level manipulation of PDF data in memory and which is documented in detail in the "MAINSAIL Utilities User's Guide":

pdfBoRead	pdfBoWrite	pdfbRead
pdfbWrite	pdfCharRead	pdfChars
pdfCharWrite	pdfcRead	pdfcWrite
pdfDeInit	pdfFldRead	pdfInit
pdfiRead	pdfiWrite	pdfLbRead
pdfLbWrite	pdfLiRead	pdfLiWrite
pdfLrRead	pdfLrWrite	pdfRead
pdfrRead	pdfrWrite	pdfWrite

1.268. $pdf

```
COMPILETIME
BITS
<macro>      $pdf;
```

Table 1.268-1. $pdf

$pdf is a bit that specifies that PDF (Portable Data Format) I/O is to be performed on the file being opened. It may be passed to $createUniqueFile, open, and $reOpen.

1.269. $platformNameAbbreviation

```
STRING
<macro>      $platformNameAbbreviation;
```

Table 1.269-1. $platformNameAbbreviation

$platformNameAbbreviation is the abbreviation for the name of the target platform. Abbreviations are shown in Table B-1 of part I of the "MAINSAIL Language Manual".

1.270. $platformNameFull

```
STRING
<macro>      $platformNameFull;
```

Table 1.270-1. $platformNameFull

$platformNameFull is the full name of the target platform. Platform names are shown in Table B-1 of part I of the "MAINSAIL Language Manual".

1.271. $platformNumber

```
INTEGER
<macro>     $platformNumber;
```

Table 1.271-1. $platformNumber

$platformNumber is the number the target platform. Platform numbers are shown in Table B-1 of part I of the "MAINSAIL Language Manual". Unlike $systemNumber, $platformNumber is evaluated at runtime, not at compiletime, and so cannot govern conditional compilation.

1.272. $preferredRadix

```
COMPILETIME
BITS
<macro>     $preferredRadix;
```

Table 1.272-1. $preferredRadix

$preferredRadix is the target system's "natural" radix (usually the radix used in the manufacturer's documentation and/or instruction-level debugger) for representing (long) bits, address, charadr, and pointer values. Possible values for $preferredRadix are hex and octal.

1.273. prevAlpha

```
$BUILTIN
INTEGER
PROCEDURE   prevAlpha   (INTEGER char);
```

Table 1.273-1. prevAlpha

prevAlpha returns the character code of the alphabetically previous character (same case) to that with character code char. It is undefined if char is not the character code for one of the lowercase letters "b" through "z" or for one of the uppercase letters "B" through "Z". prevAlpha is independent of the underlying character set.

```
prevAlpha('z') = 'y'
prevAlpha('b') = 'a'
prevAlpha('B') = 'A'
prevAlpha('A') is undefined
```

Example 1.273-2. Use of prevAlpha

1.274. proceed

```
COMPILETIME
BITS
<macro>      proceed;
```

Table 1.274-1. proceed

proceed is a bit that specifies that the scanning is to stop when a character that is not one of the scan control characters is reached. It may be passed to scan.

1.275. $processorNameAbbreviation

```
STRING
<macro>      $processorNameAbbreviation;
```

Table 1.275-1. $processorNameAbbreviation

$processorNameAbbreviation is the abbreviation for the name of the target processor. Abbreviations are shown in Table B-3 of part I of the "MAINSAIL Language Manual".

1.276. $processorNameFull

```
STRING
<macro>      $processorNameFull;
```

Table 1.276-1. $processorNameFull

$processorNameFull is the full name of the target processor. Processor names are shown in Table B-3 of part I of the "MAINSAIL Language Manual".

1.277. $processorNumber

```
COMPILETIME
INTEGER
<macro>      $processorNumber;
```

Table 1.277-1. $processorNumber

$processorNumber is the number the target processor. Processor numbers are shown in Table B-3 of part I of the "MAINSAIL Language Manual".

1.278. $programInterface

```
COMPILETIME
BITS
<macro>      $programInterface;
```

Table 1.278-1. $programInterface

$programInterface is a bit that specifies that $useProgramInterface is to be true at the start of the initial procedure of the invoked module. It may be passed to bind, $invokeModule, and new.

1.279. $programName

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
# system variable
STRING      $programName;
```

Table 1.279-1. $programName

$programName is set to the name of the currently executing MAINSAIL bootstrap file or the string typed by the user in the command that invoked MAINSAIL, if available from the operating system; otherwise, it is set to the null string. The effect of modifying $programName is undefined.

1.280. prompt

```
COMPILETIME
BITS
<macro>     prompt;
```

Table 1.280-1. prompt

prompt is a bit that specifies that the given file name is to be used as a prompt for a file name rather than as a file name. It may be passed to open.

1.281. $queryFileCacheParms

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
BOOLEAN
PROCEDURE    $queryFileCacheParms
                          (POINTER(file) f;
                           OPTIONAL PRODUCES LONG BITS
                               attributes;
                           OPTIONAL PRODUCES LONG INTEGER
                               requestedMinSize,
                               requestedMaxSize,
                               currentSize;
                           OPTIONAL PRODUCES INTEGER
                               requestedHitPercent,
                               currentHitPercent;
                           OPTIONAL BITS ctrlBits);
```

Table 1.281-1. $queryFileCacheParms

$queryFileCacheParms returns information about the cache associated with the file f, except that if f is nullPointer, then information about the global cache is returned. If f is not nullPointer and f cannot be cached, e.g., if it is a sequential, unbuffered, or mapped disk file, an error occurs and $queryFileCacheParms returns false.

If f is a globally cached file or if f is nullPointer, information about the global cache is returned. If f is a privately cached file, information about the private cache is returned. If f is not cached, all produces parameters are set to Zero and $queryFileCacheParms returns true. Figure 1.281-2 shows the produces parameters and their meaning.

The only valid ctrlBits is errorOK. An error message is generated if an error occurs and errorOK is not specified.

attributes	If f is not nullPointer, then attributes returns information about how the file is cached, i.e., globally cached (the predefined bit $globallyCached is set), privately cached (the predefined bit $privatelyCached is set), or not cached (attributes is Zero).
requestedMinSize	the requested minimum number of buffers in the LRU list
requestedMaxSize	the requested maximum number of buffers in the LRU list
currentSize	the current number of buffers in the LRU list
requestedHitPercent	the requested hit percent
currentHitPercent	the current hit percent

Figure 1.281-2. Information Produced by $queryFileCacheParms

1.282. $raise

```
PROCEDURE   $raise      (OPTIONAL STRING exceptionName,
                             exceptionStringArg1,
                             exceptionStringArg2;
                         OPTIONAL POINTER
                             exceptionPointerArg;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL BITS
                             resultBits;
                         OPTIONAL POINTER($coroutine)
                             raiseeCoroutine);
```

Table 1.282-1. $raise

$raise either causes the occurrence of an exception or propagates the current exception, depending on the value of exceptionName.

If exceptionName is not the null string, $raise causes an occurrence of the exception denoted by exceptionName. In this case, exceptionStringArg1, exceptionStringArg2, and exceptionPointerArg are assumed to contain extra information about the exception that a handler can access by means of the system procedures $exceptionStringArg1, $exceptionStringArg2, and $exceptionPointerArg.

Table 1.282-2 shows the valid predefined bits constants for ctrlBits. To require that control eventually be returned to the current point, an exception may be raised with the ctrlBits $cannotFallOut!$returnIfNoHandler.

Bit	Meaning
$cannotReturn	Do not permit a call to $raiseReturn for this exception. The exception must be handled by falling out of a handler.
$returnIfNoHandler	If no handler handles the exception, ignore the exception and return from $raise.
$cannotFallOut	Error occurs if handler attempts to handle the exception by falling out (or terminates with Done, Continue, or Return Statement).

Table 1.282-2. Predefined Bits Constants for $raise ctrlBits

Table 1.282-3 shows the valid predefined bits constants for resultBits.

Bit	Meaning
$noHandler	Control returned from $raise because $returnIfNoHandler was set in ctrlBits and there was no handler.

Table 1.282-3. Predefined Bits Constants for $raise resultBits

If exceptionName is the null string, the current exception is propagated to another handler and the parameters exceptionStringArg1, exceptionStringArg2, exceptionPointerArg, and ctrlBits are ignored. If there is no current exception, a system exception is raised.

raiseeCoroutine indicates the coroutine in which the exception should first be raised (the "raisee coroutine"). A Zero value denotes the current coroutine (the "raiser coroutine").

The exception may be either a user exception or a predefined exception.

1.283. $raiseReturn

```
PROCEDURE    $raiseReturn;
```

Table 1.283-1. $raiseReturn

$raiseReturn terminates the execution of the current exception's handler and continues execution at the place where the current exception occurred. If no exception is active, an error occurs and a system exception is raised. If the current exception was not caused by means of an explicit call to the system procedure $raise, another exception is raised. Calls to $raiseReturn may appear outside the text of a handler, i.e., within a procedure called by a handler. All active procedures invoked as a result of a handler's execution are terminated by a call to $raiseReturn.

1.284. $rand

$rand returns the next pseudo-random number produced by one of the algorithms in $ranMod, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.285. random

```
COMPILETIME
BITS
<macro>      random;
```

Table 1.285-1. random

random is a bit that specifies that random I/O is to be allowed on the file that is being opened. It may be passed to $createUniqueFile, open, and $reOpen.

1.286. rcRead

```
$BUILTIN
INTEGER
PROCEDURE    rcRead       (MODIFIES STRING s);
```

Table 1.286-1. rcRead

rcRead ("reverse cRead") returns the character code of the last character of the string s, and then removes that character from the string.

If s is "", -1 is returned. -1 is not a valid character code.

```
INTEGER t; STRING s;
s := "abc";
t := rcRead(s);
    # Now t = 'c', s = "ab"

...

STRING PROCEDURE reverse (STRING s);
BEGIN # reverse characters of s
STRING r;
r := "";
WHILE s DO cWrite(r,rcRead(s));
RETURN(r) END
```

Example 1.286-2. Use of rcRead

1.287. rcWrite

```
PROCEDURE    rcWrite      (MODIFIES STRING s;
                           REPEATABLE INTEGER char);

PROCEDURE    rcWrite      (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE INTEGER char);
```

Table 1.287-1. rcWrite (Generic)

rcWrite ("reverse cWrite") concatenates the character char onto the front of the string s. In the area form, area specifies the destination area for the resulting string.

```
    s := "bc"; rcWrite(s,'a');    # s = "abc"

The "reverse" procedure of Example 1.286-2
could also be written with "rcWrite" replacing
"cWrite" and "cRead" replacing "rcRead":

    STRING PROCEDURE reverse (STRING s);
    BEGIN # reverse characters of s
    STRING r;
    r := "";
    WHILE s DO rcWrite(r,cRead(s));
    RETURN(r) END
```

Example 1.287-2. Use of rcWrite

1.288. read

```
PROCEDURE    read          (MODIFIES STRING s;
                            PRODUCES REPEATABLE BOOLEAN v);

PROCEDURE    read          (POINTER(textFile) f;
                            PRODUCES REPEATABLE BOOLEAN v);

PROCEDURE    read          (POINTER(dataFile) f;
                            PRODUCES REPEATABLE BOOLEAN v);

$BUILTIN
PROCEDURE    read          (MODIFIES ADDRESS a;
                            PRODUCES REPEATABLE BITS v);

PROCEDURE    read          (MODIFIES STRING s;
                            PRODUCES REPEATABLE BITS v);

PROCEDURE    read          (POINTER(dataFile) f;
                            PRODUCES REPEATABLE BITS v);

PROCEDURE    read          (POINTER(textFile) f;
                            PRODUCES REPEATABLE BITS v);

PROCEDURE    read          (POINTER(dataFile) f;
                            PRODUCES REPEATABLE INTEGER v);

PROCEDURE    read          (POINTER(textFile) f;
                            PRODUCES REPEATABLE INTEGER v);

PROCEDURE    read          (MODIFIES STRING s;
                            PRODUCES REPEATABLE INTEGER v);

$BUILTIN
PROCEDURE    read          (MODIFIES ADDRESS a;
                            PRODUCES REPEATABLE INTEGER v);

PROCEDURE    read          (POINTER(dataFile) f;
                            PRODUCES REPEATABLE LONG BITS v);
```

Table 1.288-1. read (Generic) (continued)

```
PROCEDURE    read        (POINTER(textFile) f;
                          PRODUCES REPEATABLE LONG BITS v);

PROCEDURE    read        (MODIFIES STRING s;
                          PRODUCES REPEATABLE LONG BITS v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE LONG BITS v);

PROCEDURE    read        (POINTER(dataFile) f;
                          PRODUCES REPEATABLE LONG INTEGER
                              v);

PROCEDURE    read        (POINTER(textFile) f;
                          PRODUCES REPEATABLE LONG INTEGER
                              v);

PROCEDURE    read        (MODIFIES STRING s;
                          PRODUCES REPEATABLE LONG INTEGER
                              v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE LONG INTEGER
                              v);

PROCEDURE    read        (POINTER(dataFile) f;
                          PRODUCES REPEATABLE LONG REAL v);

PROCEDURE    read        (POINTER(textFile) f;
                          PRODUCES REPEATABLE LONG REAL v);

PROCEDURE    read        (MODIFIES STRING s;
                          PRODUCES REPEATABLE LONG REAL v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE LONG REAL v);

PROCEDURE    read        (POINTER(dataFile) f;
                          PRODUCES REPEATABLE REAL v);
```

Table 1.288-1. read (Generic) (continued)

```
PROCEDURE    read        (POINTER(textFile) f;
                          PRODUCES REPEATABLE REAL v);

PROCEDURE    read        (MODIFIES STRING s;
                          PRODUCES REPEATABLE REAL v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE REAL v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE STRING v);

PROCEDURE    read        (POINTER(textFile) f;
                          PRODUCES REPEATABLE STRING s);

PROCEDURE    read        (MODIFIES STRING s;
                          PRODUCES REPEATABLE STRING v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE POINTER v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE ADDRESS v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE CHARADR v);

$BUILTIN
PROCEDURE    read        (MODIFIES ADDRESS a;
                          PRODUCES REPEATABLE BOOLEAN v);

PROCEDURE    read        (POINTER(dataFile) f;
                          PRODUCES REPEATABLE STRING s);

PROCEDURE    read        (POINTER(textFile) f;
                          POINTER($area) area;
                          PRODUCES REPEATABLE STRING s);
```

Table 1.288-1. read (Generic) (continued)

```
PROCEDURE    read          (POINTER(dataFile) f;
                            POINTER($area) area;
                            PRODUCES REPEATABLE STRING s);
```

Table 1.288-1. read (Generic) (end)

read reads values from an input file, a string, or a memory location.

The forms that read a boolean, (long) integer, (long) real, or (long) bits from a text file or string use a scan for the proper constant representation. As soon as the scan finds what could be the start of a constant of the desired type, it begins forming the value. All characters involved in the scan are removed from the source string or text file. Characters not involved in the scan are not removed; e.g., if the characters involved in the scan occur immediately before an eol, the eol is left in the source.

The forms of read that read a boolean from a text file or string scan for the string representation "TRUE" or "FALSE". Case is ignored. As soon as one of these string representations is found, or there are no more characters in the source string or text file, the scan stops. All scanned characters are removed from the source string or text file. The boolean value produced is true if the characters "TRUE" were found in the source; otherwise, it is false. The characters "TRUE" or "FALSE" need not be preceded or followed by a blank, tab, or end of line.

Numeric ((long) integer and (long) real) and (long) bits scans ignore dots, single quotes, and minuses that are not associated with valid digits. For example, if the string:

```
"These ' and - and . are ignored; the value read is 2"
```

is scanned for a numeric or bits, the "2" is found, and the symbols "'", "-", and "." are ignored since they are not associated with valid digits.

The "L" that follows long constants as written in a source program is not used by the scan. For example, if the string:

```
"123L 456"
```

is scanned for a (long) integer, 123 is returned, and the string becomes:

```
"L 456"
```

A (long) integer scan looks for one or more digits, possibly preceded by "-". For example, if the string:

```
"123. 456"
```

is scanned for an integer, 123 is returned, and the string becomes:

```
". 456"
```

On the other hand, if it were scanned for a real, then 123. would be returned, and the string would become:

```
" 456"
```

A (long) real scan accepts an integer constant. For example, if the string:

```
"123 456."
```

is scanned for a real, 123. is returned, and the string becomes:

```
" 456."
```

A bits scan accepts the standard representation, i.e., "'" optionally followed by a base letter, then digits. It also accepts a sequence of octal digits not preceded by "'". For example, if the string:

```
"1238 '456"
```

is scanned for a bits, '123 is returned, and the string becomes:

```
"8 '456"
```

The effect is undefined of reading a (long) integer or (long) real from a string or text file if the text scanned represents a value outside the MAINSAIL guaranteed range.

A string read from a string or file returns the next line by scanning for eol (end-of-line), which is then discarded. Characters in a data file are stored as described in Sections 1.82 and 1.104. If the file is opened for PDF I/O, characters may be translated to the host character set. In the area forms, area specifies the destination area for the resulting string.

Data types other than string read from a data file are removed from the file; i.e., the file position is updated to be immediately beyond the values read. This means that, in reading a non-string variable from a text file, the remainder of the line on which the value occurs is left in the file; see Example 1.288-2.

In all reads from a file or string, the result is Zero if no value is found when reading from the file or string (i.e., if end-of-file or end-of-string is encountered).

```
The following code fragment:

    INTEGER i; STRING s;
    ...
    read(cmdFile,i): read(cmdFile,s);

given the input line (read from cmdFile):

    16<eol>

sets i to the value 16, leaving the <eol> in cmdFile,
unread.  The next call to the textFile/string form of read
scans up to the (still unread) eol, discards it, and
therefore sets s to the value "".  Only one line is read
from cmdFile for both calls to read, since the first call
does not exhaust the input line.
```

Example 1.288-2. Integers Read from cmdFile

After reading from an address, the address is displaced to the location immediately following that from which the value was read. The result is undefined if the address is nullAddress, or if "displace(a,size(<data type of v>))" is undefined.

1.289. $registerException

```
PROCEDURE   $registerException
                          (STRING exceptionName;
                           OPTIONAL STRING comment;
                           OPTIONAL BITS ctrlBits;
                           OPTIONAL STRING arg);
```

Table 1.289-1. $registerException

$registerException "registers" the exception denoted by exceptionName; i.e., it adds the exception to the list of exceptions known to the system procedure errMsg. Distinctions between upper and lower case letters are ignored when comparing exceptionName to the strings denoting the previously registered exceptions.

```
    INTEGER i; REAL r; BITS b;
    ...
    read(inFile,i,r,b)

reads an integer into i, a real into r, and a bits into b.
If inFile is a text file, the file is scanned for the
string representations; if it is a data file, the proper
number of storage units are input.

    STRING s; INTEGER height,weight;
    ...
    s := "Height is 70 inches, weight 150 pounds.";
    read(s,height,weight)

The read picks the first two integers out of s, thereby
shortening s.  height becomes 70, weight 150, and s
" pounds.".

    ADDRESS a; INTEGER i; REAL r;
    ...
    read(a,i,r)

reads an integer and a real starting at memory location a,
and updates a to the value given by

    displace(a,size(integerCode) + size(realCode))
```

Example 1.288-3. Use of read

The string passed as the parameter comment is used only by errMsg. In response to "?", errMsg lists the registered exceptions to logFile with a description in the right margin of the effect of raising each exception. The description listed for an exception is the string that was passed as the comment parameter when the exception was registered.

Valid ctrlBits bits are useKeyWord and $doNotMatch.

If useKeyWord is set, then when the exception is specified in a response to errMsg, extra phrases in the response and extra words in the phrase preceding the extra phrases are ignored during the matching process and are instead set aside as an argument to the response. When errMsg raises the specified registered exception, that argument is passed as the argument exceptionStringArg1 to $raise.

If $doNotMatch is set, the exception is ignored when errMsg searches for a registered exception to raise. Exceptions with $doNotMatch set are listed in response to "?" in errMsg, but cannot be raised from errMsg, except by explicitly invoking the module RAISE to do so (see the "MAINSAIL Utilities User's Guide" for a description of RAISE). When an exception is registered more than once, no "duplicate registered exception" error is given if either the new instance or all previous instances of the exception were registered with $doNotMatch set. When an exception is deregistered, all instances of the exception are deregistered.

The same exception can be registered more than once without setting the $doNotMatch bit, if the subsequent calls to $registerException have the same arguments as the first call that did not set the $doNotMatch bit. In comparing string arguments, a caseless comparison is done.

An exception is removed from the list of registered exceptions only when $deregisterException has been called for that exception at least as often as $registerException was called without setting the $doNotMatch bit (i.e., a count is kept, incremented by $registerException and decremented by $deregisterException, and the exception is considered deregistered when the count reaches zero).

It is an error to try to register an exception that has the same name as another registered exception, if neither one was registered with the $doNotMatch bit set, and any of their corresponding arguments to $registerException are not equal.

arg is used only when listing the registered exceptions in response to "?" in errMsg. If arg is not Zero, then a blank followed by arg is written after the exception name. Usually arg is a string referred to in the $registerException comment argument.

1.290. relFileName

```
PROCEDURE   relFileName (STRING modName);
```

Table 1.290-1. relFileName

relFileName releases a module file name association previously established for modName by setFileName. It is an error if modName is not a valid module name, i.e., a one- to six-character identifier. No error occurs if no module file name association was established for modName.

```
MODULE m (PROCEDURE p);
...
setFileName("m","f");
p; # calls p in m contained in file f
...
unBind("m"); relFileName("m");
p; # calls p in m contained in default file
```

Example 1.290-2. Use of relFileName

1.291. relModName

```
PROCEDURE    relModName   (STRING dummyName);
```

Table 1.291-1. relModName

relModName releases a module name association previously established for dummyName by setModName. It is an error if modName is not a valid module name, i.e., a one- to six-character identifier. No error occurs if no module name association was established for dummyName.

1.292. relPos

```
BOOLEAN
PROCEDURE    relPos       (POINTER(file) f;
                           INTEGER n;
                           OPTIONAL BITS ctrlBits);
```

Table 1.292-1. relPos

relPos provides relative positioning within a random file.

"relPos(f,n,ctrlBits)" is equivalent to "setPos(f,getPos(f) + cvli(n),ctrlBits)".

relPos is restricted to integer ranges.

```
scan(f," ",omit); # skip to next blank in file f
relPos(f,2);      # ignore blank and next character
```

Example 1.292-2. Use of relPos

1.293. $removeBits

```
STRING
PROCEDURE   $removeBits (MODIFIES STRING s;
                         OPTIONAL BITS ctrlBits);
```

Table 1.293-1. $removeBits

$removeBits reads a bits representation from the beginning of s. $removeBits returns the null string if s does not begin with a valid string representation of a bits value; otherwise, it removes the longest prefix of s that represents a valid bits string (as if the procedure "read" had been called to read a bits value from s).

The valid ctrlBits bits are discard, formatted, binary, octal, and hex. If discard is set, initial blanks and tabs are removed from s before looking for the bits value. If formatted is set, the null string is returned if the bits value in s does not begin with the single quote ("'") character. If binary, octal, or hex is set, a string representing a value in the corresponding radix is removed unless overridden by an initial "'B", "'O", or "'H".

If s does not begin with a valid bits string representation (following blanks and tabs if discard is set), s is not altered.

1.294. $removeBoolean

```
STRING
PROCEDURE    $removeBoolean
                          (MODIFIES STRING s;
                           OPTIONAL BITS ctrlBits);
```

Table 1.294-1. $removeBoolean

$removeBoolean reads a boolean string representation from the beginning of s. $removeBoolean returns the null string if s does not begin with either "TRUE" or "FALSE" (case is ignored). Otherwise, it removes the character representation from s. The boolean string representation in s need not be followed by a blank, tab, or end-of-line; i.e., $removeBoolean looks for and removes from s only the characters "TRUE" or "FALSE".

The only valid ctrlBits bit is discard. If set, initial blanks and tabs are removed from s before looking for the boolean string representation.

If s does not begin with a valid boolean string representation (following blanks and tabs if discard is set), s is not altered.

1.295. $removeInteger

```
STRING
PROCEDURE    $removeInteger
                          (MODIFIES STRING s;
                           OPTIONAL BITS ctrlBits);
```

Table 1.295-1. $removeInteger

$removeInteger reads an integer representation from the beginning of s. $removeInteger returns the null string if s does not begin with a valid string representation of an integer value; otherwise, it removes the longest prefix of s that represents a valid integer string (as if the procedure "read" had been called to read an integer value from s).

The only valid ctrlBits bit is discard. If set, initial blanks and tabs are removed from s before looking for the integer value.

If s does not begin with a valid integer string representation (following blanks and tabs if discard is set), s is not not altered.

1.296. $removeLeadingBlankSpace

```
PROCEDURE    $removeLeadingBlankSpace
                          (MODIFIES STRING s);
```

Table 1.296-1. $removeLeadingBlankSpace

$removeLeadingBlankSpace removes leading tab and blank characters from s.

1.297. $removeMemMngModule

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
PROCEDURE    $removeMemMngModule
                          (POINTER dataSec);
```

Table 1.297-1. $removeMemMngModule

$removeMemMngModule unlocks a garabge collection interception module from memory so that it can be swapped out. More detail may be found at the description of $addMemMngModule.

1.298. $removeDateAndTime

```
STRING
PROCEDURE    $removeDateAndTime
                          (MODIFIES STRING s;
                           OPTIONAL BITS ctrlBits);
```

Table 1.298-1. $removeDateAndTime

$removeDateAndTime reads a date and time string from the beginning of s. If s begins with a valid substring representing a date, time, or date followed by time, $removeDateAndTime removes the substring from s and returns it. Otherwise, $removeDateAndTime does not alter s and returns the null string.

The only valid ctrlBits bit is discard. If set, initial blanks and tabs are removed from s before searching for the date and/or time string.

1.299. $removeReal

```
STRING
PROCEDURE    $removeReal (MODIFIES STRING s;
                          OPTIONAL BITS ctrlBits);
```

Table 1.299-1. $removeReal

$removeReal reads a real representation from the beginning of s. $removeReal returns the null string if s does not begin with a valid string representation of a real value; otherwise, it removes the longest prefix of s that represents a valid real string (as if the procedure "read" had been called to read a real value from s).

The only valid ctrlBits bit is discard. If set, initial blanks and tabs are removed from s before looking for the real value.

If s does not begin with a valid real string representation (following blanks and tabs if discard is set), s is not altered.

1.300. $removeTrailingBlankSpace

```
PROCEDURE    $removeTrailingBlankSpace
                          (MODIFIES STRING s);
```

Table 1.300-1. $removeLeadingBlankSpace

$removeTrailingBlankSpace removes trailing tab and blank characters from s.

1.301. $removeWord

```
STRING
PROCEDURE    $removeWord (MODIFIES STRING s);
```

Table 1.301-1. $removeWord

$removeWord first removes leading tab and blank characters from s. It then removes all characters up to the next tab or blank (or end of string); these characters constitute the "word". It then removes any additional leading tab or blank characters from s, and returns the word.

1.302. $rename

```
BOOLEAN
PROCEDURE    $rename      (STRING oldFileName,newFileName;
                           OPTIONAL BITS ctrlBits);
```

Table 1.302-1. $rename

$rename renames the existing file with the name oldFileName to have the name newFileName. The rename fails and an error is generated if oldFileName and newFileName do not use the same MAINSAIL device module (see Section 18.11 of part I of the "MAINSAIL Language Manual"), or if newFileName is the null string.

The only valid ctrlBits bits are errorOK, alterOK, and $useOriginalFileName. If errorOK is specified and the rename fails, $rename returns false. If errorOK is not specified and the rename fails, a prompt is written to logFile and oldFileName is read from cmdFile. If it is the null string, false is returned. Otherwise, another prompt is written to logFile and newFileName is read from cmdFile. $rename then attempts the rename again.

If alterOK is specified on a system without file version numbers (i.e., on a system where the $hasFileVersions bit is not set in $attributes), $rename attempts to delete the file with the name newFileName if it exists before renaming the file with the name oldFileName. $rename fails if such a deletion fails or if a file with the name newFileName exists and alterOK is not specified. On systems with file version numbers, alterOK is ignored; $rename attempts to create a new version of the file named newFileName.

If $useOriginalFileName is set, no logical name lookup or application of searchpaths is done; oldFileName and newFileName are used as specified.

Calling $rename with the name of an open file (as oldFileName or newFileName) has undefined effects on the program that has the file open.

1.303. $reOpen

```
BOOLEAN
PROCEDURE   $reOpen      (POINTER(file) f;
                          BITS openBits);
```

Table 1.303-1. $reOpen

$reOpen closes f, then reopens it (using f.name and the same file record) with the specified openBits (all bits described under open except "prompt" are valid). It returns true if the reopen was successful, false otherwise. This is useful, e.g., if a file opened read-only needs to be updated; it can be reopened with output set in openBits. The same file record is used, so there is no need to track down all pointers to it as there would be if individual calls to open and close were made. $reOpen also provides a way to reposition to the beginning of a sequentially opened file, since $reOpen always repositions the file to position 0L.

1.304. $reportAllVersions

```
COMPILETIME
BITS
<macro>      $reportAllVersions;
```

Table 1.304-1. $reportAllVersions

$reportAllVersions is a bit that specifies that all existing versions of a file are included in the output (applicable only where a file may have multiple versions). It may be passed to $directory.

1.305. reorder

reorder reorders the elements of an array according to the contents of a parallel index array. It is one of the procedures provided by the sorting package, SRTMOD, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.306. $resumeCoroutine

```
BOOLEAN
PROCEDURE    $resumeCoroutine
                        (OPTIONAL POINTER($coroutine) p;
                         OPTIONAL BITS ctrlBits);

BOOLEAN
PROCEDURE    $resumeCoroutine
                        (STRING coroutineName;
                         OPTIONAL BITS ctrlBits);
```

Table 1.306-1. $resumeCoroutine (Generic)

$resumeCoroutine is used to switch context from the current coroutine to another coroutine, and resume execution of the other coroutine where it last left off (usually at a call to $resumeCoroutine). If the coroutine has not yet been resumed (i.e., it has been created, but not yet given control), it invokes the procedure (the "initializing procedure") specified when the

coroutine was created with $createCoroutine, using the data section given at the same time. The context of the current coroutine is stored in the $coroutine record, to be used when it is next resumed. Control returns from $resumeCoroutine when the resuming coroutine is itself next resumed. If the coroutine being resumed is the same as the resuming coroutine, then control returns immediately, with no overall effect.

The exception $coroutineExcpt is raised if a coroutine attempts to return from its initializing procedure. The initializing procedure must terminate by means of a call to $resumeCoroutine.

If p is Zero in the pointer form of $resumeCoroutine, $thisCoroutine.$next is resumed (it is an error if $thisCoroutine.$next is Zero).

The string form of $resumeCoroutine is equivalent to "$resumeCoroutine($findCoroutine(coroutineName),ctrlBits)".

The only valid ctrlBits bits are delete, errorOK, and $nonRecursive.

delete indicates that the current coroutine (and its subtree) is to be killed before resuming the target coroutine; see Section 1.208. It is an error to kill the target coroutine in this way; i.e., this option cannot be specified if the target coroutine is in the subtree rooted at the current coroutine.

The errorOK bit in ctrlBits is used to suppress error messages. In any case, if an error occurs, $resumeCoroutine returns false, without resuming the coroutine. A value of true means that the target coroutine was successfully resumed, and that the original coroutine has now itself been resumed from the coroutine given by "$thisCoroutine.$next".

$nonRecursive is permitted only if the delete bit is set. The children of the current coroutine replace it in the coroutine tree, as if the current coroutine were killed by $killCoroutine with $nonRecursive set; see Section 1.208.

1.307. retain

```
COMPILETIME
BITS
<macro>      retain;
```

Table 1.307-1. retain

retain is a bit that specifies that the break character is to be retained in the scanned string. It may be passed to scan.

1.308. $returnExcpt

```
# system variable
STRING $returnExcpt;
```

Table 1.308-1. $returnExcpt

$returnExcpt is a predefined exception that is raised when the end of a typed procedure is reached without executing a Return Statement.

1.309. $returnIfNoHandler

```
COMPILETIME
BITS
<macro>      $returnIfNoHandler;
```

Table 1.309-1. $returnIfNoHandler

$returnIfNoHandler is a bit that specifies that exception processing should terminate as if $raiseReturn had been called if no handler handles the exception. It may be passed to $raise and tested in $exceptionBits.

1.310. reverse

reverse reverses the order of elements in an array. It is one of the procedures provided by the sorting package, SRTMOD, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.311. $reverseDateAndMonth

```
COMPILETIME
BITS
<macro>      $reverseDateAndMonth;
```

Table 1.311-1. $reverseDateAndMonth

$reverseDateAndMonth is a bit that specifies that the month is to precede the day in the output date string. It may be passed to $dateAndTimeToStr and $dateToStr.

1.312. scan

```
STRING
PROCEDURE    scan          (MODIFIES STRING source;
                            STRING scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (MODIFIES STRING source;
                            BITS scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (POINTER(textFile) source;
                            STRING scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);
```

Table 1.312-1. scan (Generic) (continued)

```
STRING
PROCEDURE    scan          (POINTER(textFile) source;
                            BITS scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (MODIFIES STRING source;
                            INTEGER scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (POINTER(textFile) source;
                            INTEGER scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (POINTER(dataFile) source;
                            STRING scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (POINTER(dataFile) source;
                            BITS scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);

STRING
PROCEDURE    scan          (POINTER(dataFile) source;
                            INTEGER scanCtrl;
                            OPTIONAL BITS ctrlBits;
                            PRODUCES OPTIONAL INTEGER brkChr;
                            OPTIONAL POINTER($area) area);
```

Table 1.312-1. scan (Generic) (end)

A file or string (the "source") is scanned as directed by scanCtrl and ctrlBits. area specifies the destination area for the resulting string.

Characters (the "scanned characters") are removed from the source while or until any one of a set of characters (the "scan characters") is encountered in the source. The scanned characters are returned as the result string. Thus, scan allows the programmer to scan up to (or over) any one of a set of characters.

scanCtrl indicates what scan characters are to guide the scan. There are three ways to specify the scan characters:

1. scanCtrl can be a string that directly gives the scan characters; every character in the string is a scan character.

2. scanCtrl can be a bits obtained from scanSet. The scan characters are those in the string that was used as the argument to scanSet. scanCtrl may also have the value of several such bits IOR'd together (see Example 1.314-2), in which case the scan characters are taken from the string that is the concatenation of the corresponding arguments to scanSet. At most fifteen such sets may be in use at the same time, so the use of $scanSet is preferred if the values need not be IOR'd together.

3. scanCtrl can be an integer obtained from $scanSet. The scan characters are those in the string that was used as the argument to $scanSet.

The bits and integer forms of scanCtrl are more efficient for scan characters that are used repeatedly, since the string form involves some overhead that depends on the length of scanCtrl.

brkChr is the "break character", the character that causes the scan to terminate ("break"). The break character is often useful for determining the action following a scan.

In the file forms, if the file is opened for PDF I/O, the characters read from the file may be translated from the PDF to the host character set.

The valid predefined bits constants that may be set in ctrlBits are shown in Table 1.312-2.

End-of-source breaks the scan, with the result string equal to the string scanned up to the end-of-source, and brkChr equal to -1. -1 is not a valid character code.

How to use the scan characters:

break	The scan characters are to break the scan. All other characters are proceed characters. This is the default.
proceed	The scan characters cause the scan to proceed. All other characters are break characters.

What to do with the break character (it is always put into brkChr):

retain	Stays in source, not put into result string. This is the default.
append	Becomes last character of result string (removed from source).
discard	Discarded (removed from source, not put into result string).

Characteristics of result string:

omit	Discard characters instead of putting them into the result string (which is "", even if append is set). This option may result in a significant savings in execution time and string space.
upperCase	Source characters are converted to upper case before they are checked for breaking the scan and put into the result. The break character is the original character taken from the string (i.e., not converted to upper case). It is more efficient to use the upperCase option with compare than with scan if its only purpose is to provide a caseless comparison.

Table 1.312-2. Predefined Bits Constants for scan ctrlBits

```
    WHILE s DOB
        scan(s,letters,omit!upperCase);
        ttyWrite(scan(s,lettersAndDigits,
            proceed!upperCase),eol) END;

scans s for all identifiers, and writes each to tty
(assuming letters and lettersAndDigits are bits assigned
values as in Example 1.314-2).

    s := scan(f,".")

s is assigned all characters from the current position of
f up to the first period.  The period is retained as the
next character of the file, and is not put into s.
```

Example 1.312-3. Use of scan

1.313. scanRel

```
PROCEDURE   scanRel     (REPEATABLE BITS scanBits);

PROCEDURE   scanRel     (REPEATABLE INTEGER scanInteger);
```

Table 1.313-1. scanRel (Generic)

scanRel releases the 1-bits in scanBits so they may be reused by scanSet or releases the scanInteger so that it may be reused by $scanSet. The integer form gives an error if scanInteger was not assigned by $scanSet; the bits form has an undefined result if scanBits contains bits not assigned by scanSet.

1.314. scanSet

```
BITS
PROCEDURE    scanSet      (STRING scanChars);
```

Table 1.314-1. scanSet

scanSet is used to associate characters with a bit that can be used with the procedure "scan".

The bits value returned by scanSet has a single 1-bit. This bit is associated with the characters of scanChars when used by scan.

Fifteen bits are available as scan bits.

```
BITS letters,digits,lettersAndDigits;
...
letters := scanSet("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
digits  := scanSet("0123456789");

lettersAndDigits := letters!digits

scan can use "letters" for scanning letters, "digits" for
scanning digits, and "lettersAndDigits" for scanning
letters and digits.
```

Example 1.314-2. Use of scanSet

$scanSet provides integers that may be used with scan. Only one integer may be used at a time with scan, but over 32000 of them are available.

1.315. $scanSet

```
INTEGER
PROCEDURE    $scanSet     (STRING scanChars);
```

Table 1.315-1. $scanSet

scanSet is used to associate characters with an integer that can be used with the procedure "scan".

This integer value returned by $scanSet is associated with the characters of scanChars when used by scan.

Over 32000 integers are available as scan integers.

scanSet provides bits that may be used with scan. More than one bits may be used at a time with scan, but only fifteen of them are available.

1.316. scratchDispose

```
PROCEDURE    scratchDispose
                        (MODIFIES REPEATABLE ADDRESS a);

PROCEDURE    scratchDispose
                        (MODIFIES REPEATABLE CHARADR c);
```

Table 1.316-1. scratchDispose (Generic)

scratchDispose is used to dispose of scratch space obtained with newScratch (the address form) or $newScratchChars (the charadr form). The argument a or c is set to Zero. The result is undefined if the storage at a or c was not obtained by a call to newScratch or $newScratchChars, respectively.

MAINSAIL automatically keeps track of the size of scratch space allocations.

1.317. $searchCallChain

```
POINTER
PROCEDURE    $searchCallChain
                          (STRING moduleName;
                           OPTIONAL BITS ctrlBits);
```

Table 1.317-1. $searchCallChain

$searchCallChain searches the procedure call chain of the current coroutine (i.e., the current procedure's caller, its caller, and so on back to the initializing procedure of the coroutine) for a procedure invocation associated with the module named moduleName. If such a procedure invocation is found, $searchCallChain returns a pointer to the data section from which the most recent such call was made; otherwise, $searchCallChain returns nullPointer. The only valid ctrlBits bit is errorOK, which suppresses an error message if no call from moduleName is found.

For example, consider two unbound modules A and B. If A knows it was created by B, and that B is therefore in the call chain of A's initial procedure, and B contains some interface fields to which A needs access, A may find B's data section with:

```
$searchCallChain("B")
```

Without $searchCallChain, it would be necessary for B to make a pointer to itself available in a known place before allocating A, or to provide a special initialization procedure in A to which B could pass a pointer to itself.

1.318. $setCommandLine

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

$setCommandLine allows a program to set the command arguments for another module to s. This procedure is described in greater detail under the entry for $getCommandLine.

```
PROCEDURE    $setCommandLine
                          (OPTIONAL STRING s);
```

Table 1.318-1. $setCommandLine

1.319. $setConfigurationBit

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
<macro>      $setConfigurationBit
                          (BITS b);
```

Table 1.319-1. $setConfigurationBit

$setConfigurationBit sets various bits, called configuration bits, that control MAINSAIL execution. The possible arguments and their meanings are shown in Table 1.319-2. cmdFile and logFile echoing is described in Section 18.14 of part I of the "MAINSAIL Language Manual".

Only the bits shown in Table 1.319-2 should be changed at runtime. Changing other configuration bits from a program has undefined effects. Users must reference bits by identifier only, not by value, since their values may change in future releases. Some bits may become obsolete. Some system bits (see the description of $setSystemBit) may become configuration bits or vice versa. The type(s) of the configuration and/or system bits are subject to change; e.g., they may become long bits.

Configuration Bit	Meaning
$intFileFirst	Change the default search order
$objFileFirst	from libraries first to files
$exeFileFirst	first for the specified search.
$noAutoCmdFileSwitching	See the description of CONF in the "MAINSAIL Utilities User's Guide".
$echoCmdFile	Control cmdFile and logFile
$echoIfRedirected	echoing.
$lineOrientedDebug	Use the line-oriented interface for MAINDEBUG even when invoked from MAINEDIT (value '10).

Table 1.319-2. Configuration Bit Identifiers

1.320. $setExitCode

```
<macro>     $setExitCode
                        (LONG BITS bb);
```

Table 1.320-1. $setExitCode

At any time during MAINSAIL execution, "$setExitCode(bb)" can be used to set the MAINSAIL exit code to bb, where bb is a long bits value. When MAINSAIL exits, the value of the exit code is passed back to the operating system if it is possible to do so.

Two portable exit codes, $successExitCode and $failureExitCode, are defined for each operating system and can be used to portably return success or failure, respectively. These exit codes are defined according to the convention used by each system to determine whether or not a process terminates normally. Other values passed to $setExitCode have system-dependent meanings; the correspondence between MAINSAIL and OS values is described in the system-dependent MAINSAIL user's guides for those operating systems that support exit codes.

By default, MAINSAIL exits with the code $successExitCode.

1.321. $setFileCacheParms

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
BOOLEAN
PROCEDURE   $setFileCacheParms
                        (POINTER(file) f;
                         OPTIONAL LONG BITS attributes;
                         OPTIONAL BITS ctrlBits;
                         OPTIONAL BOOLEAN setDefaultParms;
                         OPTIONAL BOOLEAN setParms;
                         OPTIONAL LONG INTEGER
                             requestedMinSize;
                         OPTIONAL LONG INTEGER
                             requestedMaxSize;
                         OPTIONAL INTEGER
                             requestedHitPercent);
```

Table 1.321-1. $setFileCacheParms

$setFileCacheParms sets the parameters for the cache associated with the file f. If the input parameters are invalid or if f cannot be cached, an error occurs and $setFileCacheParms returns false.

If f is nullPointer, the global cache parameters are set and $setFileCacheParms returns true.

IF f is not nullPointer, $setFileCacheParms examines the attributes parameter and f's current attributes to determine which cache parameters to set. Figure 1.321-2 shows the possible combinations of these variables and the actions taken by $setFileCacheParms.

attributes specifies how the file is to be cached, i.e., $globallyCached or $privatelyCached. If not specified, the way the file is cached remains unchanged.

If setDefaultParms is true, the file cache parameters are to be set to their default values. If setParms is true, the file cache parameters are set to the values specified by requestedMinSize, requestedMaxSize, and requestedHitPercent. If neither setDefaultParms nor setParms is true, the file cache parameters remain unchanged.

requestedMinSize and requestedMaxSize are the requested minimum and maximum number of buffers in the LRU list, respectively. requestedMinSize and requestedMaxSize must be non-negative integers and requestedMinSize must be less than or equal to requestedMaxSize. requestedHitPercent must be an integer between 0 and 100.

$setFileCacheParms returns true if it successfully set the file cache parameters.

The only valid ctrlBits is errorOK. An error message is generated if an error occurs and errorOK is not specified.

1.322. setFileName

```
PROCEDURE    setFileName (STRING modName,fileName);
```

Table 1.322-1. setFileName

setFileName is used to form or remove an association between a module name and the name of the file that contains the executable form of the module.

It is an error if modName is not a valid module name, i.e., a one- to six-character identifier. modName is the actual name of the module, not a dummy name as established by setModName. If fileName is the null string, any association for modName is removed; otherwise, fileName specifies the name of a file containing the executable form of the module, and must be a valid host system file name. The file must contain only a single module; it cannot be a module library.

setFileName provides the ability to override the default name of a file on which the module is assumed to reside. In addition, it can utilize the full syntax of host file names, rather than just the syntax of module names, and thus can specify characteristics such as the directory on which the file resides.

1.323. setModName

```
PROCEDURE    setModName   (STRING dummyName,actualName);
```

Table 1.323-1. setModName

f's current attributes	attributes parameter	Action(s) Taken
global	global or Zero	The global cache parameters are set and the global cache LRU list is adjusted so that it does not exceed requestedMaxSize.
global	private	All of f's cached buffers are written if dirty and are removed from the global cache. All buffers except f's current buffer are disposed. f is privately cached with the specified cache parms.
private	private or Zero	f's private cache parameters are set and the private cache LRU list is adjusted so that it does not exceed requestedMaxSize.
private	global	All of f's cached buffers are written if dirty and the private cache is disposed (including all of f's cached buffers except the current buffer). f is globally cached. The global cache parameters are set and the global cache LRU list is adjusted so that it does not exceed requestedMaxSize.

Figure 1.321-2. Actions Taken by $setFileCacheParms (continued)

not cached	If the attributes parameter is nonZero, then f is cached according to the specified value. If the attributes parameter is Zero, then f is globally cached if its buffer size is the same as the size of buffers in the global cache; otherwise, f is privately cached. The appropriate cache parameters are set. If f is globally cached, then the global cache LRU list is adjusted so that it does not exceed requestedMaxSize.

Figure 1.321-2. Actions Taken by $setFileCacheParms (end)

```
MODULE m (PROCEDURE p);
...
setFileName("m",
    IF someCondition THEN "file1" ELSE "file2");
p; # Call procedure p in module m

Assuming the call to p causes m to be bound, m is
obtained from "file1" if someCondition is true, otherwise
from "file2".
```

Example 1.322-2. Use of setFileName

setModName is used to form or remove an association between a dummy name (dummyName) for a module and the actual name (actualName) of a module. It is an error if dummyName and actualName are not valid module names, i.e., one- to six-character identifiers, unless actualName is the null string.

Whenever a data section is to be allocated for a module m (for which no data section yet exists), the current list of associations is searched to determine if m is a dummy name. If so, the corresponding actual name is used as the name of the module for which the control section is found. It is not checked whether the actual name is itself a dummy name; i.e., the actual

name must in fact be the name of a module. Thus, a module can use a dummy module name to make references to a module the actual name of which is not known, provided that a setModName call has associated the dummy name with the actual name.

It is not specified whether module name associations (as established by setModName) have any effect on any data section for a given module if another data section for the same module already exists.

If setModName is issued for a dummy name that is already associated with an actual name, then the new association replaces the old one.

The association set up by "setModName(dummyName,actualName)" can be released by "relModName(dummyName)" or "setModName(dummyName,"")".

1.324. setPos

```
BOOLEAN
PROCEDURE   setPos      (POINTER(file) f;
                         OPTIONAL LONG INTEGER n;
                         OPTIONAL BITS ctrlBits);
```

Table 1.324-1. setPos

setPos sets the position within a file opened for random access to n.

The units used for positioning are characters for text files and files open for PDF I/O, storage units for data files not open for PDF I/O. The first position in a file is position 0, the next position 1, and so forth.

If the specified position n is greater than the end-of-file position or negative, setPos returns false. In this case it also gives an error message unless the errorOK bit is set in ctrlBits. errorOK is the only valid ctrlBits bit.

Example 1.324-2 shows how an array that is too large for memory may be maintained as a random file.

Specifying a constant numeric value other than 0L for a data file position is machine-dependent. For portability, all file positions in data files should be specified as multiples of sizes of the MAINSAIL data types using $ioSize. For example, to position to the second integer in a data file, use "setPos(f,cvli($ioSize(f,integerCode)))".

```
<module a>
...
ttyWrite("Target machine: ");
setModName("codGen",ttyRead);
...
<module b is invoked>
...
relModName("codGen"); # release the association
...

<module b>
...
MODULE codGen (... PROCEDURE addGen (...); ...);
...
addGen(...); # Assume codGen has not been bound at
             # this point; the call to addGen binds it
...
```

The actual module referenced by the call to addGen, in module b, is determined by what the user types in response to the inquiry in module a. This allows module b to access an anonymous module, of which only the interface characteristics are known. Anonymous modules of which the interfaces are known may also be accessed by explicit pointers, but explicit pointers are syntactically clumsier; however, explicit pointers may be required if it is necessary to execute simultaneously two different programs needing separate copies of the same module.

Example 1.323-2. Use of setModName

1.325. $setSearchPath

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

The procedure $setSearchPath allows a user program to get the effect of the MAINEX "SEARCHPATH" subcommand (see the "MAINSAIL Utilities User's Guide"). For example, the effect of the subcommand

```
# simulate an array on disk

DEFINE
   liIntegerSize    =        cvli(size(integerCode)),
   hugeArySize      =        20001L * liIntegerSize;

# simulate INTEGER ARRAY(0 TO hugeArySize - 1) hugeAry
POINTER(dataFile) hugeAry;
open(hugeAry,"hugeAry",create!random!alterOk,hugeArySize);

# simulate hugeAry[li] := n
setPos(hugeAry,li * liIntegerSize); write(hugeAry,n);

# simulate n := hugeAry[li]
setPos(hugeAry,li * liIntegerSize); read(hugeAry,n);
```

Example 1.324-2. Use of setPos

```
PROCEDURE   $setSearchPath
                          (STRING pattern,path);
```

Table 1.325-1. $setSearchPath

```
SEARCHPATH source:* *.msl /9.0/*.msl
```

may be obtained from a MAINSAIL program with

```
$setSearchPath("source:*","*.msl /9.0/*.msl");
```

The searchpath syntax may change in future releases.

1.326. $setSystemBit

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
<macro>     $setSystemBit
                        (BITS b);
```

Table 1.326-1. $setSystemBit

$setSystemBit sets various bits, called system bits that control MAINSAIL execution. The possible arguments and their meanings are shown in Table 1.326-2.

Only the bits shown in Table 1.326-2 should be changed. Changing other system bits from a program has undefined effects. Users must reference bits by identifier only, not by value, since their values may change in future releases. Some bits may become obsolete. Some configuration bits (see the description of $setConfigurationBit) may become system bits or vice versa. The type(s) of the configuration and/or system bits are subject to change; e.g., they may become long bits.

1.327. $setTheDate

```
BOOLEAN
PROCEDURE   $setTheDate (LONG INTEGER date);
```

Table 1.327-1. $setTheDate

If the operating system does not provide the date, $setTheDate sets MAINSAIL's internal date to the value specified by date.

$setTheDate returns false if the date is available from the operating system, true otherwise.

If the date is not provided by the operating system and $setTheDate has been called, $date and $dateAndTime return the value specified instead of prompting for the current date.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

System Bit	Meaning
$swapBit	Write swapping information to "TTY" as if the MAINEX "SWAPINFO" subcommand had been given.
$memInfoBit	Write memory management information to "TTY" as if the MAINEX "MEMINFO" subcommand had been given.
$mapAtMemInfoBit	Write memory maps as if the MAINEX "MAP" subcommand (with no arguments) had been given.
$noCheckConsistency	Do not perform interface consistency checking when modules are bound, as if the MAINEX "NOCHECKCONSISTENCY" subcommand had been given.
$fileInfoBit	Write file information to "TTY" as if the "FILEINFO" subcommand had been given.
$controlInfoBit	Write coroutine and exception information to "TTY" as if the "CONTROLINFO" subcommand had been given.
noResponse	Responses are not requested from errMsg as if the "NORESPONSE" MAINEX or compiler subcommand had been given.

Table 1.326-2. System Bit Identifiers

1.328. sin

```
REAL
PROCEDURE    sin          (REAL r);

LONG REAL
PROCEDURE    sin          (LONG REAL r);
```

Table 1.328-1. sin (Generic)

sin returns the sine of its argument, which is in radians.

1.329. sinh

```
REAL
PROCEDURE    sinh         (REAL r);

LONG REAL
PROCEDURE    sinh         (LONG REAL r);
```

Table 1.329-1. sinh (Generic)

sinh returns the hyperbolic sine of its argument, which is in radians.

1.330. size

```
COMPILETIME
INTEGER
PROCEDURE    size         (INTEGER typeCode);
```

Table 1.330-1. size (Generic) (continued)

```
$BUILTIN SPECIAL
INTEGER
PROCEDURE   size        (CLASS c);

LONG INTEGER
PROCEDURE   size        (POINTER p);
```

Table 1.330-1. size (Generic) (end)

size computes the number of storage units in a data type or class.

The integer form returns the number of storage units required by the data type with type code typeCode (see Section 2.7 of part I of the "MAINSAIL Language Manual").

The class form returns the number of storage units required by the data fields in a record of the class c. Procedure fields play no part in the size. The class form is always evaluated at compiletime.

In the pointer form, p points to a record, array, or data section. The size of the record, array, or data section, in storage units, is returned. In the case of the data section, the size of the entire data section, not just the data interface fields, is returned. If p is Zero, 0L is returned; if p is not valid, an error message is issued.

These procedures produce machine-dependent results, since the sizes of the data types and the interpretation of storage units vary across implementations.

$ioSize should be used instead of size to compute the sizes of data types written to data files.

A garbage collection cannot occur during a call to size.

```
    i := size(integerCode);

i is set to the the number of storage units
required by an integer.
```

Example 1.330-2. Use of size

1.331. sort

sort is one of the procedures provided by the sorting package, SRTMOD, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.332. sqrt

```
REAL
PROCEDURE    sqrt          (REAL x);

LONG REAL
PROCEDURE    sqrt          (LONG REAL x);
```

Table 1.332-1. sqrt (Generic)

sqrt returns the square root of x. It is an error if x is less than zero.

1.333. $sRand

$sRand returns the next pseudo-random number produced by one of the algorithms in $ranMod, which is documented in detail in the "MAINSAIL Utilities User's Guide".

1.334. $stackOverflowExcpt

```
# system variable
STRING $stackOverflowExcpt;
```

Table 1.334-1. $stackOverflowExcpt

$stackOverflowExcpt is a predefined exception that is raised on systems where stack overflow can be usefully caught when a procedure stack overflow occurs (see Section 9.13 of part I of the "MAINSAIL Language Manual"). Usually a little extra stack is held in reserve for the handling the exception, but not much, so no deep procedure recursion should occur while

handling $stackOverflowExcpt. A stack overflow in a coroutine in which $stackOverflowExcpt has already been raised has undefined effects, since the stack reserve has already been used up.

Stack overflows on machines that do not raise $stackOverflowExcpt also have undefined effects and frequently cause a program to hang.

1.335. $storageUnitRead

```
LONG INTEGER
PROCEDURE   $storageUnitRead
                          (POINTER(dataFile) f;
                           LONG INTEGER numStorageUnits;
                           POINTER ptrBase;
                           OPTIONAL LONG INTEGER dspl;
                           OPTIONAL ADDRESS adrBase);
```

Table 1.335-1. $storageUnitRead

$storageUnitRead reads numStorageUnits storage units of data from the data file f to an address computed as "displace(IF adrBase THEN adrBase EL cva(ptrBase),dspl)". An error occurs if both adrBase and ptrBase are Zero. The number of storage units read is returned.

If adrBase is non-Zero, the effect is undefined if "displace(adrBase,dspl)" does not lie within an area of scratch space obtained with newScratch or newPage or if the area does not contain at least numStorageUnits of space beyond the computed address. If adrBase is Zero, the effect is undefined if ptrBase does not point to a valid MAINSAIL data structure or if the data structure is smaller than numStorageUnits storage units.

For large amounts of data, $storageUnitRead is more efficient than a series of calls to the procedure "read".

Garbage collections may occur during a call to $storageUnitRead or $storageUnitWrite. Therefore, converting an array to an address before reading data into it or writing data from it by means of these procedures has undefined effects. The proper way to do a $storageUnitRead of n elements of type t from a file f into an array ary (starting at the first element) is to convert it to pointer:

```
$storageUnitRead(f,n * cvli(size(tCode)),cvp(ary),
    lDisplacement(cva(cvp(ary)),$adrOfFirstElement(ary)))
```

where tCode is the type code for type t, i.e., one of integerCode, longIntegerCode, bitsCode, etc.

$pageRead and $characterRead are other procedures used to read large amounts of data from a file with a single procedure call.

1.336. $storageUnitWrite

```
PROCEDURE    $storageUnitWrite
                          (POINTER(dataFile) f;
                           LONG INTEGER numStorageUnits;
                           POINTER ptrBase;
                           OPTIONAL LONG INTEGER dspl;
                           OPTIONAL ADDRESS adrBase);
```

Table 1.336-1. $storageUnitWrite

$storageUnitWrite writes numStorageUnits storage units of data to the data file f from an address computed as "displace(IF adrBase THEN adrBase EL cva(ptrBase),dspl)". An error occurs if both adrBase and ptrBase are Zero.

If adrBase is non-Zero, the effect is undefined if "displace(adrBase,dspl)" does not lie within an area of scratch space obtained with newScratch or newPage or if the area does not contain at least numStorageUnits of space beyond the computed address. If adrBase is Zero, the effect is undefined if ptrBase does not point to a valid MAINSAIL data structure or if the data structure is smaller than numStorageUnits storage units.

For large amounts of data, $storageUnitWrite is more efficient than a series of calls to the procedure "write".

$pageWrite and $characterWrite are other procedures used to write large amounts of data to a file with a single procedure call.

1.337. store

```
$BUILTIN
PROCEDURE    store        (CHARADR c;
                           INTEGER v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           INTEGER v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           ADDRESS v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           POINTER v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           BITS v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           STRING v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           REAL v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           LONG INTEGER v;
                           OPTIONAL INTEGER dspl);
```

Table 1.337-1. store (Generic) (continued)

```
$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           LONG REAL v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           LONG BITS v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           BOOLEAN v;
                           OPTIONAL INTEGER dspl);

$BUILTIN
PROCEDURE    store        (ADDRESS a;
                           CHARADR v;
                           OPTIONAL INTEGER dspl);
```

Table 1.337-1. store (Generic) (end)

store is used to store a value into a memory or character address.

The forms in which an address is the first parameter store the value v into the memory location given by "displace(a,dspl)". If "displace(a,dspl)" is undefined, the effect of store is undefined. The form in which a charadr is the first argument stores the character code v at the character address given by "displace(c,dspl)". If "displace(c,dspl)" is undefined, the effect of store is undefined.

The effect is undefined if a or c is Zero.

1.338. $strToDate

```
LONG INTEGER
PROCEDURE    $strToDate   (STRING s;
                           PRODUCES OPTIONAL STRING remS;
                           OPTIONAL BITS ctrlBits;
                           PRODUCES OPTIONAL BOOLEAN
                               success);
```

Table 1.338-1. $strToDate

$strToDate produces a MAINSAIL date given a string. The date specified may be in any of the formats produced by $dateToStr.

A two-digit year is generally assumed to be a year in the current century; a year in the first century may be specified by terminating the date string with "A.D."; e.g., "17 A.D." represents the year 17, not 1917.

s need only start with a valid date string. remS is that part of s that remains after the part containing the date string has been removed.

If s does not begin with a valid date, 0L is returned, success is set to false, and the value of remS is unspecified. 0L is a valid return value if the string represents a date difference, so success must be examined if an invalid date difference string may be input.

The valid ctrlBits bits are $localTime, $gmt, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the string is interpreted as a local date and returned in local date format. If $gmt is specified, a GMT format date is returned.

Unless errorOK is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.339. $strToDateAndTime

```
BOOLEAN
PROCEDURE    $strToDateAndTime
                          (STRING s;
                           PRODUCES LONG INTEGER date,time;
                           PRODUCES OPTIONAL STRING remS;
                           OPTIONAL BITS ctrlBits);
```

Table 1.339-1. $strToDateAndTime

$strToDateAndTime produces a MAINSAIL date and time given a string. The date and time may be specified in any of the formats output by $dateAndTimeToStr. If the string includes a time zone name, it must be "GMT", the name of the local standard or daylight savings time zone, or a time zone defined to MAINSAIL with the "DEFINETIMEZONE" MAINEX subcommand (see the description of MAINEX in the "MAINSAIL Utilities User's Guide"). If the string does not have the format of a time difference and no time zone is included in the string, local time is assumed by default (either daylight savings or standard time, depending on the date part of the string). $strToDateAndTime returns true if successful.

s need only start with a valid date and time string. remS is that part of s that remains after the part containing the time string has been removed.

If s does not begin with a valid date and time, false is returned, and the value of remS is unspecified.

Valid ctrlBits bits are errorOK, $localTime, $localTimeToGMT, $GMTtoLocalTime, and $gmt. errorOK suppresses error messages for invalid input values. The other four values are ignored if the string has the format of a time difference; otherwise, at most one of the four bits may be specified, and the bits have the effects shown in Table 1.339-2.

$localTime is the default if none of the four bits is specified. If a time zone name is included in s, it overrides the assumption about the time zone specified by the ctrlBits bit.

Bit	Input String Interpreted as	Output Format for date and time
$localTime	Local time	Local format
$localTimeToGMT	Local time	GMT format
$GMTtoLocalTime	GMT	Local format
$gmt	GMT	GMT format

Table 1.339-2. $strToDateAndTime ctrlBits Bits

1.340. $strToTime

```
LONG INTEGER
PROCEDURE   $strToTime  (STRING s;
                         PRODUCES OPTIONAL STRING remS;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL BOOLEAN
                             success);
```

Table 1.340-1. $strToTime

$strToTime produces a MAINSAIL time given a string. The time specified may be in any of the formats produced by $timeToStr. In addition, if s represents a time difference, the hours specified may exceed 24 and the minutes and seconds 60; e.g., "+45:75:75" means a time difference of 46 hours, 16 minutes, and 15 seconds.

s need only start with a valid time string. remS is that part of s that remains after the part containing the time string has been removed.

If s does not begin with a valid time, 0L is returned, success is set to false, and the value of remS is unspecified. 0L is a valid return value if the string represents a time difference, so success must be examined if an invalid time difference string may be input.

The valid ctrlBits bits are $localTime, $gmt, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the string is interpreted as a local time and returned in local time format. If $gmt is specified, a GMT format time is returned.

Unless errorOK is specified, an error message is generated for erroneous input values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.341. Structure Blaster Procedures

The Structure Blaster provides a number of procedures that operate on an entire data structure at once (i.e., all chunks and strings accessible from a given pointer). The Structure Blaster is a separate product, and is documented in detail in the "MAINSAIL Structure Blaster User's Guide".

$structureCompare compares two structures. $structureCopy copies an entire structure. $structureDataToText translates a structure into a human-readable text form for examination, porting to another system, or editing. $structureDispose disposes an entire structure. $structureInfo returns information about a structure written to a file. $structureRead reads a structure from a file into memory. $structureSetUp does preliminary processing for $structureWrite. $structureTextToData translates a text form into a data structure. $structureUnSetUp undoes $structureSetUp, if the structure is not to be written after all. $structureWrite writes an entire structure to a file.

1.342. $subscriptExcpt

```
# system variable
STRING $subscriptExcpt;
```

Table 1.342-1. $subscriptExcpt

$subscriptExcpt is a predefined exception that is raised when a subscript error (out-of-range array subscript) occurs in code with runtime checking enabled (see Section 15.2 of part I of the "MAINSAIL Language Manual").

1.343. $systemExcpt

```
# system variable
STRING $systemExcpt;
```

Table 1.343-1. $systemExcpt

$systemExcpt is a predefined exception that is raised by errMsg. See Section 16.8 of part I of the "MAINSAIL Language Manual" and Section 1.143. $systemExcpt can be caught to handle various error conditions, but the programmer should be aware that MAINSAIL system error messages are subject to change.

1.344. $systemNameAbbreviation

```
STRING
<macro>     $systemNameAbbreviation;
```

Table 1.344-1. $systemNameAbbreviation

$systemNameAbbreviation is the abbreviation for the name of the target operating system. Abbreviations are shown in Table B-2 of part I of the "MAINSAIL Language Manual".

1.345. $systemNameFull

```
STRING
<macro>     $systemNameFull;
```

Table 1.345-1. $systemNameFull

$systemNameFull is the full name of the target operating system. Operating system names are shown in Table B-2 of part I of the "MAINSAIL Language Manual".

1.346. $systemNumber

```
COMPILETIME
INTEGER
<macro>      $systemNumber;
```

Table 1.346-1. $systemNumber

$systemNumber is the number the target operating system. Operating system numbers are shown in Table B-2 of part I of the "MAINSAIL Language Manual".

1.347. tab

```
COMPILETIME
STRING
<macro>      tab;
```

Table 1.347-1. tab

tab is the string consisting of the tab character.

1.348. tan

```
REAL
PROCEDURE    tan          (REAL r);

LONG REAL
PROCEDURE    tan          (LONG REAL r);
```

Table 1.348-1. tan (Generic)

tan returns the tangent of its argument, which is in radians.

1.349. tanh

```
REAL
PROCEDURE    tanh          (REAL r);

LONG REAL
PROCEDURE    tanh          (LONG REAL r);
```

Table 1.349-1. tanh (Generic)

tanh returns the hyperbolic tangent of its argument, which is in radians.

1.350. $thisCoroutine

```
# system variable
POINTER($coroutine) $thisCoroutine;
```

Table 1.350-1. $thisCoroutine

$thisCoroutine points to the current coroutine. Explicitly altering $thisCoroutine has undefined effects.

1.351. thisDataSection

```
$BUILTIN
POINTER
PROCEDURE    thisDataSection;
```

Table 1.351-1. thisDataSection

thisDataSection returns a pointer to the data section of the module from which it is called.

1.352. $time

```
LONG INTEGER
PROCEDURE   $time       (OPTIONAL BITS ctrlBits);
```

Table 1.352-1. $time

$time returns the current time of day.

The valid predefined bits constants for ctrlBits are $localTime, $gmt, and errorOK.

If $localTime is specified (or if neither $localTime nor $gmt is specified), the local time is returned, if available. If $gmt is specified, the GMT time is returned, if available.

If the predefined bits constant errorOK is specified and the time is not provided by the operating system, 0L is returned. If errorOK is not specified, the user is prompted for the time if it is not provided by the operating system.

$dateAndTime should be used if both date and time are to be obtained for the same instant. Otherwise, a wraparound can occur at midnight.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

1.353. $timeDifference

```
COMPILETIME
BITS
<macro>     $timeDifference;
```

Table 1.353-1. $timeDifference

$timeDifference is bit that specifies that a time difference (rather than an absolute time) is to be output. It may be passed to $assembleTime. It may be returned by $dateFormat and $timeFormat to indicate the format of the long integer date or time argument.

1.354. $timeFormat

```
BITS
PROCEDURE    $timeFormat (LONG INTEGER time);
```

Table 1.354-1. $timeFormat

$timeFormat returns $gmt if its argument is a GMT time, $localTime if its argument is a local time, $timeDifference if its argument is a time difference, or '0 if its argument is not a valid time value.

1.355. $timeSubcommandsSet

```
BOOLEAN
PROCEDURE    $timeSubcommandsSet;
```

Table 1.355-1. $timeSubcommandsSet

$timeSubcommandsSet returns true if and only if any of the MAINEX time zone subcommands has been issued. These subcommands are necessary for correct processing of GMT dates and times. It is presumed that all time zone subcommands have been issued with correct values if any of them have been issued. The operating system may provide part of the information normally provided by the subcommands, but the subcommands override the operating system if the subcommands are set. See Section 19.3 of part I of the "MAINSAIL Language Manual" for additional information.

1.356. $thisFileName

The compiletime string pseudo-procedure $thisFileName returns the name of the current source file, i.e., of the file containing the call to $thisFileName. This can be especially useful to specify the location of a forward procedure:

```
FORWARD($thisFileName) PROCEDURE ...
```

since this use of the "FORWARD" directive works even if the source file containing it is renamed.

1.357. $timeout

```
PROCEDURE    $timeout     (LONG INTEGER seconds);
```

Table 1.357-1. $timeout

$timeout pauses for approximately the number of seconds specified. On some operating systems, it is necessary to implement $timeout with a loop that waits until the time has elapsed; on such systems, $timeout may consume a good deal of CPU time.

1.358. $timeToStr

```
STRING
PROCEDURE    $timeToStr   (LONG INTEGER time;
                           OPTIONAL BITS ctrlBits;
                           OPTIONAL POINTER($area) area);
```

Table 1.358-1. $timeToStr

$dateToStr produces a string from a MAINSAIL time of day, which may be an absolute (local or GMT) time or a time difference. area specifies the destination area for the resulting string.

The default format for $timeToStr if time is an absolute time is "<hour>:<minute>:<second>", where <hour> is measured on a 24-hour clock, e.g., "22:16:09", "0:44:00". The <minute> and <second> fields always occupy exactly two digits, even if they are zero; the <hour> field may occupy one or two digits, depending on its value.

Time differences are converted by default to the format:

```
{-}<h> hour{s} <m> minute{s} <s> second{s}
```

The "-" is included if time is negative.

The null string is returned if an invalid input value is detected.

The predefined bits constants shown in Table 1.358-2 are valid. Example 1.358-3 shows some sample output values.

The standard MAINSAIL date and time formats are described in Section 19.1 of part I of the "MAINSAIL Language Manual".

Bit	Meaning
$twelveHour	A twelve-hour clock is used. The time is followed by " A.M." or " P.M.", as appropriate. Midnight is given as "12" rather than "0".
$excludeSeconds	The time is truncated to the minute.
$briefFormat	If time is a time difference, convert it to "[+|-]<h>:<m>:<s>", e.g., "+02:17:03", "-00:01:55". A zero difference has a plus sign ("+00:00:00").
errorOK	No error message is given if an invalid input value is detected.

Table 1.358-2. Predefined Bits Constants for $timeToStr ctrlBits

For sample times of 0:00:17 and 15:46:54, the following string representations are possible:

$twelveHour	$excludeSeconds	Resulting strings	
clear	clear	"0:00:17"	"15:46:54"
clear	set	"0:00"	"15:46"
set	clear	"12:00:17 A.M."	"3:46:54 P.M."
set	set	"12:00 A.M."	"3:46 P.M."

Example 1.358-3. Sample $timeToStr Output Formats

1.359. truncate

```
$BUILTIN
INTEGER
PROCEDURE    truncate     (REAL v);

$BUILTIN
LONG INTEGER
PROCEDURE    truncate     (LONG REAL v);
```

Table 1.359-1. truncate (Generic)

truncate returns the (long) integer obtained by discarding v's fraction; i.e., it rounds towards zero.

See Table 1.169-3 for a table contrasting ceiling, cvi, floor, and truncate.

```
truncate(10.5)   = 10
truncate(-10.5)  = -10
```

Example 1.359-2. Use of truncate

1.360. $truncateFile

```
BOOLEAN
PROCEDURE    $truncateFile
                          (POINTER(file) f;
                           LONG INTEGER fileSize;
                           OPTIONAL BITS ctrlBits);
```

Table 1.360-1. $truncateFile

$truncateFile truncates the file f, if possible. It returns true if the file is truncated, false otherwise. An error occurs if f is not open for random output. fileSize is the requested size in characters if f is a text file and in storage units if f is a data file.

If the operating system does not support file truncation, then the file size is not changed, $truncateFile returns false, and no error message is issued. If the operating system supports file truncation, then f is truncated so that its end-of-file position is as close to the requested size as possible and $truncateFile returns true. The end-of-file position of f may be larger than the requested size, but never smaller.

An error occurs if f is a sequential file, if f is opened for input only, or if the operating system's file truncation procedure returns an error. In the first two cases, an error message is written to logFile regardless of the setting of the errorOK bit in ctrlBits.

The only valid ctrlBits is errorOK, which suppresses any operating system error message that might otherwise occur.

$truncateFile does not change the current MAINSAIL file position. If the current MAINSAIL file position is beyond the end-of-file position of the truncated file, then the file position is set as if a setPos beyond the end-of-file had been done. The end-of-file position is not changed until data are written beyond the end-of-file.

1.361. $tstConfigurationBit

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
BOOLEAN
<macro>     $tstConfigurationBit
                        (BITS b);
```

Table 1.361-1. $tstConfigurationBit

$tstConfigurationBit examines various bits that control MAINSAIL execution. The bits are documented in detail under $setConfigurationBit.

1.362. $tstSystemBit

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
BOOLEAN
<macro>     $tstSystemBit
                          (BITS b);
```

Table 1.362-1. $tstSystemBit

$tstSystemBit examines various bits that control MAINSAIL execution. The bits are documented in detail under $setSystemBit.

1.363. ttycWrite

```
PROCEDURE   ttycWrite   (REPEATABLE INTEGER char);
```

Table 1.363-1. ttycWrite

ttycWrite writes the character with code char to the file "TTY".

For example, "ttycWrite('c')" writes the letter "c" to the primary output file. The effect is the same as that of "ttyWrite("c")".

1.364. $ttyEofExcpt

```
# system variable
STRING $ttyEofExcpt;
```

Table 1.364-1. $ttyEofExcpt

$ttyEofExcpt is a predefined exception that is raised when the end of "TTY" is reached in a call to ttyRead, as described in Section 18.10 of part I of the "MAINSAIL Language Manual".

1.365. ttyRead

```
STRING
PROCEDURE   ttyRead     (OPTIONAL POINTER($area) area);
```

Table 1.365-1. ttyRead

ttyRead reads a new line from the file "TTY". area specifies the destination area for the resulting string.

The character or characters that terminate the input line (e.g., eol) are discarded.

```
STRING      s;
INTEGER     t,u;

s := ttyRead; # read next line from "TTY"
read(s,t,u);  # read integers from s into t and u
              # (see Section 1.288)
```

Example 1.365-2. Use of ttyRead

Example 1.365-2 gets two integers from the same terminal input line. The user must realize that two integers are to be typed on the same line.

1.366. ttyWrite

```
PROCEDURE   ttyWrite    (REPEATABLE STRING v);

PROCEDURE   ttyWrite    (REPEATABLE BOOLEAN v);

PROCEDURE   ttyWrite    (REPEATABLE INTEGER v);

PROCEDURE   ttyWrite    (REPEATABLE BITS v);

PROCEDURE   ttyWrite    (REPEATABLE REAL v);

PROCEDURE   ttyWrite    (REPEATABLE LONG INTEGER v);

PROCEDURE   ttyWrite    (REPEATABLE LONG BITS v);

PROCEDURE   ttyWrite    (REPEATABLE LONG REAL v);
```

Table 1.366-1. ttyWrite

ttyWrite converts its argument to a string representation if it is not a string, and then writes the string to the file "TTY". The conversions performed are the same as performed by the system procedure "write".

```
    ttyWrite(eol & "i and j are ",i," and ",j,"." & eol)

If i = 10 and j = 11, the following is written to tty on a
new line:

    i and j are 10 and 11.
```

Example 1.366-2. Use of ttyWrite

1.367. $twelveHour

```
COMPILETIME
BITS
<macro>      $twelveHour;
```

Table 1.367-1. $twelveHour

$twelveHour is a bit that specifies that a twelve-hour clock (instead of the usual twenty-four-hour clock) is to be used in forming the output string. It may be passed to $dateAndTimeToStr and $timeToStr.

1.368. $twoYearDigits

```
COMPILETIME
LONG BITS
<macro>      $twoYearDigits;
```

Table 1.368-1. $twoYearDigits

$twoYearDigits is a bit that specifies that only the last two digits of a year are to be included in the output of the procedure to which it is passed. It may be passed to $dateToStr and $dateAndTimeToStr.

1.369. $typeName

```
STRING
PROCEDURE    $typeName    (INTEGER typeCode;
                           OPTIONAL BITS ctrlBits);
```

Table 1.369-1. $typeName

The procedure $typeName returns a string type name corresponding to a type code typeCode. The correspondence between type codes and names is shown in Table 1.369-2. The only valid ctrlBits bit is errorOK. If not specified, an error message is issued if typeCode is not a valid MAINSAIL type code. The null string is returned for invalid MAINSAIL type codes.

Type Code	Name
booleanCode	BOOLEAN
integerCode	INTEGER
longIntegerCode	LONG INTEGER
realCode	REAL
longRealCode	LONG REAL
bitsCode	BITS
longBitsCode	LONG BITS
stringCode	STRING
addressCode	ADDRESS
charadrCode	CHARADR
pointerCode	POINTER

Table 1.369-2. MAINSAIL Data Type Codes and Corresponding Names

1.370. unBind

```
PROCEDURE   unBind      (REPEATABLE MODULE m);

PROCEDURE   unBind      (REPEATABLE STRING modName);
```

Table 1.370-1. unBind (Generic)

unBind undoes the effect of a call to "bind"; i.e., it executes the final procedure (if any) of the module m or the module named modName, and then disposes the module's bound data section (see Section 1.127). Unlike the system procedure "dispose", unBind does not release the module's control section. unBind has no effect if the module does not have a bound data section.

The linkage of any modules that have established linkage to the module is broken.

The string form generates an error if modName is not a valid module name, i.e., a one- to six-character identifier.

1.371. $unboundModuleExcpt

```
# system variable
STRING $unboundModuleExcpt;
```

Table 1.371-1. $unboundModuleExcpt

$unboundModuleExcpt is a predefined exception that is raised when an interface data field of an unallocated bound module is accessed in code with runtime checking enabled (see Section 15.2 of part I of the "MAINSAIL Language Manual").

1.372. $unbuffered

```
COMPILETIME
BITS
<macro>     $unbuffered;
```

Table 1.372-1. $unbuffered

$unbuffered is a bit that specifies that unbuffered I/O is to be allowed on the file that is being opened. It may be passed to $createUniqueFile, open, and $reOpen.

1.373. upperCase

```
COMPILETIME
BITS
<macro>     upperCase;
```

Table 1.373-1. upperCase

upperCase is a bit that specifies that case is ignored in the processed text. It may be passed to cmdMatch, compare, equ, and scan.

1.374. useKeyWord

```
COMPILETIME
BITS
<macro>      useKeyWord;
```

Table 1.374-1. useKeyWord

useKeyWord is a bit that specifies processing based on individual words instead of a whole string. It may be passed to cmdMatch and $registerException.

1.375. $useOriginalFileName

```
COMPILETIME
BITS
<macro>      $useOriginalFileName;
```

Table 1.375-1. $useOriginalFileName

$useOriginalFileName is a bit that suppresses transformation of a file name through logical file names and searchpaths. It may be passed to $createUniqueFile, $delete, $directory, $fileInfo, open, $rename, and $reOpen.

1.376. $useProgramInterface

```
BOOLEAN
<macro>      $useProgramInterface;
```

Table 1.376-1. $useProgramInterface

$useProgramInterface is true if and only if the initial procedure of the current module m is being invoked for one of the following reasons:

- An interface procedure is being called.
- "bind(m,b)" or "new(m,b)" was called, where b has the $programInterface bit set.

Because of the way it is implemented, $useProgramInterface must be used only in the initial procedure of the module, before it makes any procedure calls; otherwise, the use of $useProgramInterface is undefined.

Normal uses of bind and new within the MAINSAIL runtime system do not set the $programInterface bit. For example, when a module is invoked from MAINEX, the bit is not set, so that $useProgramInterface is false if queried by the module's initial procedure.

1.377. $userID

```
STRING
PROCEDURE    $userID      (OPTIONAL BITS ctrlBits);
```

Table 1.377-1. $userID

The user ID of the current user is returned, if possible. The form of the user ID is described in each operating-system-dependent MAINSAIL user's guide.

If the operating system does not provide a user ID, a prompt is written to logFile and a user name read from cmdFile. After the user is prompted once, the user ID is remembered and returned on all subsequent calls to $userId.

The only valid ctrlBits bit is errorOK; if set, the null string is returned without dialogue if the user ID is unavailable.

1.378. warning

```
COMPILETIME
BITS
<macro>     warning;
```

Table 1.378-1. warning

warning is a bit that specifies that an error message is just warning. It may be passed to errMsg and tested in $exceptionBits. It is set in a call to $raise made from errMsg if the warning bit is set in the call to errMsg.

1.379. write

```
PROCEDURE   write       (MODIFIES STRING s;
                         REPEATABLE BOOLEAN v);

PROCEDURE   write       (MODIFIES STRING s;
                         POINTER($area) area;
                         REPEATABLE BOOLEAN v);

PROCEDURE   write       (POINTER(textFile) f;
                         REPEATABLE BOOLEAN bo);

PROCEDURE   write       (POINTER(dataFile) f;
                         REPEATABLE BOOLEAN bo);

PROCEDURE   write       (POINTER(dataFile) dst;
                         REPEATABLE BITS v);

PROCEDURE   write       (POINTER(textFile) dst;
                         REPEATABLE BITS v);

PROCEDURE   write       (MODIFIES STRING dst;
                         REPEATABLE BITS v);
```

Table 1.379-1. write (Generic) (continued)

```
$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE BITS v);

PROCEDURE    write        (POINTER(dataFile) dst;
                           REPEATABLE INTEGER v);

PROCEDURE    write        (POINTER(textFile) dst;
                           REPEATABLE INTEGER v);

PROCEDURE    write        (MODIFIES STRING dst;
                           REPEATABLE INTEGER v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE INTEGER v);

PROCEDURE    write        (POINTER(dataFile) dst;
                           REPEATABLE LONG BITS v);

PROCEDURE    write        (POINTER(textFile) dst;
                           REPEATABLE LONG BITS v);

PROCEDURE    write        (MODIFIES STRING dst;
                           REPEATABLE LONG BITS v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE LONG BITS v);

PROCEDURE    write        (POINTER(dataFile) dst;
                           REPEATABLE LONG INTEGER v);

PROCEDURE    write        (POINTER(textFile) dst;
                           REPEATABLE LONG INTEGER v);

PROCEDURE    write        (MODIFIES STRING dst;
                           REPEATABLE LONG INTEGER v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE LONG INTEGER v);
```

Table 1.379-1. write (Generic) (continued)

```
PROCEDURE    write        (POINTER(dataFile) dst;
                           REPEATABLE LONG REAL v);

PROCEDURE    write        (POINTER(textFile) dst;
                           REPEATABLE LONG REAL v);

PROCEDURE    write        (MODIFIES STRING dst;
                           REPEATABLE LONG REAL v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE LONG REAL v);

PROCEDURE    write        (POINTER(dataFile) dst;
                           REPEATABLE REAL v);

PROCEDURE    write        (POINTER(textFile) dst;
                           REPEATABLE REAL v);

PROCEDURE    write        (MODIFIES STRING dst;
                           REPEATABLE REAL v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE REAL v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE STRING v);

PROCEDURE    write        (POINTER(textFile) dst;
                           REPEATABLE STRING v);

PROCEDURE    write        (MODIFIES STRING dst;
                           REPEATABLE STRING v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE POINTER v);
```

Table 1.379-1. write (Generic) (continued)

```
$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE ADDRESS v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE CHARADR v);

$BUILTIN
PROCEDURE    write        (MODIFIES ADDRESS dst;
                           REPEATABLE BOOLEAN v);

PROCEDURE    write        (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE INTEGER v);

PROCEDURE    write        (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE LONG INTEGER v);

PROCEDURE    write        (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE REAL v);

PROCEDURE    write        (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE LONG REAL v);

PROCEDURE    write        (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE BITS v);

PROCEDURE    write        (MODIFIES STRING s;
                           POINTER($area) area;
                           REPEATABLE LONG BITS v);

PROCEDURE    write        (MODIFIES STRING r;
                           POINTER($area) area;
                           REPEATABLE STRING s);

PROCEDURE    write        (POINTER(dataFile) dst;
                           REPEATABLE STRING v);
```

Table 1.379-1. write (Generic) (end)

"write" writes the value v to dst, which may be an output file, string, or memory address.

The forms that write a boolean, (long) integer, (long) real, or (long) bits to a text file or a string automatically convert to a string representation, which is the same as the default format produced by cvs. A string is written to a text file, data file, or string as the sequence of its characters.

If dst is a string, the string representation of v is concatenated onto the end of dst.

After writing to an address, the address is displaced to the location immediately following that to which the value was written. The effect is undefined if the address is nullAddress or if "displace(dst,size(<data type of v>))" is undefined.

The form that writes a string to an address writes the machine-dependent string descriptor of v to dst, i.e., stores the string descriptor at dst and displaces a by the size of a string descriptor.

In the forms that write a string to a file, if the file is opened for PDF I/O, characters may be translated to the PDF character set.

In the area forms, area specifies the destination area for the resulting string.

1.380. $writeCalls

```
TEMPORARY FEATURE:  SUBJECT TO CHANGE
```

```
PROCEDURE   $writeCalls (OPTIONAL POINTER(textFile) f;
                         OPTIONAL POINTER($coroutine) p;
                         OPTIONAL BITS ctrlBits);
```

Table 1.380-1. $writeCalls

The code that handles the "CALLS" response to errMsg is available as the procedure $writeCalls.

f is the file to which the list of callers is written. If f is nullPointer, logFile is assumed. p is the coroutine the call chain of which is to be listed. If p is nullPointer, the current coroutine is assumed.

```
    INTEGER i,j; REAL r; BITS b;
    ADDRESS a; STRING s;

    write(outFile,i,r,b)

writes an integer, real, and bits to outFile.
If outFile is a text file, a conversion to a
string representation takes place.

    write(s,i," ",j)

has the same effect as

    s := s & cvs(i) & " " & cvs(j)

    write(a,i,r,b)

writes an integer, real, and bits to memory locations
starting at address a.  a is updated to have the value
given by "displace(a,size(integerCode) + size(realCode) +
size(bitsCode))".
```

Example 1.379-2. Use of write

The only valid ctrlBits bit is $ignoreMe. If this bit is set, $writeCalls's immediate caller is not included in the call list (used only if the current coroutine's call chain is being listed).

A sample output from $writeCalls is shown in Example 1.380-2.

```
MODULE DECIMAL OFFSET PROCEDURE (most recently called 1st)
WC               50     INITIALPROC
KERMOD        19290     $NEWDATASEC
KERMOD        20066     $LBBIND
MAINEX         3592     $INVOKEMODULE
MAINEX         2128     EXECUTEMODULE
MAINEX         2540     $MAINSAILEXEC
KERMOD        38434     RUNMAINSAIL
KERMOD        28504     INITIALPROC
```

Example 1.380-2. Sample $writeCalls Output

Index

XIDAK, Inc., 530 Oak Grove Avenue, M/S 101, Menlo Park, CA 94025, (415) 324-8745