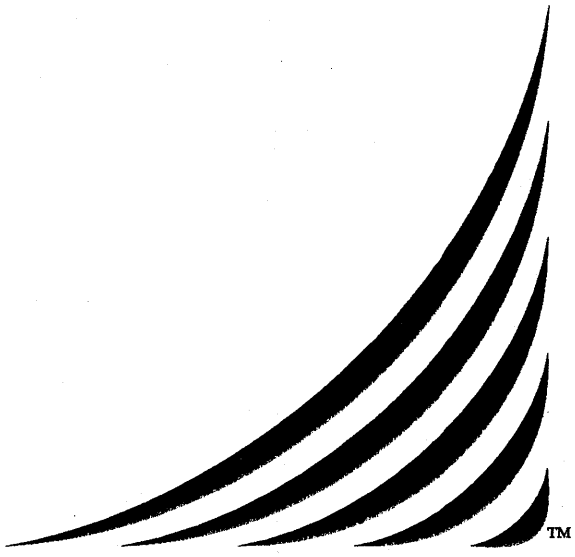


MAINSAIL® Combined Release Notes

Version 12.10 through 12.15 Releases

11 July 1991

XIDAK

Table of Contents

List of Figures

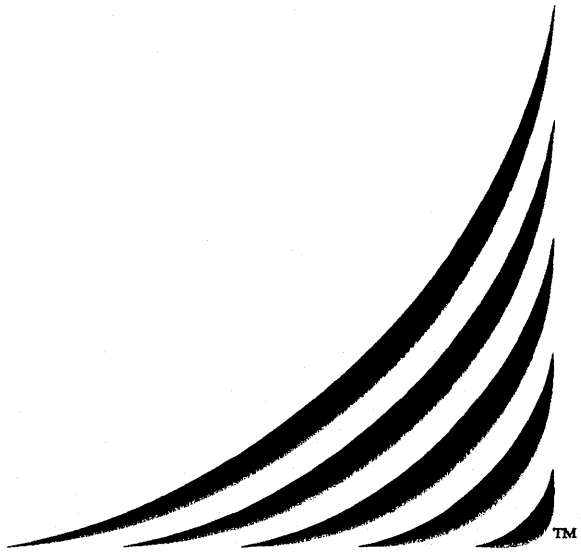
List of Tables

1. Introduction

This combined release note documents the changes made between Versions 12.10 and 12.15 of MAINSAIL. The Version 12.10 manual set (the most recently published complete set of documentation) should be used in conjunction with the following release notes, all included in this combined release note, in order to have a complete description of the features of Version 12.15:

- "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27",
- "MAINSAIL Release Notes, Version 11.28 Release",
- "MAINSAIL Release Notes, Version 11.30 Release",
- "MAINSAIL Release Notes, Version 12.14 Release",
- and "MAINSAIL Release Notes, Version 12.15 Release".

Because it has been over two years since the last documentation set has been issued, and a large number of features have been documented since then only in release notes, it has been decided to issue a combined release note with an index to make it easier to find features added since the Version 12.10 release. The release notes in this combined note are included in reverse order, from most recent to least recent.



MAINSAIL® Release Notes

Version 12.15 Release

11 July 1991

2. Introduction

This release note documents the changes made between Versions 12.14 and 12.15 of MAINSAIL. The Version 12.10 manual set (the most recently published complete set of documentation) should be used in conjunction with the following release notes in order to have a complete description of the features of Version 12.15:

- "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27",
- "MAINSAIL Release Notes, Version 11.28 Release",
- "MAINSAIL Release Notes, Version 11.30 Release",
- "MAINSAIL Release Notes, Version 12.14 Release",
- and the current release note.

Version 12.15 object modules cannot be run under previous Version 11 or Version 12 releases, and object modules from previous Versions cannot be run under Version 12.15; you must rebuild all intmods and recompile all object modules. In addition, the layout of data structures in memory has changed between Version 11 and Version 12; all Version 11 (but not older Version 12) Structure Blaster data images must be translated or remade.

3. Recent and Proposed Changes

3.1. $commandLineArgs No Longer Supported

The $commandLineArgs array, introduced as a temporary feature in the last release, is no longer supported. Instead, use the new procedure $nextCommandLineArg (see Section 6.1.2).

3.2. 24-Bit Mode on IBM's AIX on IBM System/370 XA

On IBM's AIX on IBM System/370 XA, 24-bit mode is no longer supported. The only form of the "cc" command that works to build a MAINSAIL bootstrap is now:

```
cc -Hxa -o mainsa mainsa.s <msDir>/m.o<eol>
```

where <msDir> is the MAINSAIL directory.

3.3. disconnserver() Should Be Called instead of close() in C RPC Facility

When a C program using the RPC (remote procedure call) facility closes its connection, the new function disconnserver() should be called instead of close(). Its argument should be the value returned from connserver().

4. Clarification of Documentation

4.1. CONF "FOREIGNMODULES" Command

The default foreign modules specified in the system configuration file supplied by XIDAK should always be included in any "FOREIGNMODULES" command issued by the builder of a MAINSAIL bootstrap. The default foreign modules, where present, provide services that are needed by the runtime system (on some platforms, there may not be any default foreign modules). The interfaces to the default modules themselves are not documented and are subject to change.

To include the default foreign modules in your own list of foreign modules, make the first line of your "FOREIGNMODULES" argument a string consisting of just an equals sign ("="). For example, to declare the foreign modules ABC and XYZ:

```
FOREIGNMODULES<eol>
=<eol>
ABC<eol>
XYZ<eol>
```

The effect of building a MAINSAIL bootstrap without the default foreign modules is undefined, although on many systems you will get an error when you attempt to link the bootstrap.

4.2. The Definition of DIV and MOD with Negative Arguments

Hardware designers are divided (no pun intended) about how integer DIV and MOD should work with negative numbers. Some favor the convention that integer division truncates toward zero, others that it truncates toward negative infinity. Neither convention is obviously "correct". The following are examples of DIV operations under each convention:

Truncate toward zero	Truncate toward negative
7 DIV 3 = 2	7 DIV 3 = 2
-7 DIV 3 = -2	-7 DIV 3 = -3
7 DIV -3 = -2	7 DIV -3 = -3
-7 DIV -3 = 2	-7 DIV -3 = 2

The truncate toward zero convention has the advantage of obeying these relations that also apply to floating point numbers:

```
a DIV b = - (-a DIV b) = - (a DIV -b) = -a DIV -b
```

The truncate toward negative infinity convention has the advantage that division by positive powers of two can be implemented as an arithmetic right shift for both positive and negative dividends, since, e.g.:

```
-1 DIV 4 = -1
```

the same as the result of a right shift of two bits.

MOD is usually defined in terms of DIV by the relation:

```
(a DIV b) * b + a MOD b = a
```

i.e.:

```
a MOD b = a - (a DIV b) * b
```

The following examples show MOD operations obeying this relation under each convention, corresponding to the DIV operations above:

Truncate toward zero	Truncate toward negative
7 MOD 3 = 1	7 MOD 3 = 1
-7 MOD 3 = -1	-7 MOD 3 = 2
7 MOD -3 = 1	7 MOD -3 = -2
-7 MOD -3 = -1	-7 MOD -3 = -1

For reasons of efficiency, XIDAK implements DIV and MOD on each machine with whatever instruction the hardware provides. This leads to different results for these operations on different machines when negative integers are involved. XIDAK does not plan to change this behavior, because it would require extra tests that would result in significantly slower DIV and MOD on some machines (the code generated for DIV and MOD would have to implement the logic of the routines below, which is non-trivial).

The following inline procedures can be used to implement the truncate toward zero convention:

```
INLINE INTEGER PROCEDURE iDivZ (INTEGER dividend,divisor);
BEGIN
IF NOT divisor THEN errMsg("iDivZ: division by zero","",fatal);
RETURN(
    IF dividend > 0 THEN
        IF divisor > 0 THEN dividend DIV divisor
        EL - (dividend DIV - divisor)
    EL  IF divisor > 0 THEN - (- dividend DIV divisor)
        EL - dividend DIV - divisor);
END;

INLINE INTEGER PROCEDURE iModZ (INTEGER dividend,divisor);
BEGIN
IF NOT divisor THEN errMsg("iModZ: division by zero","",fatal);
RETURN(
    IF dividend > 0 THEN dividend MOD abs(divisor)
    EL - (- dividend MOD abs(divisor)));
END;
```

The following inline procedures can be used to implement the truncate toward negative infinity convention:

```
INLINE INTEGER PROCEDURE iDivN (INTEGER dividend,divisor);
BEGIN
IF NOT divisor THEN errMsg("iDivN: division by zero","",fatal);
RETURN(
    IF dividend > 0 THEN
        IF divisor > 0 THEN dividend DIV divisor
        EL - (dividend DIV - divisor) -
            (IF dividend MOD - divisor THEN 1 EL 0)
    EL  IF divisor > 0 THEN - (- dividend DIV divisor) -
            (IF - dividend MOD divisor THEN 1 EL 0)
        EL - dividend DIV - divisor);
END;

INLINE INTEGER PROCEDURE iModN (INTEGER dividend,divisor);
BEGIN
INTEGER i;
IF NOT divisor THEN errMsg("iModN: division by zero","",fatal);
IF dividend > 0 THEN
    IF divisor > 0 THEN RETURN(dividend MOD divisor)
    EB  i := dividend MOD - divisor;
        IF i THEN i .+ divisor END
EL  IF divisor > 0 THENB
        i := - dividend MOD divisor;
        IF i THEN i := divisor - i END
    EL  RETURN(- (- dividend MOD - divisor));
RETURN(i);
END;
```

Analogous procedures can be written for long integers.

[UCR 90-470]

4.3. Arithmetic Overflow

MAINSAIL does not support a portable notion of arithmetic exceptions, especially overflow, and particularly floating-point overflow. The notion of overflow visible to a MAINSAIL program is whatever notion is supported by the underlying hardware, with all its quirks. Since different platforms have different notions of overflow, overflow is not a portable concept. Programs should not be written expecting the rules for overflow to be the same across different platforms.

In particular, on some platforms, intermediate calculations are done using more precision than can be represented by variables of the given type. Thus, overflow (as defined by the underlying hardware) might occur only when a result is finally stored in memory, since the representable range of exponents is smaller for memory operands than for intermediate operands. Usually results are stored in memory fairly near where they were calculated. But the store could be far away from the calculation, maybe even in a different procedure.

For example, a value returned by a Return Statement might be too large to be representable in memory, but not too large for the intermediate representation. It is possible for such a value to be calculated by the processor with no overflow, and returned to the caller where it is then stored in a variable, at which point overflow would occur. A programmer expecting overflow to occur when the value was originally calculated would be disappointed. According to the processor's notion of overflow, the calculation didn't overflow at all; overflow occurred only when the caller eventually stored the result in memory.

The above observations apply to integer overflow as well as to floating point overflow. For example, integers on the IBM System/370 occupy 16 bits in memory, but all integer intermediate calculations are done using 32 bits. This is not an arbitrary decision that was made when XIDAK implemented MAINSAIL on the System/370; it is how the System/370 16-bit instructions behave. An integer procedure could return the result of multiplying, say, 200 times 200, which is outside the range of integers on the System/370. But because the result is returned in a 32-bit register, no overflow would occur within the procedure. Overflow would only occur when the caller finally stored the result in memory as a 16-bit integer, and even then it would only be detected if the ACHECK compiler option were set.

Programs that calculate values that are outside the allowed range of values for a given type just are not legal programs, and the results of executing such programs are not defined. XIDAK does what it can, within reason, to do something sensible, but there are no guarantees. Overflow ultimately occurs only when the processor says it occurs. [UCR 90-125]

4.4. Classes Declared in Different Intmods Are Different Classes

Consider the following three short modules:

```
BEGIN "foo1"
$DIRECTIVE "SAVEON","NOGENCODE";
CLASS c;
MODULE e (PROCEDURE d (POINTER(c) p));
PROCEDURE d (POINTER(c) p);;
END "foo1"

BEGIN "foo2"
$DIRECTIVE "SAVEON","NOGENCODE";
CLASS c;
MODULE e (PROCEDURE d (POINTER(c) p));
END "foo2"

BEGIN "foo3"
RESTOREFROM "foo1";
RESTOREFROM "foo2";
POINTER(c) q;   # this "c" is foo2$c
INITIAL PROCEDURE;
d(q);
END "foo3"
```

FOO3 includes the definitions of the class c when it restores from FOO1 and FOO2. However, the class c declared in those two modules is actually two classes: foo1$c and foo2$c.

In FOO3, d(q) is compiled as foo1$d(q) rather than e.d(q), since outers take precedence over fields. foo1$d's parameter p is classified with foo1$c, whereas q is classified with foo2$c. Since foo1$c and foo2$c come from different declarations, they are not the same class. Hence, the compiler complains of an argument mismatch when compiling FOO3.

There are two simple ways to eliminate the error from this code:

- Change the declaration of q to be "POINTER(foo1$c) q".
- Reverse the RESTOREFROM's (this would classify q with foo1$c).

Neither of these simple changes is ideal, since the fundamental problem is the duplicate declarations of c and e in foo1 and foo2. A better change is to move the duplicate declarations into a third common module which is restored from by foo1 and foo2.

[UCR 91-19]

4.5. $waitForDescendants

The description of $waitForDescendants in the "MAINSAIL STREAMS User's Guide" says:

```
If the array [children] is omitted or nullArray is given,
it waits until all of its children have died.
```

The wording should be amended to say that it waits for all the children in existence at the time of the call to $waitForDescendants to die. New children can come into existence during the call for $waitForDescendants.

In order to wait for ALL children of the calling coroutine to die, even those created during a call to $waitForDescendants, do:

```
WHILE $thisCoroutine.$down DO $waitForDescendants;
```

[UCR 90-430]

4.6. PDF I/O

PDF is a format that specifies the representation of individual data types in a file. The individual data in the file are not "tagged", so it is not possible to read the file unless the sequence of data types in the file is known. For example, if you write one INTEGER, followed by one REAL, followed by one LONG BITS with PDF I/O, you must read the same data using PDF I/O with an INTEGER read, followed by a REAL read, followed by a LONG BITS read. Otherwise, the data will not be correctly interpreted.

4.6.1. PDF I/O and $storageUnit/$page I/O

$storageUnit I/O (using the procedures $storageUnitRead and $storageUnitWrite) and $page I/O (using the procedure $pageRead and $pageWrite) simply copy the bytes at the specified address to a file or vice versa, with no modification. These procedures do not understand or interpret the data written as MAINSAIL data types: the data read or written are just a sequence of bytes.

Each procedure performing PDF I/O must know the data type of each value read or written. If a procedure does not have a way to specify the data type of the datum read or written, it does not know how to do the translation to or from PDF format, and so cannot support PDF I/O.

$pageRead, $pageWrite, $storageUnitRead, and $storageUnitWrite do not have any way of specifying the data type(s) of the data involved. They may be given arbitrary addresses at which any sort of data could be stored. Therefore, they cannot support PDF translations.

By contrast, read and write are generic procedures, so a different instance procedure is called for each data type, and the appropriate PDF translation for that type can be performed. The other procedures that perform PDF I/O (including $characterRead/$characterWrite) all operate on text, and so perform translations to and from the PDF character set.

4.6.2. PDF I/O and $structureRead

In both the dataFile and textFile forms, $pdf is valid in ctrlBits to $structureRead. If this bit is set or if the file is open for PDF I/O, then $structureRead reads a PDF image from the file. If $pdf is not set in ctrlBits and if the file is not open for PDF I/O, $structureRead automatically determines the format of the image in the file and reads it accordingly. This automatic detection introduces some overhead during $structureRead. Also, the heuristic used to detect what type of image resides in the file is not foolproof and could give the wrong result. For these reasons, the $pdf bit should be specified if the image is known to be a PDF image.

[UCR's 91-51, 91-115]

4.7. MAINSAIL Swap Files ("swpxxx.tmp")

If MAINSAIL opens a swap file during execution, it is deleted when MAINSAIL returns normally back to the operating system. The swap file is opened with the delete bit set so that when it is closed, it is deleted.

If MAINSAIL does not exit normally, e.g., if you type CTRL-C or the process is otherwise killed, then the logic that closes all open files is not executed, and hence the swap file is not deleted. This also happens if execution starts in foreign code. [UCR 91-111]

4.8. PC Monitoring in MAINPM

PC monitoring may be disabled during directory manipulations, such as creating, moving, deleting, or renaming directories. This is due to the inability of the child process that is created on some operating systems to perform these tasks to handle the PC interrupt. If your program does much directory manipulation, your PC monitoring results may be inaccurate.

4.9. The Use of MAINPM's "TRACECHUNKS" Command

Several questions have arisen on the use of the MAINPM "TRACECHUNKS" command, introduced in Version 12.14. These are some clarfications to the documentation in the "MAINSAIL Release Notes, Version 12.14 Release":

- It is not necessary to issue the "PC" command in order to use the "TRACECHUNKS" command.
- It is not necessary to compile to the modules in which you wish to trace chunks with the "MONITOR" compiler option.
- If you give either the "NOMONITORLIB" or "NOMONITORMODULE" MAINPM command, or if information from $KERMOD shows up in your "TRACECHUNKS" listing, you should give the command "NOMONITORMODULE $KERMOD". Information about $KERMOD (the MAINSAIL kernel) is unlikely to be useful to the average MAINSAIL user.

A sample session with the "TRACECHUNKS" command is shown in Figure 4.9-1. Note that SAMPLE is compiled with the "DEBUG" compiler option, which includes instruction maps in the intmod so that the statement text can be written in the "TRACECHUNKS" listing.

[UCR 91-20]

4.10. Entering a Debugging Session

There are four basic ways to get into a debugging session:

- An M command to establish a module context, followed by B or T command to set a breakpoint in that module, followed by an E command to execute a program that (presumably) hits the breakpoint in the module.
- A K command to set a count break followed by an E command to run a program (which gives control to the debugger when the specified number of debuggable procedures have been entered).
- A "DEBUG" response to the "Error response:" prompt.
- A call to $debugExec from a program.

```
*compil<eol>
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, 1989, and 1990
 by XIDAK, Inc., Palo Alto, California, USA.

compile (? for help): /p/sample.msl,<eol>
> debug<eol>
> <eol>
Opening intmod for $SYS from
 intlib(/std/12/14/sun3/mainsail/sys-um2.ilb)>$sys-um20

/p/sample.msl 1 2
Objmod for SAMPLE stored on sample-um2.obj
Intmod for SAMPLE stored on sample-um2.int

compile (? for help): <eol>
*mainpm<eol>

MAINPM (tm) (type ? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, 1989, and 1990
 by XIDAK, Inc., Palo Alto, California, USA.
MAINPM: monitormodule sample<eol>
MAINPM: tracechunks<eol>
MAINPM: execute sample<eol>
Size of hash bucket: 131<eol>
Next key to be hashed (eol to stop): abc<eol>
36

Next key to be hashed (eol to stop): de<eol>
21

Next key to be hashed (eol to stop): f<eol>
45

Next key to be hashed (eol to stop): <eol>
Trace data in z29662.tmp
MAINPM: list tty<eol>
Total real time: 12 seconds
Total cpu time: .440 seconds
Total chunk space usage: 480
Total string space usage: 3774
Analyzing trace data...
Deleted z29662.tmp
Printing trace data...
```

Figure 4.9-1. Sample "TRACECHUNKS" Session (continued)

```
module{.proc}.offset                                          $clear/
{file filePos}                                                $dispose   still
{source text}              new's   dispose's  collects    Area    allocated
---------------------- --------- --------- --------- --------- ---------
Opening intmod for SAMPLE from sample-um2.int
Opening intmod for $SYS from
 intlib(/std/12/14/sun3/mainsail/sys-um2.ilb)>$sys-um20

SAMPLE.INITIALPROC.280          4         4         0         0         0
/p/sample.msl 2598
IF NOT s := ttyRead THEN DONE;

SAMPLE.INITIALPROC.228          1         1         0         0         0
/p/sample.msl 2512
bucketSize := cvi(ttyRead);
                          --------- --------- --------- --------- ---------
                                5         5         0         0         0
MAINPM: quit<eol>
```

Figure 4.9-1. Sample "TRACECHUNKS" Session (end)

4.11. The Easiest Way to Insert a Page Mark in MAINVI

The easiest way to insert a page mark in MAINVI is to issue the command ":swm p". The ":swm" command allows MAINED-specific commands to be issued (in this case the "P" command to insert a page mark). Thus, all commands available in MAINED are also available from MAINVI.

4.12. $characterWrite Parameter ctrlBits

The correct declaration of the system procedure $characterWrite is:

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

Previous documentation omitted the ctrlBits parameter.

5. Bugs Fixed

5.1. Language

5.1.1. $cpuTimeResolution

$cpuTimeResolution was set incorrectly on some UNIX systems. [UCR 90-452]

5.2. Runtime System

5.2.1. $moveCoroutine and Child Coroutines

When $moveCoroutine was called to move a coroutine and its children to someplace else in the coroutine tree, it lost the children; i.e., they were pruned from the coroutine tree and never reinserted.

The form of $moveCoroutine that moves only a single coroutine and promotes the coroutine's children to replace the coroutine being moved (i.e., the form with the $nonRecursive bit set in ctrlBits), worked fine. [UCR 91-43]

5.2.2. Short-Array Check and newUpperBound

MAINSAIL is supposed to check at runtime when a short array exceeds the allowable bounds when the array is first allocated and also when its bounds are adjusted by means of newUpperBound. MAINSAIL was doing the check properly when the array was allocated, but not for calls to newUpperBound. This bug has been fixed; newUpperBound now enforces the same short array rules as the procedure "new". [91-96]

5.2.3. $strToDateAndTime

$strToDateAndTime got a segmentation violation if no time zone was specified in the input string. [UCR 91-12]

5.2.4. $removeDateAndTime

$removeDateAndTime documentation says it should remove either a date, a time, or a date followed by a time. However, it did not correctly remove a date by itself. [UCR 90-53]

5.2.5. Conversion to (Long) Integer of Values with Fractional Part near .5 on IBM RISC System/6000 UNIX

The MAINSAIL routine that implements rounding on the IBM RISC System/6000 UNIX used to consider only the high-order 32 bits of a floating point value's fractional part when deciding whether to round up or down; any remaining bits were ignored. Thus, if the following were all true:

- the fractional part occupied more than 32 bits, where the high-order bit was set, the next 31 bits were zero, and the remaining bits were nonzero (so that the fractional part denoted a number that slightly exceeded .5), and
- the low order bit of the integer part was zero (so that the integer part denoted an even number),

then when the number was rounded to (long) integer, the routine would ignore the nonzero low order bits of the fractional part, which made the fractional part appear to be exactly .5. IEEE rules for rounding specify that if a number is exactly halfway between the two closest values it could be rounded to, the number is to be rounded to the value with low-order bit equal to zero. Thus, 2.5 would be rounded down to 2, whereas 3.5 would be rounded up to 4. [UCR 91-119]

5.3. Compiler

5.3.1. Optimizer Error "changeLoopBBNum: expected invariant CSE"

The optimizer issued the error message "changeLoopBBNum: expected invariant CSE" when it encountered a loop containing two or more loop-invariant interface variables of another module, e.g., mod.itf1 and mod.itf2. [UCR's 90-463, 90-466]

5.3.2. Optimizer and Interface Fields Accessed by load/store

The optimizer binds variables to registers over certain regions of a program in order to reduce the number of memory references and thereby speed up the program's execution. However, the optimizer was binding interface fields of the current module to registers even within regions that might have accessed those interface fields by other paths, or aliases.

For example, consider a loop that accessed interface fields both by their names and by means of the built-in procedures "load" and "store", using thisDataSection as the base address and the fields' displacements. Since the fields were bound to registers over the loop, the changes made by assignments to the fields using the fields' names were reflected only in the registers that the fields were bound to, and not in memory (at least until the end of the region, when the registers were stored if necessary). Meanwhile, the values loaded from memory by means of the procedure "load" were obsolete, since the fields' current values existed only in registers. Likewise, the values stored into memory by means of the procedure "store" weren't reflected in the registers to which the fields were bound.

The optimizer was fixed so that it no longer binds an interface field of the current module to a register if either:

- the field is modified within the region and any of its possible aliases are accessed within the region, or
- any of the field's possible aliases are modified within the region.

[UCR 91-162]

5.3.3. Optimizer and Handle Statements

The optimizer sometimes generated bad code for Handle Statements because it made invalid assumptions about what happened when an exception was raised. [UCR 91-83]

5.3.4. Bad Statement Map When Macro SOURCEFILE's Another File

A bad statement map was generated for a procedure that called a macro that first SOURCEFILE'd another file (to get statements) and then called another procedure. The bad map eventually elicited a "setPos failed" error when DISASM was run if the file that was SOURCEFILEd was larger than the file that was originally compiled. [UCR's 91-109, 91-150]

5.3.5. Bad Code for Floating Point Expressions on MIPS R2000

The MIPS R2000 code generator sometimes generated bad code for complicated floating point expressions because the code generator had made incorrect assumptions about the registers affected by certain floating point instructions. [UCR's 90-464, 90-469]

5.3.6. $collectLock Not Decremented after Return from FLI Call on IBM RISC System/6000 UNIX

$collectLock was not decremented after return from an FLI call on the IBM RISC System/6000 UNIX if the call had collectable parameters. [UCR 90-461]

5.3.7. Spurious Errors Compiling FLI Modules on PRISM and IBM RISC System/6000

On the PRISM and the IBM RISC System/6000, a series of spurious error messages was issued when an FLI interface procedure had at least eleven more parameters than the FLI module's first interface procedure:

```
passArgsToCallee: expected pred list to be empty
Error Response:

passArgsToCallee: expected succ list to be empty
Error Response:

regName: expected register
Error Response:
```

[UCR 91-79]

5.3.8. Code Generator Error "decKeepVal: register not kept"

Several different code generators issued the message "decKeepVal: register not kept" when compiling procedure calls with many modifies parameters for which were passed field variables of which the pointer part was a subscripted variable, e.g.:

```
PROCEDURE p (MODIFIES INTEGER i1,i2,...,in);
CLASS c (INTEGER i1,i2,...,in; ...);
POINTER(c) ARRAY(*) ary;
INTEGER i;
...
p(ary[i].i1,ary[i].i2,...,ary[i].in);
```

The code generators have been modified so that this error is less likely to occur, although it is still possible for the code generator to run out of allocatable registers in certain situations. [UCR's 90-304, 91-56, 91-73]

5.3.9. ACHECK Compiler Option on CLIPPER

The CLIPPER chip (or at least some versions of it) has a bug in the way it sets the overflow condition code for the "divw" and "modw" instructions, which are used to implement the MAINSAIL "DIV" and "MOD" operations, respectively. These instructions are advertised as setting overflow if the dividend is -2^31 and the divisor is -1, but experimentation has revealed that they set overflow whenever the dividend and divisor have different signs and the dividend's absolute value is less than the divisor's (e.g., -497 divided by 512).

Previously, if ACHECK was set, code was generated to catch overflow after "divw" or "modw" by checking the overflow condition code. Now, no code is generated after "modw", since overflow can't occur for a MOD operation. For "divw", code is generated beforehand to check explicitly for the dividend equal to -2^31 and the divisor equal to -1, and the overflow condition code is ignored.

[UCR 90-59]

5.3.10. ACHECK Option on Intel 80386

The Intel 80386 code generator now emits code to check for overflow after a (long) integer multiplication if ACHECK is set. Previously the checking code was omitted.

5.4. Utilities

5.4.1. MODLIB and INTLIB and Uncleared Pages

MODLIB and INTLIB did not clear partial pages at the ends of modules inserted into libraries. Consequently, OBJCOM sometimes reported spurious differences between object modules that were identical. MODLIB and INTLIB now clear trailing partial pages. [UCR 90-488]

5.4.2. DVIEW

DVIEW under MAINSAIL Version 12 had several bugs. These bugs were not present in Version 11.

The bugs were:

- The "w" (write) command should have changed the (long) integer that was most recently displayed, as it did under Version 11. Instead, it changed the (long) integer immediately following the one that was most recently displayed.

- If DVIEW was displaying data in reverse order and the unit of data being displayed was switched from integer to long integer or vice versa, DVIEW should have displayed the new unit immediately preceding the most recently displayed value. Instead, it began displaying data at a position that was off by the difference in size between the old unit and the new one. For example, if you were displaying integers in reverse order and the sequence of locations displayed had been 24, 22, and 20, then switching to long integers should have caused the data at location 16 to be displayed. Instead, DVIEW displayed the long integer at location 14.

- If DVIEW was displaying data in reverse order and a file position from which data should have been displayed was explicitly specified, the first value displayed should have been from the specified location. Instead, DVIEW displayed data starting at two units earlier. For example, if you were displaying long integers in reverse order and you specified file position 20 as the new location from which data should be displayed, DVIEW would display data at location 12, then 8, then 4, etc.

- The "s" (search) command always searched forwards, regardless of the setting of the "direction" toggle. It should have searched backwards when the "direction" toggle denoted "backwards".

[UCR 91-133]

5.4.3. CONF on Apollo's DOMAIN/OS on Motorola MC68020/MC68881

On Apollo's DOMAIN/OS on Motorola MC68020/MC68881, if an environment variable was passed to the installation script, e.g.:

```
iix20.com \$MS
```

CONF did not properly expand the logical name and was not able to produce the ".bin" file. [UCR 91-38]

5.5. Debugger

5.5.1. Spurious Complaint about $descendantKilledExcpt

The debugger sometimes issued a spurious error message:

```
MAINSAIL interpreter: unexpected exception.
$exceptionName = MAINSAIL: Coroutine descendant killed
Continue with this exception (Yes or No):
```

[UCR 91-32]

5.5.2. Breakpoint on a Local Procedure with No Uses or Modifies Parameters on the IBM RISC System/6000

On the IBM RISC System/6000, if a breakpoint were placed on a procedure call with no uses or modifies parameters, execution would jump to an unpredictable location on return from the procedure call. [UCR 90-465]

5.5.3. Segmentation Violation Due to Cache Problems on the IBM RISC System/6000

On the IBM RISC System/6000, a segmentation violation sometimes occurred when continuing from a breakpoint. This was because the processor instruction and data caches were not properly synchronized. [UCR 90-465]

5.6. MAINEDIT

5.6.1. Display Module Name Not Updated in "eparms" When Invoked from Another Program

When MAINEDIT was invoked from another program that uses the editor (e.g., MAINDEBUG or ISQL/D), and a display module other than the default specified in "eparms" was given, "eparms" was not updated to make the specified display module the new default. [UCR 91-102]

5.6.2. MAINED -<n>.B and -<n>.F Commands

The MAINED -<n>.B and -<n>.F commands displayed one line fewer than they were supposed to (i.e., n - 1 lines) in the new window. [UCR 90-472]

5.6.3. MAINVI ":1,$ sub /x/y"

A MAINVI command of the form ":1,$ sub /x/y" issued a spurious error message. [UCR 91-24]

5.6.4. MAINVI and Inserting Page Marks

In insert mode, MAINVI normally accepted the CTRL-L key (even when that was the display module's abort character, under most circumstances), and inserted a page mark (along with an extra blank line). If the autoIndent option was set, however, it did not; it inserted a blank line instead.

CTRL-L now behaves as it should, always inserting just the eop character, never the extra blank line. [UCR 91-125]

5.6.5. MAINVI Error Messages from Unsupported Commands in ".exrc"

Errors encountered by MAINVI while reading ".exrc" (and only while reading ".exrc") are now ignored. Such errors are likely because commands unrecognized by MAINVI are commonly given for real vi. [UCR 90-290]

5.6.6. MEDT SWM Command

The MEDT SWM command without arguments did not work. [UCR 90-259]

5.6.7. FBORRO and FFRAME Display Modules

The FBORRO and FFRAME display modules did not work if they had to prompt for information. GPR_$INIT was called before the last prompt had been issued, causing FRRAME to abort and FBORRO to hang. [UCR's 90-23, 90-24]

5.7. MAINPM

5.7.1. Fatal Error When Could Not Find Module

MAINPM got a fatal error if it could not find a module when generating a report file. It now no longer tries to process statistics collected for the unfindable module. [UCR 90-489]

5.8. STREAMS

5.8.1. Fatal Exceptions from a Remote Module

There was a bug in the routine that handled fatal exceptions from a remote module. The result was that if a fatal error occurred in the remote module, and the local module handled this error, and fell out of the handler, then any attempt to dispose the module produced a protocol error. [UCR 91-25]

5.8.2. "server.log" File Problems

"server.log" was sometimes closed twice, resulting in a spurious error message. Also, the last entry in "server.log" was not always flushed to disk. [UCR 91-30]

6. Enhancements

6.1. Language

6.1.1. Innocuous Intmod Changes

Certain kinds of "innocuous" changes to intmods no longer necessarily require recompilation of other intmods and modules that relied on the intmods that were changed. This means, for example, that if an intmod A is used by an intmod B, and A is later changed in a way that does not affect B, the compiler would be able to use an old version of B with a newer version of A without blowing up.

Additions of new declarations and definitions to an intmod A are considered "innocuous" to an intmod B, since declarations and definitions added to A since the last time B was compiled obviously cannot affect B. Any changes to exisiting definitions or declarations that might be used by B, however, are not innocuous; in particular, adding new fields to classes or modules (except at the end) or changing the data type of a field or outer used by B is not innocuous.

The compiler is unable to detect automatically when an intmod change is innocuous. Thus, its default action is to complain about incompatible intmods as it has always done, although the message issued has been changed from an error to a warning. If a non-innocuous change were made, e.g., if a field added in the middle of an existing class or module, then all intmods that rely on the changed intmod would have to be recompiled, as was required before; however, it is the user's responsibility to determine when this is necessary.

6.1.2. $nextCommandLineArg

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

```
BOOLEAN
PROCEDURE   $nextCommandLineArg
                        (MODIFIES STRING commandLine;
                         PRODUCES STRING nextArg);
```

Table 6.1.2-1. $nextCommandLineArg

The current implementation of command line arguments does not accommodate null arguments or arguments containing spaces (unless the application implements its own syntax for these things). MAINSAIL now defines a syntax for specifying null arguments and arguments that contain blank space (blanks or tabs). If an argument contains blank space, it must be enclosed in double quotes (it can be enclosed in double quotes even if it does not contain blank space). Double quotes inside a "quoted" argument must be doubled, as in a MAINSAIL string constant. Null arguments are specified as a pair of double quotes (like the MAINSAIL null string).

The new procedure $nextCommandLineArg can be used to get the next argument from a command string, e.g., as obtained from $getCommandLine.

If commandLine is the null string, $nextCommandLineArg returns FALSE. Otherwise, it parses and removes the next argument from the command line.

Existing programs using $getCommandLine are not affected. However, the $commandLineArgs array, introduced as a temporary feature in the last release, is no longer supported.

6.1.3. New Global Symbol Facilities

New global symbol facilities have been added so that MAINEX subcommands can be used to set and examine string-valued global symbols. This makes it easier to establish program parameters in a bootstrap file or in the "site.cmd" file on the MAINSAIL directory (this file of MAINEX subcommands is always read when MAINSAIL starts execution).

Note: the global symbols under discussion are the language global symbols, as described by the class $globalSymbol, not compiler global symbols.

The following facilities have been added to the language (see Section 6.2.1 for the new MAINEX subcommands related to global symbols):

```
# class for string global symbols
CLASS($globalSymbol) $stringGlobalSymbol (
    STRING                  $value;
    );

STRING PROCEDURE $stringGlobalLookUp (STRING key);

PROCEDURE $stringGlobalEnter (STRING key,value);
```

$stringGlobalLookUp and $stringGlobalEnter look up and enter records of the class $stringGlobalSymbol, which has a single string value in the global symbol record. $stringGlobalLookUp verifies that key is the key for a record of the class $stringGlobalSymbol, and issues an error message if it is a key for a record of some other class. It returns the null string (without issuing an error message) if no record with that key is found. $stringGlobalEnter enters a $stringGlobalSymbol record with $key = key and $value = value.

An additional procedure for general global symbol use, not related to $stringGlobalSymbol, has also been added:

```
POINTER($globalSymbol) PROCEDURE $globalNext
    (POINTER($globalSymbol) p);
```

$globalNext returns the next global symbol in the global symbol table following p. If p is NULLPOINTER, it returns the first global symbol in the table. If p is the last symbol in the table, it returns NULLPOINTER. The order in which the $globalSymbol records are returned is not specified. [UCR 90-17]

6.1.4. Setting/Examining Time Zone Information from a Program

The MAINEX subcommands "DEFINETIMEZONE", "DSTENDRULE", "DSTNAME", "DSTOFFSET", "DSTSTARTRULE", "GMTOFFSET", and "STDNAME" have been provided in the past to set time zone information. This information can now be set or examined from a program as well as from MAINEX.

The following procedures are provided:

```
PROCEDURE    $setGMTOffset
                        (LONG INTEGER ii);

PROCEDURE    $setDSTOffset
                        (LONG INTEGER ii);

PROCEDURE    $setDSTStartRule
                        (STRING s);

PROCEDURE    $setDSTEndRule
                        (STRING s);

PROCEDURE    $setStdName (STRING s);

PROCEDURE    $setDSTName (STRING s);

PROCEDURE    $addDefinedTimeZone
                        (STRING tzName;
                         LONG INTEGER gmtOffset);

LONG INTEGER
PROCEDURE    $getGMTOffset
                        (PRODUCES OPTIONAL BOOLEAN
                             wasSet);

LONG INTEGER
PROCEDURE    $getDSTOffset
                        (PRODUCES OPTIONAL BOOLEAN
                             wasSet);

STRING
PROCEDURE    $getDSTStartRule
                        (PRODUCES OPTIONAL BOOLEAN
                             wasSet);

STRING
PROCEDURE    $getDSTEndRule
                        (PRODUCES OPTIONAL BOOLEAN
                             wasSet);

STRING
PROCEDURE    $getStdName (PRODUCES OPTIONAL BOOLEAN
                             wasSet);

STRING
PROCEDURE    $getDSTName (PRODUCES OPTIONAL BOOLEAN
                             wasSet);

STRING
PROCEDURE    $getDefinedTimeZones
                        (PRODUCES OPTIONAL BOOLEAN
                             wasSet);
```

Their correspondence to the existing MAINEX subcommands is as follows:

MAINEX Subcommand	Procedure to Set	Procedure to Examine
GMTOFFSET	$setGMTOffset	$getGMTOffset
DSTOFFSET	$setDSTOffset	$getDSTOffset
DSTSTARTRULE	$setDSTStartRule	$getDSTStartRule
DSTENDRULE	$setDSTEndRule	$getDSTEndRule
STDNAME	$setStdName	$getStdName
DEFINETIMEZONE	$addDefinedTimeZone	$getDefinedTimeZones

In each "$set" procedure, the value specified replaces the parameter's existing value. In the case of $addDefinedTimeZone, tzName and its offset from GMT are added to the list of known time zones.

The "$get" procedures, except for $getDefinedTimeZones, return the current value of the corresponding parameter. $getDefinedTimeZones returns a string describing the information on all known time zones in tabular form.

Calling any of the "$set" procedures causes $timeSubcomandsSet to become true (calling $addDefinedTimeZone does not, however). In the "$get" procedures, wasSet is set to the value $timeSubcommandsSet. [UCR 89-448]

6.1.5. Recovery from bind/new Errors

It is now possible to handle errors encountered by the system procedure bind and new and have the procedures try to allocate the module again. To do this, call $raiseReturn at the end of the handler for the $systemExcpt raised by bind or new. For example, to handle a call to bind that cannot find a module FOO by opening the library where FOO is supposed to reside and retrying the bind:

```
$HANDLE bind("FOO")
$WITH
    IF $exceptionName = $systemExcpt THENB
        openLibrary("libWithFoo.olb");
        $raiseReturn; # bind will try FOO again
        END
    EL  $raise;
```

[UCR 91-97]

6.1.6. $memoryManagementInfo

> TEMPORARY FEATURE: SUBJECT TO CHANGE

$memoryManagementInfo allows a program to tell the MAINSAIL runtime system that little or no garbage is being generated during a certain phase of execution. This is useful if a lot of memory is being allocated, but none of the

```
LONG INTEGER
PROCEDURE    $memoryManagementInfo
                          (INTEGER info);
```

Figure 6.1.6-1. $memoryManagementInfo

allocated memory will contain garbage. Allocation of a large amount of memory would normally trigger garbage collections, but if no garbage is being generated, the collections would serve no purpose. Calling $memoryManagementInfo reduces the likelihood that such unprofitable garbage collections will occur, although it does not completely prevent collections, as would incrementing $collectLock. Also, $memoryManagementInfo may be specified for either chunk or string garbage; $collectLock locks out both kinds of garbage collection.

The parameter info takes one of the following four values, with the meanings shown:

Value	Meaning
$noNewChunkGarbage	chunks being allocated will not be garbage
- $noNewChunkGarbage	undoes $noNewChunkGarbage
$noNewStringGarbage	strings being allocated will not be garbage
- $noNewStringGarbage	undoes $noNewStringGarbage

$memoryManagementInfo actually increments or decrements a counter, so calling it with the value - $noNewChunkGarbage or - $noNewStringGarbage does not guarantee that normal assumptions about garbage will resume; to ensure this, $memoryManagementInfo(- $noNewChunkGarbage) must be called exactly as many times as was $memoryManagement($noNewChunkGarbage); the same is true of $noNewStringGarbage.

6.2. Utilities

6.2.1. MAINEX Subcommands for String-Valued Global Symbols

The changes to the global-symbol-related system procedures are described in Section 6.1.3. This section describes the MAINEX subcommands that add and remove string-valued global symbols.

The new MAINEX subcommands are:

```
GLOBALREMOVE r       Undefine global symbol r
GLOBALSYMBOL r s     Define global symbol string r as s
GLOBALSYMBOL r       Show definition of global symbol r
GLOBALSYMBOL         Show all defined global symbols
```

"GLOBALSYMBOL r s" calls "$stringGlobalEnter(r,s)". r must be quoted using the double quote character if it contains spaces or tabs; s must be quoted if it contains trailing spaces or tabs or if it is the null string. To include a double quote character in a quoted string, double it, as in a MAINSAIL string constant.

"GLOBALSYMBOL r" displays the value of "$stringGlobalLookUp(r)".

"GLOBALSYMBOL" (with no arguments) displays all global symbols. String global symbols (those of the class $stringGlobalSymbol) are displayed with their values; others are displayed with a note that they are not $stringGlobalSymbol records.

"GLOBALREMOVE r" calls "$globalRemove(r)".

[UCR 90-17]

6.2.2. Examining Time Zone Information from MAINEX

The MAINEX subcommands "DEFINETIMEZONE", "DSTENDRULE", "DSTNAME", "DSTOFFSET", "DSTSTARTRULE", "GMTOFFSET", and "STDNAME" now display the current value of their parameters if no parameter is specified. [UCR 89-448]

6.3. MAINEDIT

6.3.1. Default MAINVI Key Mappings for FRAME and FFRAME Display Modules

By default, for the FRAME and FFRAME display modules, MAINVI now maps some raw key codes even if there is no ".mainvirc" file. The codes mapped are for the RETURN, BACKSPACE, and ABORT/EXIT keys, which are mapped to CTRL-M, DELETE, and ESCAPE, respectively. These defaults are established whether or not there is a ".mainvirc" file, but may be overridden by the ".mainvirc" file, if desired.

6.3.2. SUN-Specific Keypad Mapping for MEDT

A default SUN-specific keypad mapping has been added to MEDT, as shown in Figure 6.3.2-1.

6.4. STREAMS

6.4.1. Child Process Name Syntax

The format of the name passed to $openStream for SOCPRO and PTYPRO streams has been made more flexible to accommodate the passing of environment variables (on systems that support environment variables). The new syntax is:

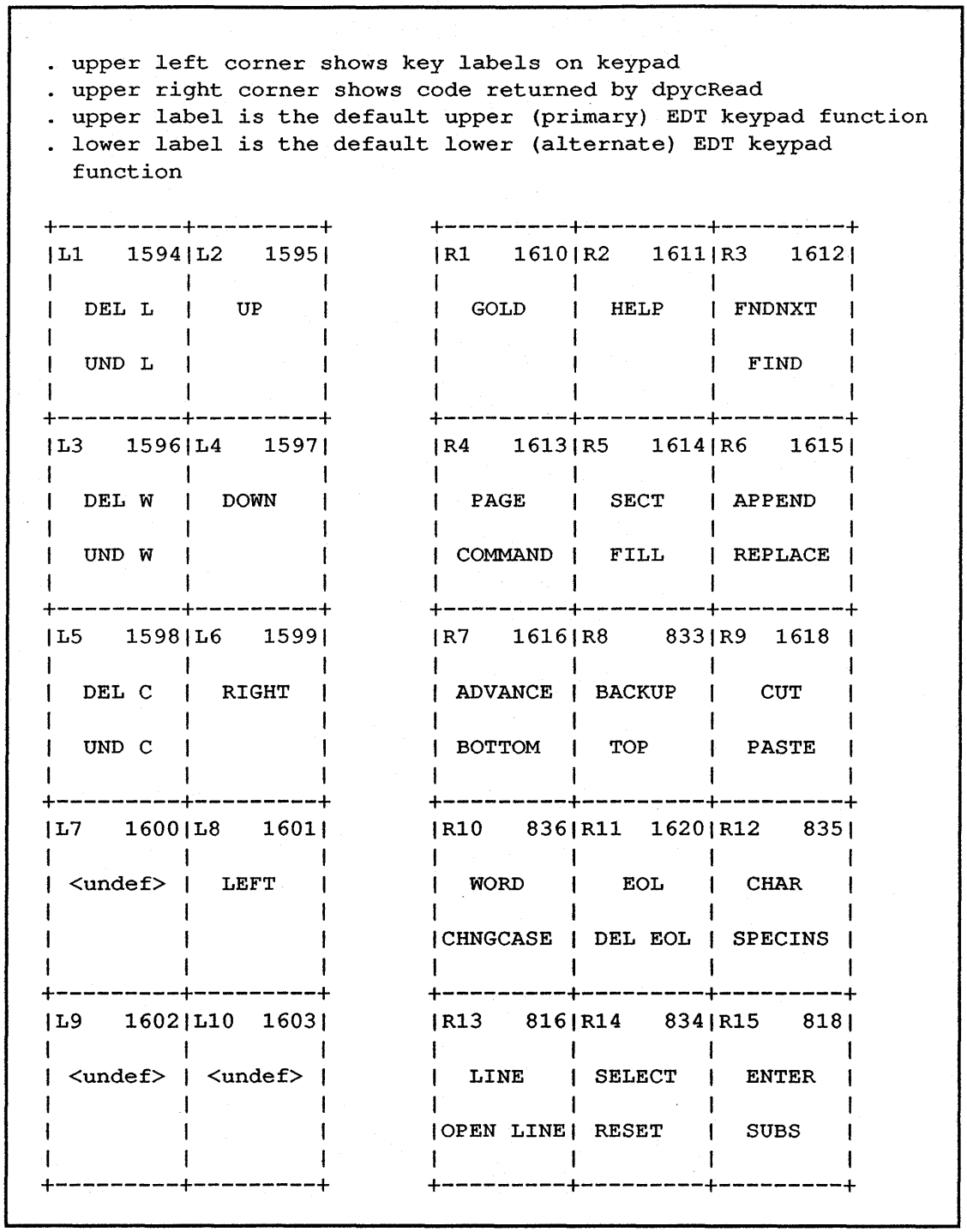


Figure 6.3.2-1. SUN-Specific Keypad Mapping for MEDT

```
socpro[(hostName)]>programName [programArgs]<eol>
[userName]<eol>
[password]<eol>
[startupDirectory]<eol>
[numberOfEnvironmentVariables]<eol>
[environmentVariable1]<eol>
   .
   .
   .
[environmentVariableN]
```

or, in the current release:

```
socpro>[hostname:]programName [programArgs]...
```

where the components in square brackets are optional, and trailing blank lines can be omitted.

The meanings of programArgs, userName, password, and startupDirectory are as previously documented. numberOfEnvironmentVariables is the number (in textual form) of environment variable lines that follow. Each environment variable has the form:

```
VARIABLENAME=value
```

For example, to create the process named "thisProgram", with arguments "hello" and "goodbye", running under the user ID "george", connected to "/user/dir", with the environment variables "ENV1" set to "/usr/baz", and "ENV2" set to "hostname", you would open the following stream:

```
$openstream(st,
    "SOCPRO>thisProgram hello goodbye" & eol &
    "george" & eol &
    "george's password" & eol &
    "/user/dir" & eol &
    "2" & eol &
    "ENV1=/usr/baz" & eol &
    "ENV2=hostname");
```

In addition, programName itself can be environment variable of the current program if it begins with a dollar sign. For example, if the environment variable "environVar" is defined as "foo", then:

```
$openstream(st,"SOCPRO>$environVar");
```

is equivalent to:

```
$openstream(st,"SOCPRO>foo");
```

NOTE: You must be superuser in order to specify non-null userName and password parameters.

6.4.2. Raw Internet Addresses

Raw internet addresses are now accepted wherever a TCP/IP host name would be required. For example, if the host named "PaloAlto" were at internet address 192.9.200.5, then "192.9.200.5:serviceName" would be accepted wherever "PaloAlto:serviceName" would be accepted.

6.4.3. Support for STREAMS on Intergraph's System V UNIX on Interpro 32C

STREAMS is now fully supported on Intergraph's System V UNIX on Interpro 32C. As on IBM's AIX on IBM RISC System/6000, however, all MAINSAIL bootstraps must consequently be linked with the "-lbsd" switch. Since the gethostbyaddr() system call does not work on Intergraph's System V UNIX on Interpro 32C, connection names are reported using raw internet addresses rather than host names.

6.4.4. $tooManyOpenFilesStr raised by $openStream and $acceptClient

A new exception, $tooManyOpenFilesStr, can be raised by calls to $openStream and $acceptClient. The exception is raised when it is impossible to open the stream or accept the client because no more stream handles are available. If the exception is handled by falling out of the handler, the call is aborted; otherwise, an error message is issued as before.

6.5. MAINKERMIT

A new version of MAINKERMIT, Version 3.1, is introduced in this release. It contains several new commands:

```
SHOWQUEUE       Show current transfer queue
DIAL            Edit the dial directory
CALL s          Call system s, where s is an entry
                  in the dial directory
```

and several new SET options:

```
SET BLOCKCHECK [1|2|3]
SET ERRORLOG <file  name>
SET HALFDUPLEX
SET FULLDUPLEX
SET TRANSLATION <ebcdic char code> <ascii char code>
SET PARITY
SET NOPARITY
SET ESCAPE <new escape character>
```

The "dial directory" is a list of host systems and their characteristics. It is maintained in the file "kdiald.ini" in the user's home directory. It should be edited only by using the MAINKERMIT "DIAL" command.

The "DIAL" command enters "dial mode", which provides the following commands:

```
INSERT     Insert new record
DELETE n   Delete record
EDIT n     Edit record number n
CALL s     Call system named s
SAVE       Save changes
QUIT       Return to Kermit prompt (exit dial mode)
EXIT       Return to Kermit prompt (exit dial mode)
```

Upon entry to dial mode, the current dial directory is displayed. Each entry contains a record number, a system name (which may be used as an argument to the "CALL" command, either in dial mode or at MAINKERMIT's top level), a baud rate to use when calling, a phone number to dial to reach the system, a modem prefix transmitted before the phone number, a modem suffix transmitted after the phone number, and the name of an optional take file (a file of MAINKERMIT commands) that is executed before dialing. The dial mode "INSERT" and "EDIT" commands prompt for the values of each of these parameters.

The new SET options have the following effects:

```
SET BLOCKCHECK [1|2|3]
    This has the standard Kermit meaning of setting the
    block checksum.  1 is a six-bit arithmetic checksum, 2
    is a 12-bit arithmetic checksum, and 3 is a 16-bit CRC.

SET ERRORLOG <file name>
    Save error messages in the specified file.

SET HALFDUPLEX
    Set Kermit to echo locally (used when talking to
    half-duplex system).

SET FULLDUPLEX
    Set Kermit not to echo locally (i.e., echo is remote).
    Used with full-duplex systems.

SET TRANSLATION <ebcdic char code> <ascii char code>
    Override default EBCDIC-ASCII conversion.  Codes are
    entered in decimal.

SET PARITY
    Indicate that local machine is using parity.

SET NOPARITY
    Indicate that local machine is not using parity.

SET ESCAPE <new escape character>
    Set the Kermit escape character (the character used to
    escape from the terminal emulator). The charater is
    typed as CTRL-<character entered>.  For example, the
    default escape character (CTRL-\) would be entered as
    "SET ESCAPE \".
```

It is now possible to abort a MAINKERMIT file transfer by typing the keyboard interrupt character (CTRL-C on many systems).

7. Known Problems

7.1. Some Versions of UNIX Kill Processes When Low on Memory

Some versions of UNIX, including IBM's AIX on IBM RISC System/6000 and Silicon Graphics' UNIX on MIPS R2000, kill processes when system swap space runs short, rather than simply disallowing the allocation of more memory by existing processes. Care must be taken when running important programs on such systems if there is any possibility of exhausting swap space.

The versions of UNIX with this problem, mostly UNIX System V R4, generally do not offer much control over the selection of which process is killed.

On IBM's AIX on IBM RISC System/6000, there is a setting for each user in the file "/etc/security/limits" that determines how much memory a user/process is allowed. That parameter is the rss value. The rss value can also be set using the "SMIT/Security and Users/Users/Change show Characteristics" menu item in the SMIT tool, by changing the value for Max physical memory (note: this value must be a multiple of 512). To give a user priority for real memory usage the rss value should be set to a number which is larger than what the application will try to allocate. This will allow that user to keep running when paging becomes low.

On the Silicon Graphics, the "limit" command can be used to limit the amount of virtual memory that will be allocated to the shell and its child processes. Determine (if possible) the amount of swap and physical memory that will be available during the execution of your program, then issue the command to specify a value less than that; this will ensure that no process is killed for lack of swap space. The problem with this strategy is that it requires predicting the memory requirements of all programs to be run during the execution of your program.

Other versions of UNIX may have similar problems; check your system documentation for more information. We recommend that you configure systems with this problem with ample swap space to avoid triggering the problem: figure out the maximum amount of memory that all jobs could reasonably be using at once, double that value for safety, then subtract the amount of physical memory available on the system, and use the resulting value as the amount of swap space to configure the system with.

7.2. $descendantKilledExcpt and the MAINSAIL STREAMS Scheduler

At present, it is especially important to handle informational exceptions carefully when running in conjunction with the MAINSAIL STREAMS Scheduler. In particular, a handler handling $descendantKilledExcpt raised because a scheduled coroutine is being killed should not attempt any action (such as file I/O) that might invoke the Scheduler. This is because the Scheduler is in an unusual state during $descendantKilledExcpt. This problem may be corrected in some future release of MAINSAIL.

7.3. PC Monitoring Status

7.3.1. AIX

On AIX on both the IBM System/370 Extended Architecture and the IBM RISC System/6000, PC monitoring is supported only on recent releases of the operating system. On the IBM System/370 Extended Architecture, the supported releases are release 1.2.600 and later; on the IBM RISC System/6000, release 03.01.0001.0003 and later.

One way to find out version status and upgrade information on AIX on the IBM RISC System/6000 is to run the command:

```
lslpp -h
```

which generates a table with the columns option name, state, event, date, release number, and user name. Look for a state of ACTIVE and an event of COMMIT to find the most recent upgrade for that option. For that option there will be a release number that looks something like 03.01.0001.0003. The 03.01 corresponds to the the AIX version 3.1 that is displayed when you log in. The 0001 and the 0003 correspond to upgrades.

7.3.2. PRISM

Due to a bug in the operating system, PC monitoring does not work on the PRISM. Apollo has been informed of the problem. Currently, attempting to use PC monitoring on this system has undefined effects and may crash MAINSAIL.

7.3.3. IBM's VM/XA SP CMS on IBM System/370

PC monitoring is not yet implemented on IBM's VM/XA SP CMS on IBM System/370. Currently, attempting to use PC monitoring on this system has undefined effects and may crash MAINSAIL.

7.3.4. Intergraph's System V UNIX on Interpro 32C and SCO's UNIX on HP Vectra with Intel 80386

On Intergraph's System V UNIX on Interpro 32C and SCO's UNIX on HP Vectra with Intel 80386, it is not possible to implement PC monitoring. Attempting to use PC monitoring on these systems issues an error message.

7.4. SIGFPE on Intergraph's System V UNIX on Interpro 32C

On Intergraph's System V UNIX on Interpro 32C, when a SIGFPE signal (generated by floating point arithmetic exceptions) occurs and the user program's signal handler returns, the UNIX signal handler does not pop off the correct number of bytes when trying to return control to the program where the signal occurred. This may manifest itself as a bus error, illegal instruction, or by hanging the affected program.

7.5. $userID on Intergraph's System V UNIX on Interpro 32C

On Intergraph's System V UNIX on Interpro 32C, $userID always returns the login user ID, never the effective user ID, regardless of whether $login is set in the ctrlBits parameter to $userID. This is because of a bug in the operating system.

7.6. SPARCstation Operating System Version Numbers

MAINSAIL runs on both versions 3.x and 4.x of the SunOS operating system on SPARC-based workstations. There are some incompatibilities between these two versions of SunOS that make it more difficult to build a MAINSAIL bootstrap on networks that contain both versions of the operating system. If you have only one version of SunOS for the SPARC (either all 3.x or all 4.x) on your network, then you will not have any of the problems described below, and you do not need to read this section any further.

The incompatibilites are two-fold:

1. MAINSAIL bootstraps built on 3.x will run under 4.x, but not vice versa.

2. All pieces of a bootstrap must be compiled, assembled, and linked under the same version of the operating system; bootstraps that contain pieces built under different versions of the operating system will not work correctly.

The first problem means that if you have a MAINSAIL system that will be used from both 4.x and 3.x nodes, then it must be installed on a 3.x node. The second problem means that when you are building any piece of a MAINSAIL bootstrap or code that is linked with a MAINSAIL bootstrap, you must always run "as", "cc", or "ld" (or whatever the local equivalents of those programs may be called) on the same version of the operating system as on the node where the MAINSAIL system is installed. If you follow the above recommendation to install your MAINSAIL systems on 3.x nodes, then you must always issue any bootstrap-making commands on 3.x nodes as well. If you accidentally build a bootstrap with pieces from different versions, the symptoms are unpredicatble, but are often manifested as file I/O errors.

7.7. sbrk() vs. malloc()

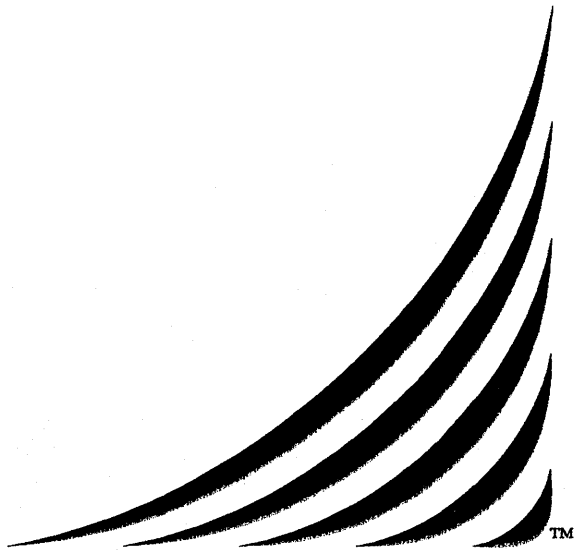
On SunOS versions 4.1 and later, sbrk() and malloc() are incompatible, and should not both be called from the same program. MAINSAIL uses sbrk() to allocate memory on SunOS versions before 4.1, and malloc() on versions 4.1 and later. Programmers who link C code with MAINSAIL through a MAINSAIL FLI should be aware that they should not call sbrk() on SunOS versions 4.1 and later.

On IBM's AIX on IBM RISC System/6000, there are two malloc()s: the old one and the new one. The new one is incompatible with sbrk(), which is still used by MAINSAIL on that operating system.

In order to get the new version of malloc(), you must make the following call:

```
mallopt(M_MXFAST,0)
```

Thus, the old version (which works OK with sbrk()) is the default. Programmers who link C code with MAINSAIL through a MAINSAIL FLI should be aware that they must not make the above mallopt() call on IBM's AIX on IBM RISC System/6000.



MAINSAIL® Release Notes

Version 12.14 Release

11 July 1991

XIDAK

8. Introduction

This release note documents the changes made between Versions 11.30 and 12.14 of MAINSAIL, except that it does not document features that were already described in the "MAINSAIL Release Notes, Version 12.10 Release", even if those features were not present in Version 11.30 of MAINSAIL. The Version 12.10 manual set (the most recently published complete set of documentation) should be used in conjunction with the following release notes in order to have a complete description of the features of Version 12.14:

- "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27",
- "MAINSAIL Release Notes, Version 11.28 Release",
- "MAINSAIL Release Notes, Version 11.30 Release",
- and the current release note.

Version 12.14 object modules cannot be run under previous Version 11 or Version 12 releases, and object modules from previous Versions cannot be run under Version 12.14; you must rebuild all intmods and recompile all object modules. In addition, the layout of data structures in memory has changed between Version 11 and Version 12; all Version 11 (but not older Version 12) Structure Blaster data images must be translated or remade.

9. Recent and Proposed Changes

9.1. Case of File Names

MAINSAIL now treats all file names (and search paths) as case-sensitive (all uses of $fileNamesAreCaseSensitive have been eliminated from the runtime system and compiler). The $directory and $fileInfo functions for systems where file names are not case-sensitive return lowercase names. The $attributes bit $fileNamesAreCaseSensitive is still defined and applies to the default disk module for that system. [UCR 85-377]

9.2. fileSize Parameter to "open"

The last parameter to open is an optional long integer fileSize. This parameter will be removed in the next release of MAINSAIL (it has not been examined by any existing device module for a long time; it originally specified the initial size of the file, but is not relevant on any modern file system).

9.3. Supported Platforms

The following MAINSAIL platforms that were supported under Version 11 are not currently supported under Version 12:

Platform Abbreviation	Platform Name
aeg	Apollo's Aegis on Motorola M68000
alnt	Alliant's CONCENTRIX on Motorola M68000
cms	IBM's VM/SP CMS on IBM System/370
hpux	HP's HP-UX on Motorola M68000
ipsc2	Intel's iPSC/2 System V UNIX on Intel 80386
spix	Bull's SPIX on Ridge 32
sun2	Sun Microsystems' SunOS on Motorola M68000

Note that M68000-based platforms have been dropped in favor of MC68020/MC68881-based platforms on the assumption that all users have converted to MC68020 and MC68881 or compatible processors, and that VM/SP CMS support has likewise been dropped in favor of VM/XA CMS (the extended addressing version of CMS). Apollo's Aegis operating system has been dropped in favor of Apollo's DOMAIN/OS.

If the lack of suport for any of the above platforms imposes any hardship, please contact XIDAK immediately.

10. Release Highlights

The most important new features introduced at this release are:

- Module names are no longer restricted in length to six characters.
- A new type of variable, the shared variable, is shared by all data sections of a given module.
- A new MAINPM command, "TRACECHUNKS", provides a detailed allocation history of each chunk allocated in a monitored area. This can be very useful in helping a programmer to determine how a program is using memory and where improvements in memory usage can be made.
- Certain interface procedures with all-constant arguments can be evaluated during compilation instead of at runtime if declared with the procedure qualifier "COMPILETIME".
- Some system procedures have been added to manipulate file directories (e.g., to create and delete directories and change the working directory).

11. Clarification of Documentation

11.1. Address Calculation of Modifies and Produces Arguments

It is not specified whether the addresses where modifies and produces procedure arguments are stored are calculated before or after the procedure call, or both. The address of a modifies argument is computed before the call, and may be recomputed in whole or in part after the call; the address of a produces argument may be computed before the call, and may be recomputed in whole or in part after the call.

This means that a procedure body should not change the value of any component of its modifies or produces arguments, since the address used after the call (to store the value being passed back to the caller) would vary depending on whether or not the address was calculated before or after the call.

For example, assume the declarations:

```
CLASS c (INTEGER i; POINTER(c) next);

POINTER(c) PROCEDURE foo (MODIFIES POINTER(c) p1,p2);

PROCEDURE bar (MODIFIES INTEGER i; PRODUCES OPTIONAL INTEGER j);

INTEGER i,j;

POINTER(c) p,q;

INTEGER ARRAY(*) ary;
```

In the procedure call:

```
foo(p,p.next)
```

the "p.next" affected may be calculated based on the value of p before the call to foo, or (since p is passed as a modifies parameter and is changed by the call) on the value of p after the call to foo; in fact, on input, it may be calculated based on the value of p before the call, and on output, on the value of p after the call.

In the procedure call:

```
bar(ary[i .+ 1])
```

"i .+ 1" may or may not be computed twice. In the call:

```
bar(ary[i + j],j)
```

the altered value of j may be used when storing the first argument into ary. In:

```
bar(foo(p,q).i)
```

foo may wind up being called twice.

In the first two calls to bar above, the effect is undefined if bar disposes ary, even if it subsequently reallocates it.

It is the responsibility of the programmer to avoid passing modifies or produces procedure arguments of which the address might be altered by the procedure call. One way to do this is to replace subscripted variables and field variables by simple variables when passed as modifies or produces parameters. Thus, to ensure that the first call to foo above changes "p.next" based on the original value of p, do:

```
q := p; foo(p,q.next)
```

or to ensure that the "p.next" changed is based on the resulting value of p, do:

```
foo(p,q); p.next := q
```

[UCR 90-173]

11.2. Informational Exceptions

Exceptions are usually raised when some condition occurs that requires action on the part of a procedure invocation ancestral to the current one. It is usually expected that an ancestral procedure will either abort the raising procedure (by falling out of a handler) or will repair the condition that provoked the exception and return to the place of the exception with $raiseReturn.

Some exceptions, however, do not necessarily require any action, but merely inform the ancestral procedures of something that has occurred. The system exception $descendantKilledExcpt falls into this category. System exceptions that are informational can be distinguished because they have both the $cannotReturn and $cannotFallOut bits set in $exceptionBits.

General-purpose error handlers should be written so as to handle informational exceptions properly, i.e., simply to call $raise with no parameters to propagate the exception. In the past, some MAINSAIL programmers have written general-purpose error handlers with the assumption that all exceptions with the $cannotReturn bit set in $exceptionBits should produce some sort of error message and fall out of the handler if they have not been handled by the time they are propagated to the general-purpose exception handler. This is not true of informational exceptions (which also have $cannotFallOut set in $exceptionBits); such exceptions should be propagated even by the "bottommost" exception handler.

11.3. MAINPM PC Monitoring

The MAINPM PC monitoring facility (invoked with the "PC" command) is a temporary feature subject to change.

11.4. MM "di" Command

The MM "di" (descriptor information) command displays information associated with each chunk descriptor in memory. Information for each descriptor includes the number of allocations, deallocations, and garbage collections of chunks with that descriptor. However, the number of calls to $disposeArea when a chunk of the given descriptor is located in the area disposed is NOT included in the deallocation count. This can lead to misleading displays; e.g., the "di" command may report 100 allocations and 50 deallocations for a given descriptor, which would normally mean that 50 objects associated with that descriptor remain in memory, when in fact none currently remain in memory because they were removed by $disposeArea.

If you need more a detailed history of memory usage (which takes calls to $disposeArea into account), you should use the new MAINPM "TRACECHUNKS" command; see Section 13.6.1.

11.5. Freeing the MRPCMOD Struct after Calling the _final Function in a C RPC Client

In a C RPC client, the MRPCMOD struct allocated by the _init function should be freed by the calling program after it passes the pointer to the MRPCMOD struct to the _final function. Previous versions of the documentation neglected to point this out, so users may have been writing C RPC clients with unnecessary memory leaks.

11.6. Specifying the Target System in $moduleInfo

The procedure $moduleInfo returns information about object modules based on a string, cmdLine, that has a form similar to the arguments to MODLIB's "DIRECTORY" command.

A target system must be specified in $moduleInfo calls if the target system is different from the host. Previous versions of the documentation failed to describe the syntax that allows the target system to be specified in cmdLine. If a target is specified, its abbreviation follows the library file name, and is separated from the library name by a comma (spaces are not allowed). So, cmdLine's syntax is really:

```
libName{,targetAbbreviation}{=fileName} {modList}
```

instead of:

```
libName{=fileName} {modList}
```

as previously documented.

11.7. Where $copyFile Leaves File Positions

$copyFile leaves the file positions of its arguments at the end of the copied region.

11.8. Detecting When a Child Process Has Exited

When manipulating a child process with MAINSAIL STREAMS, "$closeStream(child)" kills the child (as documented), and is in fact the ONLY way to kill a child process. It does not wait for the child to die voluntarily.

The correct way to wait in the parent for the child to exit cleanly is to do a "$readStream(child,s,errorOk)" and look for an $eos return. The $eos return means that the child has closed its end of the stream, which happens on exit.

EXCEPTION: The above does not work for the VMSMBX protocol, because there is no way to detect the child exiting on VMSMBX. If you have to run using VMSMBX, you can only run child processes that cooperate with the parent by returning some prearranged end-of-stream indication.

11.9. MAINED "..B" and "..F" Commands

The MAINED "..B" and "..F" commands (go to buffer/file and make all windows same size) may be prefixed with "+" or "-". The meanings of all versions of the commands are:

```
..B     Do a .B, then make all windows same size
+..B    Do a +.B, then make all windows same size
-..B    Do a -.B, then make all windows same size
..F     Do a .F, then make all windows same size
+..F    Do a +.F, then make all windows same size
-..F    Do a -.F, then make all windows same size
```

The "+" and "-" versions were inadvertently omitted from previous documentation.

11.10. cmdMatch and "?"

When the noResponse bit is set, cmdMatch normally uses the promptString as the match string, and does not write any information to logFile or read information from cmdFile. However, if promptString is "?", the contents of the cmds array are written to logFile, as when noResponse is not set. Therefore, programs that want to suppress this behavior should check to see if promptString is "?" before calling cmdMatch with noResponse set. [UCR's 90-80, 90-210]

11.11. $dateAndTimeCompare

The explanation of $dateAndTimeCompare in the Version 12.10 manuals of March, 1989 has its description of the return values backwards. $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 first time is after the second. [UCR 90-204]

11.12. VAX/VMS File and Directory Manipulation Procedures

The system procedure $delete and the new system procedures $renameDirectory and $copyDirectory sometimes spawn a process under VAX/VMS. For instance, if the file to be deleted is a directory (e.g., "foo.dir") and the delete protection bit has not been set (protection is "RWE" by default for directories), a process is spawned to execute the command "set protection=owner:RWED" before attempting to delete the directory. If the directory still contains files, VAX/VMS won't let you delete the directory reguardless of the protection bits. According to the "VMS User's Manual" for VMS Version 5.2 (June 1989; Digital Equipment Corporation order number AA-LA98B-TE), the SPAWN command has the following restrictions (page DCL-399):

> The RESOURCE_WAIT state is required to spawn a process. Requires TMPMBX or PRMMBX user privilege. The SPAWN command does not manage terminal characteristics. The SPAWN and ATTACH commands cannot be used if your terminal has an associated mailbox.

12. Bugs Fixed

12.1. Runtime System

12.1.1. Improved Memory Management Heurisitic

Under the old memory management heuristics, if several garbage collections took place early in the life of a MAINSAIL program but collected no garbage, MAINSAIL's estimate of the amount of garbage that could be reclaimed by a collection became very low. This meant garbage collections would not occur until a large amount of memory had been gotten from the operating system, which sometimes led to unnecessary thrashing. The new heuristics eventually force a garbage collection if enough extra memory is gotten, even if the previous several collections did not reclaim any memory. [UCR 90-52]

12.1.2. exit and File Closings

MAINSAIL would loop on exit if an open file could not be closed. If a file close fails, it now removes the file from the open file list so that it does not try to close it again. [UCR's 87-369, 87-424]

12.1.3. Conversion to String of Invalid Floating Point Values

Conversion of invalid floating point values to string sometimes went into an infinite loop or produced a garbage string. It now raises an arithmetic exception. [UCR's 87-383, 89-87]

12.1.4. $writeCalls Called from an Exception Handler

A bug occurred in $writeCalls if $writeCalls was called directly by an exception handler. It caused $writeCalls to display all stack frames as suspended.

12.1.5. Opening the Swap File

Inadvertent string space use in the runtime system when attempting to open the swap file could have led to an infinite loop. [UCR 85-226]

12.1.6. Counter Overflows

During long executions of MAINSAIL, several different kinds of counters could overflow. These overflows are now avoided. [UCR's 90-86, 90-88, 90-108]

12.1.7. copy of a Pointer to a Record with No Fields

When the pointer form of copy was passed a pointer to a record with no fields, it erroneously issued the error message "invalid pointer". [UCR 90-380]

12.1.8. $intmodInfo and $moduleInfo

$intmodInfo and $moduleInfo blew up in Version 11 when asked to get information about a module that was not in a library; in Version 12, the "* fileName" syntax did not work as advertised. [UCR's 90-85, 90-448]

12.1.9. $ranMod

In certain situations at MAINSAIL exit, $ranMod's final procedure could be called more than once, causing an error. The final procedure has been changed to prevent this situation. [UCR 90-238]

12.1.10. Invalid CMS File Names

Opening an invalid CMS file name containing more than one "." (e.g., "a.a.a") did not fail. It now issues the error message "Invalid argument". [UCR 88-248]

12.1.11. UNIX Terminal Input Buffer Overflow

When a really long line was read from the terminal on UNIX, every 256th character would be doubled due to a buffer overflow. [UCR's 89-372, 90-137]

12.1.12. VAX/VMS $directory and errorOK

errorOK passed to $directory on VAX/VMS failed to suppress some error messages. [UCR 90-119]

12.2. Compiler

12.2.1. Infinite Loop Compiling Generic Procedure with Repeatable Arguments

A bug in the compiler occurred when a procedure call of the form:

```
p.f(...)
```

occurred where f is a generic procedure, p is a pointer expression or module name, and "..." contains repeatable arguments. This caused the compiler to go into an infinite loop. [UCR 89-413]

12.2.2. Trailing Parameters Omitted from Forward or Interface Procedure

If trailing parameters were omitted from the procedure header accompanying the body of a procedure previously declared as a forward or interface procedure, the compiler did not issue an error message; instead, it generated bad code. This behavior has been fixed. [UCR's 90-239, 90-406]

12.2.3. Garbage Collections While Incrementally Compiled Module in Memory

If a garbage collection occurred while an incrementally compiled module's data sections were in memory, the memory manager could get a memory inconsistency. [UCR 90-71]

12.2.4. Extremely Long Series of Compilations

During a very long series of compilations within the same session, the compiler sometimes hung or got a segmentation violation. [UCR 90-435]

12.2.5. Optimized Case Statements in Inline Procedures

Bad code was generated when compiling optimized if a Case Statement occurred in an inline procedure called from within an Iterative Statement's WHILE clause. [UCR 90-84]

12.2.6. Optimizer Bug

An optimizer bug sometimes resulted in the message "substituteFields: expected tree in single basic block". [UCR 90-135]

12.2.7. Macro Argument Too Long

If a macro argument were longer than 32K characters (as can happen if a closing parenthesis is missing in a macro call), the compiler now aborts the current compilation instead of giving a fatal error message. [UCR 90-87]

12.2.8. $classDscrFor

The compiler did not issue an error message if $classDscrFor was called on a class name that had not yet been declared; instead, it caused a bogus class descriptor to be constructed. [UCR 90-205]

12.2.9. $compile

$compile no longer returns true if the optimizer detects an error. [UCR's 90-135, 90-306]

12.2.10. CLIPPER Code Generator

A bug in the CLIPPER entry FLI caused MAINSAIL to go into an infinite loop during initialization when the entry FLI was used. [90-69]

12.2.11. Intel 30836 Disassembler

Case Statements in inline procedures were disassembled incorrectly on the Intel 30836. [UCR 88-186]

12.2.12. IBM System/370 Code Generator

12.2.12.1. Exponentiation Overflow

Exponentiation with a long real or long integer base could overflow unnecessarily on the IBM. [UCR 90-103]

12.2.12.2. copy of One Character

Bad code was generated for calls to the charadr form of copy of the form "copy(c1,c2,1)".

12.2.13. M68000/MC68020 FLI

Problems with addressing limitations on the M68000/MC68020 prevented valid output from being generated for foreign modules with large numbers of procedures. [UCR 90-145]

12.2.14. M68000/MC68020 Entry FLI for C

The entry FLI for C on all M68000/MC68020-based platforms returned address and charadr results in the wrong register. [UCR 90-73]

12.2.15. MIPS R2000 Code Generator

12.2.15.1. Intermodule Calls with Many Arguments

Intermodule calls with large numbers of arguments sometimes clobbered some of the arguments. [UCR's 90-90, 90-99, 90-106, 90-107]

12.2.15.2. cvr(integerConstant)

When the argument to cvr or cvlr was a (long) integer constant, the code generator sometimes got the error "setEa: unexpected node, op = 2". [UCR 90-158]

12.2.15.3. NullArray Checking for Array Pseudo-Fields

The MIPS 2000 code generator did not check for nullArray when accessing array pseudo-fields, even when checking was set. [UCR 90-182]

12.2.15.4. Overflow Checking and Multiplication by Powers of Two

Overflow that resulted when multiplying by a power of two was not always detected even when ACHECK was set. [UCR 90-201]

12.2.16. PRISM Code Generator

12.2.16.1. NullArray Checking for Array Pseudo-Fields

The PRISM code generator did not check for nullArray when accessing array pseudo-fields, even when checking was set. [UCR 90-182]

12.2.17. PRISM, SPARC, and M88000 Code Generators

Optimized modules on the PRISM, SPARC, and M88000 sometimes blew up with a bus error because 8-byte variables had not been properly aligned to 8-byte boundaries. [UCR 90-181, 90-222, 90-424]

12.2.18. VAX-11 Code Generator

Bad code could be generated for expressions of the form "x GEQ 1" or "x < 1", where x is a (long) integer. [UCR 90-92]

12.2.19. VAX-11, Intel 80386, and MIPS 2000 Code Generators

Bogus "case statement too large" error messages were sometimes issued for case statements that contained unreachable code. [UCR 90-240]

12.2.20. VAX/VMS Entry FLI for C

The VAX/VMS entry FLI for C did not convert real procedure results to double precision as expected by C. [UCR 90-70]

12.2.21. Intel 80386 FLI Code Generators

Some of the Intel 80386 FLI code generators produced incorrect entry vectors, resulting in various errors. [UCR 90-325]

12.2.22. IBM RISC System/6000 Code Generator

12.2.22.1. Long Conditional Branches

Conditional branch instructions on the IBM RISC System/6000 can't branch to targets farther away than 32 Kbytes. Previously, if a conditional branch within a procedure wound up being farther than this, the code generator issued the error message "encodeShortBrhOrCallDsp: dspl out of range". Instead, it now emits an equivalent two-instruction sequence. [UCR 90-301]

12.2.22.2. Subscripted Variable as Modifies Argument

The code generator error "setReg: expected register" was elicited if:

- a subscripted variable was used as a modifies argument or the left-hand side of a dotted operation,
- the array was one-dimensional with a constant lower bound,
- the array variable's value happened to be already contained in a register when the subscripted variable's address was calculated for the second time (i.e., after the call for a modifies argument or in order to store the dotted operation's result).

[UCR 90-308]

12.2.22.3. Long Real Arguments Passed on the Stack

The code generator error "setReg: expected register" was elicited when a modifies or produces long real argument to a procedure call was passed on the stack instead of in an argument register. Furthermore, the bug would show up only if the argument's stack location happened not to be aligned on an 8-byte boundary. Since there are seven floating point argument registers that have to be tied up before any (long) real arguments are passed on the stack, this bug didn't show up very often. [UCR 90-385]

12.2.22.4. Floating Point Rounding

Reals on the IBM Risc System/6000 were truncated instead of rounded for some arithmetic operations. [UCR 90-444]

12.2.22.5. Procedures with Many Arguments and MAINPM

If a procedure had a lot of arguments and the procedure's module was compiled with the MAINPM timing options set, then on entry to or exit from the procedure, some of the arguments' values were clobbered. [UCR 90-310]

12.3. Utilities

12.3.1. MAINEX "SETFILENAME" Subcommand

The MAINEX "SETFILENAME" subcommand did not work unless the module name was specified in uppercase. [UCR 90-356]

12.3.2. LINCOM Positions

A bug in LINCOM caused it to be off by one in its line numbering if the differences between two files included an end-of-page character. The error wasn't permanent; LINCOM would resync and correct itself when it encountered the next end-of-page.

12.3.3. INTCOM and OBJCOM

The utilities INTCOM and OBJCOM were inadvertently omitted from many MAINSAIL systems shipped to customers. [UCR 90-347]

12.4. Debugger

12.4.1. "@" with No Context

When the debugger "@" command was invoked with no context (e.g., first thing upon entering the debugger), then after blanking the screen it would complain that it could not find CMDLOG and insisted on getting an existing file name. It now just enters CMDLOG without prompting. [UCR 87-388]

12.4.2. Misremembered Commands

The line-oriented debugger sometimes misapplied modifiers from previous commands (e.g., treated "N" as ".N" if a previous command contained a ".") when executing the <eol> (repeat last command) command. [UCR's 87-445, 88-134]

12.5. MAINEDIT

12.5.1. Strings Written to CMDLOG

Strings written to CMDLOG that were located in reclaimed static space or a disposed area would have caused the editor to display garbled text. MAINEDIT now ensures that strings written to CMDLOG are copied to $defaultArea if necessary. [UCR 86-497]

12.5.2. <eol> before a Blank in Insert Mode

When in insert mode with a blank immediately to the right of the cursor, an <eol> would break the line but put the cursor at the first non-blank character on the line. The cursor is now correctly positioned at the blank that begins the new line. [UCR 88-35]

12.5.3. Undesirable "Replace existing file?" Prompts

The "Replace existing file?" prompt issued when saving a file has now been suppressed for certain kinds of files where it is irrelevant, e.g., files accessed through the NUL device module. [UCR's 84-652, 86-812]

12.5.4. MAINVI and Replacement String Problems

In the ":<addressRange>s/<searchString>/<replaceString>/" command, a null <replaceString> was not accepted by MAINVI. [UCR 89-27] Also, ":<addressRange>s/<searchString>$/<replaceString>/" deleted the end-of-line at the end of <searchString>, which was inconsistent with real vi. [UCR 89-28]

12.5.5. MAINVI "dw", "cw" Commands

The MAINVI "dw" and "cw" commands deleted an end-of-line character if it followed the current word, inconsistently with real vi. [UCR 89-43]

12.5.6. MAINVI ":z^" Command

The MAINVI ":z-" and ":z^" commands are now implemented as in vi. [UCR 88-296]

12.6. Structure Blaster

12.6.1. PDF $structureRead

$structureRead on a file opened for PDF I/O where $charsInArea was set and the specified area was not the default error blew up with an addressing error. [UCR 90-260]

12.7. STREAMS

12.7.1. $becomeServer

The POINTER($stream) form of $becomeServer was inadvertently killing the coroutine in which it ran when the module was disposed.

12.7.2. Underscore in Node Names in Service Protocol Table

The underscore character ("_") is now legal in node names in the service protocol table file. [UCR 89-42]

12.7.3. $openStream and Command Line Arguments

$openStream on UNIX passed through at most nine of the command line arguments. It has now been fixed to pass them all. [UCR 90-291]

12.7.4. $newRemoteModule Errors on UNIX

$newRemoteModule failures were improperly handled on UNIX, triggering further errors. [UCR 89-398]

12.8. MAINKERMIT

12.8.1. SUNVIEW and the MAINKERMIT Terminal Emulator

A bug in the MAINKERMIT terminal emulator caused certain systems to drop characters when run under the SUNVIEW windows.

13. Enhancements

13.1. Language

13.1.1. Shared Variables

The declaration of an outer (non-interface) variable or local own variable may now be preceded with the keyword "$SHARED". The value of such a variable is shared among all data sections (both bound and unbound) for the module during a given execution. Like outer variables, shared variables initially have the value Zero.

Shared variables are allocated when the first data section of the module is allocated. They are allocated in a structure called the "shared data section", which is shared by all instances of the module (and is not to be confused with a normal data section; it contains ONLY the module's shared variables). The shared data section (and the shared variable values in it) persists until the end of the MAINSAIL execution or until the module or string form of dispose is used to dispose the module (disposing or unbinding any single data section of the module does not deallocate the shared data section).

A local own variable declared with just the keyword "$SHARED" is syntactically like an "OWN" variable; i.e., its identifier can be used only within the procedure, even though its value persists between invocations of the procedure. The two keywords "$SHARED OWN" in a variable declaration are equivalent to just "$SHARED".

The Structure Blaster does not write out or read in shared variables when it writes or reads a data section.

Example of shared variable declaration:

```
$SHARED INTEGER i; # Accessible from all instances of
                   # the current module
```

13.1.2. User Compiletime Procedures

The way in which the compiler implements compiletime procedures has been enhanced. Before, every compiletime procedure required special code in the MAINSAIL compiler. Now, though such special code is still needed for many compiletime procedures, others can be handled by having the compiler call the procedure in the same manner that the debugger can be used to call procedures. Users can now provide compiletime procedures by declaring them with the qualifer "COMPILETIME", as described below.

An example of when special code is needed is for "first(s)", where s is a string constant. The value obtained by using first(s) on the host machine (on which the compiler is running) gets the host character code for the first character of s, which must then be converted to the target character code (if the target system has a different

character set from the host system). An example of when special code is not needed is "$cvbo(s)" where s is a string constant, since scanning s for "TRUE" or "FALSE" can be done on the host system.

If a user-declared compiletime procedure P is invoked with all specified arguments constant (evaluatable at compiletime), and there is no special code in the compiler to handle P, then the compiler attempts to carry out the call to P at compiletime. In this case, P must be an interface procedure of some module M. The compiler binds M, and then calls M.P with the constant arguments. An error occurs if M cannot be bound. Thus for the user to contribute a compiletime procedure, the procedure must be an interface procedure of a module whose objmod can be found during compilation (e.g., if it resides in a library, that library must be open). It is the user's responsibility to be sure that invoking the procedure on the host system has the desired effect. For example, floating point computations may not have the same precision on the host as they would on the target.

All modifies and produces arguments must be omitted (and hence optional) in order to qualify as "constants" (thus, compiletime procedure calls do not give a way to modify or produce a variable (or macro identifier) at compiletime).

If a compiletime procedure P is typed, the call is replaced with the value returned (this does not work for address, charadr, or pointer procedures unless the result is discarded, i.e., unless the call is a Procedure Call Statement).

A compiletime procedure can of course also be invoked without all-constant arguments, in which case it behaves as if it had not been declared compiletime, and so is processed at runtime like any other procedure.

Because compiletime procedures must be invoked in a module other than the one currently being compiled, compiletime procedures may impose an order in which modules must be compiled, and could lead to an impossible order. For example, suppose module M1 has a compiletime procedure P1, and module M2 has a compiletime procedure P2. If M1 makes a compiletime call to P2, and M2 makes a compiletime call to P1, then neither module can be compiled before the other.

The following system procedures have been changed to be compiletime procedures:

Procedure	Comments
$cvbo	
cvs	boolean, bits, and (long) bits forms
$length	boolean, (long) integer, and (long) bits forms
$formParagraph	
$hash	
compare	standard string compare
cvcs	compiletime only if the arg char maps to a host char which in turn maps to the arg char
$typeName	

13.1.3. Long Module Names

Module names are no longer restricted in length to six characters.

Since the compiler forms the output file intmod and objmod name from the module name, a long module name could result in a file name which is too long to be legal on some operating systems. This problem does not arise if the intmod and objmod are put into a library. It is the user's responsibility to deal with this situation if it arises. (This problem was the reason that the six-character limit was imposed on module names in the first place.)

13.1.4. New File Directory Manipulation Procedures

```
BOOLEAN
PROCEDURE    $createDirectory
                        (STRING directoryName;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL STRING msg);

BOOLEAN
PROCEDURE    $deleteDirectory
                        (STRING directoryName;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL STRING msg);

# Following is new generic instance of $currentDirectory
STRING
PROCEDURE    $currentDirectory
                        (OPTIONAL STRING devModName;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL STRING msg);

BOOLEAN
PROCEDURE    $setCurrentDirectory
                        (STRING directoryName;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL STRING msg);

BOOLEAN
PROCEDURE    $renameDirectory
                        (STRING oldDirName,newDirName;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL STRING msg);
```

Table 13.1.4-1. New File Directory Manipulation Procedures (continued)

```
BOOLEAN
PROCEDURE    $copyDirectory
                        (STRING oldDirName,newDirName;
                         OPTIONAL BITS ctrlBits;
                         PRODUCES OPTIONAL STRING msg);

STRING
PROCEDURE    $composePath
                        (OPTIONAL STRING
                             pathPrefix,pathComponents,
                             devModStr;
                         OPTIONAL BITS ctrlBits);

BOOLEAN
PROCEDURE    $decomposePath
                        (STRING path;
                         PRODUCES OPTIONAL STRING
                             pathPrefix,pathComponents,
                             devModStr;
                         OPTIONAL BITS ctrlBits);

STRING
PROCEDURE    $composeFileName
                        (STRING base;
                         OPTIONAL STRING
                            extension,devModStr,path;
                         OPTIONAL BITS ctrlBits);

BOOLEAN
PROCEDURE    $decomposeFileName
                        (STRING fileName;
                         PRODUCES OPTIONAL STRING
                            base,extension,devModStr,path;
                         OPTIONAL BITS ctrlBits);
```

Table 13.1.4-1. New File Directory Manipulation Procedures (end)

Several new directory-related procedures have been added. These procedures are not guaranteed to work for all device modules; e.g., they are not supported for INTLIB, MODLIB, CMS DISK, TTY, MEM, etc. All of the procedures listed below produce an error string; if not supported, the first line of the error string is "not supported":

Procedure	Purpose
$createDirectory	create a directory
$deleteDirectory	delete a directory
$currentDirectory	new instance of $currentDirectory: accepts a device prefix
$setCurrentDirectory	change the current directory
$renameDirectory	rename a directory
$copyDirectory	copy a directory
$composePath	compose a directory name from its path name components
$decomposePath	reduce a directory name to its path name components
$composeFileName	compose a file name from its path name components
$decomposeFileName	reduce a file name to its path name components

[UCR's 85-412, 85-456, 88-269, 88-292]

All the new procedures are subject to change if their interfaces prove unsatisfactory.

13.1.4.1. $createDirectory

TEMPORARY FEATURE: SUBJECT TO CHANGE

$createDirectory creates a directory named directoryName. If the directory is to be created by a device module other than the default disk device module, the device prefix should be prepended to the directory name and separated from it by the string $devModBrkStr. For example, if $devModBrkStr is ">" and the directory "foo/bar" is to be created by a device module D, directoryName should be "D>foo/bar".

$createDirectory returns true on success, false on failure. If it fails, msg is set to an error message describing the failure.

Valid ctrlBits bits for $createDirectory are errorOK, alterOK, and $useOriginalFileName. If errorOK is set, then errMsg is not called when $createDirectory fails. If alterOK is set, then if a file or directory of the specified name already exists, it is deleted without prompting the user; otherwise, if a file or directory of the specified name exists, the user is prompted with "OK to delete <name>" before the existing file or directory is replaced. If $useOriginalFileName is set, no logical file name or file searchpath substitution is performed on directoryName.

13.1.4.2. $deleteDirectory

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

$deleteDirectory deletes a directory named directoryName. If the directory is to be deleted by a device module other than the default disk device module, the device prefix should be prepended to the directory name and separated from it by the string $devModBrkStr. For example, if $devModBrkStr is ">" and the directory "foo/bar" is to be deleted by a device module D, directoryName should be "D>foo/bar".

If directoryName contains files, the files are also deleted. If directoryName contains directories, the directories are recursively deleted.

$deleteDirectory returns true on success, false on failure. If it fails, msg is set to an error message describing the failure.

Valid ctrlBits bits for $deleteDirectory are errorOK and $useOriginalFileName. If errorOK is set, then errMsg is not called when $deleteDirectory fails. If $useOriginalFileName is set, no logical file name or file searchpath substitution is performed on directoryName.

13.1.4.3. $currentDirectory

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

The new generic instance of $currentDirectory accepts a string as its first argument, which is a device prefix. $currentDirectory returns the name of the current directory for the specified device. devModName should be a device prefix followed by the string $devModBrkStr; i.e., to get the directory for a device D on a system where $devModBrkStr is ">", use the string "D>".

13.1.4.4. $setCurrentDirectory

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

$setCurrentDirectory sets the current directory to directoryName. If directoryName includes a device prefix, then the current directory for that device is set; otherwise, the current directory for the default disk device (i.e., the operating system's file system) is set. The device prefix, if present, should be prepended to the directory name and

separated from it by the string $devModBrkStr. For example, if $devModBrkStr is ">" and the directory "xxx/yyy" is to become the current directory for a device module D, directoryName should be "D>xxx/yyy".

$setCurrentDirectory returns true on success, false on failure. If it fails, msg is set to an error message describing the failure.

Valid ctrlBits bits for $setCurrentDirectory are errorOK and $useOriginalFileName. If errorOK is set, then errMsg is not called when $setCurrentDirectory fails. If $useOriginalFileName is set, no logical file name or file searchpath substitution is performed on directoryName.

13.1.4.5. $renameDirectory

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

$renameDirectory renames a directory named oldDirName to newDirName. If the directory is to be renamed by a device module other than the default disk device module, the device prefix should be prepended to both oldDirName and newDirName, separated from the names by the string $devModBrkStr. For example, if $devModBrkStr is ">" and the directory "foo/bar" is to be renamed to "baz/gaz" by a device module D, oldDirName should be "D>foo/bar" and newDirName should be "D>baz/gaz".

$renameDirectory operates recursively if oldDirName itself contains directories; i.e., those directories also appear in newDirName after the call to $renameDirectory.

$renameDirectory returns true on success, false on failure. If it fails, msg is set to an error message describing the failure.

Valid ctrlBits bits for $renameDirectory are errorOK, alterOK, and $useOriginalFileName. If errorOK is set, then errMsg is not called when $renameDirectory fails. If alterOK is set, then if a file or directory named newDirName already exists, it is deleted without prompting the user; otherwise, if newDirName exists, the user is prompted with "OK to delete <newDirName>" before the existing file or directory is replaced. If $useOriginalFileName is set, no logical file name or file searchpath substitution is performed on oldDirName or newDirName.

13.1.4.6. $copyDirectory

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

$copyDirectory copies a directory named oldDirName to newDirName. If the directory is to be copied by a device module other than the default disk device module, the device prefix should be prepended to both oldDirName and newDirName, separated from the names by the string $devModBrkStr. For example, if $devModBrkStr is ">" and

the directory "foo/bar" is to be copied to "baz/gaz" by a device module D, oldDirName should be "D>foo/bar" and newDirName should be "D>baz/gaz".

$copyDirectory copies all the files contained in oldDirName into newDirName. It operates recursively if oldDirName itself contains directories; i.e., copies of the contained directories also appear in newDirName after the call to $copyDirectory.

$copyDirectory returns true on success, false on failure. If it fails, msg is set to an error message describing the failure.

Valid ctrlBits bits for $copyDirectory are errorOK, alterOK, and $useOriginalFileName. If errorOK is set, then errMsg is not called when $copyDirectory fails. If alterOK is set, then if a file or directory named newDirName already exists, it is deleted without prompting the user; otherwise, if newDirName exists, the user is prompted with "OK to delete <newDirName>" before the existing file or directory is replaced. If $useOriginalFileName is set, no logical file name or file searchpath substitution is performed on oldDirName or newDirName.

13.1.4.7. $composePath

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

$composePath takes three strings in the (device-specific) format produced by $decomposePath and returns a syntactically correct (device-specific) directory path name. The pathComponents string is expected to include only directory components, i.e., is not expected to include the leaf file name; use $composeFileName to compose a file name including a leaf file name.

The only valid ctrlBits bit for $composePath is errorOK. If set, errMsg is not called when an error occurs.

13.1.4.8. $decomposePath

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

$decomposePath takes a directory path name and reduces it to a prefix, path components, and a device module string devModStr. The path string is expected to include only directory components, i.e., is not expected to include the leaf file name; use $decomposeFileName to decompose a file name including a leaf file name.

The exact format of the decomposed path components is specific to the device, but in a tree-structured file system, the directories in the path are normally produced in pathComponents separated by spaces. pathPrefix often represents a node name on file systems where more than one node is accessible; pathComponents usually represents the directory path name (for tree-structured directories) exclusive of the node name.

The only valid ctrlBits bit for $decomposePath is errorOK. If set, errMsg is not called when an error occurs.

13.1.4.9. $composeFileName

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

$composeFileName takes base, extension, device module, and path strings in a (device-specific) format. The path format is that produced by $composePath for the device. $composeFileName returns a syntactically correct (device-specific) file name, such as could be passed to the system procedure "open".

The only valid ctrlBits bit for $composeFileName is errorOK. If set, errMsg is not called when an error occurs.

13.1.4.10. $decomposeFileName

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

$decomposeFileName takes a file name and reduces it to a base, extension, a device module string devModStr, and a path. The format of the path is the same one produced by $composePath for the device.

The only valid ctrlBits bit for $decomposeFileName is errorOK. If set, errMsg is not called when an error occurs.

13.1.5. Getting Error Messages from open, close, and $createUniqueFile; New devModStr, path, and extension Parameters to $createUniqueFile

```
BOOLEAN
PROCEDURE    open          (PRODUCES POINTER(textFile) f;
                            STRING fileName; BITS openBits;
                            PRODUCES OPTIONAL STRING msg);

BOOLEAN
PROCEDURE    open          (PRODUCES POINTER(dataFile) f;
                            STRING fileName; BITS openBits;
                            PRODUCES OPTIONAL STRING msg);
```

Table 13.1.5-1. open and $createUniqueFile (continued)

```
BOOLEAN
PROCEDURE   close       (MODIFIES POINTER(file) f;
                         OPTIONAL BITS closeBits;
                         PRODUCES OPTIONAL STRING msg);

BOOLEAN
PROCEDURE   $createUniqueFile
                        (PRODUCES POINTER(textFile) f;
                         BITS openBits;
                         PRODUCES OPTIONAL STRING msg;
                         OPTIONAL STRING
                             devModStr,path,extension);

BOOLEAN
PROCEDURE   $createUniqueFile
                        (PRODUCES POINTER(dataFile) f;
                         BITS openBits;
                         PRODUCES OPTIONAL STRING msg;
                         OPTIONAL STRING
                             devModStr,path,extension);
```

Table 13.1.5-1. open and $createUniqueFile (end)

The system procedures open, close, and $createUniqueFile now produce an error string, msg, indicating why the open or close failed.

The first line of the error string for open and $createUniqueFile is one of the following strings for which identifiers have been predefined (XIDAK reserves the right to add to this list):

Identifier	Value
$noSuchFileStr	"no such file"
$tooManyOpenFilesStr	"too many open files"
$noPermissionStr	"no permission"
$noSpaceStr	"no space"
$invalidArgumentStr	"invalid argument"
$otherErrorStr	"other error"

The error string may contain additional lines which give more detailed information about why the open failed. When testing for one of these strings, always use the predefined identifiers, since the text of the strings is subject to change.

close now returns false if and only if an error occurred while closing the file. The format of msg produced by close is not specified.

[UCR's 84-431, 85-418, 86-48, 86-801, 86-970, 90-297]

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

$createUniqueFile now allows the caller to specify the device module, path, and extension of the file created. The meaning of these parameters is device-specific, but is the same as specified to $composeFileName for the device. Each of these parameters that is non-zero overrides $createUniqueFile's default value for that parameter. For example, to cause a temporary file to have the extension "xyz" (instead of the default "tmp" used for most devices):

```
$createUniqueFile(f,output,msg,"","","xyz")
```

The parameters devModStr, path, and extension are considered temporary features subject to change.

13.1.6. Effective User ID and $userID

Some systems distinguish between a login user ID (the user ID under which the current user logged in) and the effective user ID (if the user ID was somehow changed during the login session). A new ctrlBits bit, $login, is now allowed in calls to $userId. If $login is specified, the login user ID is returned, if available; otherwise, the effective user ID is returned, if available.

It is advisable always to check $userID's return for the null string, since it is fairly common for either the effective user ID or the login user ID to be unavailable, depending on the operating system.

Because a known bug in the operating system on Intergraph's System V UNIX on Interpro 32C prevents the effective user ID from being available if the Yellow Pages is running, the login user ID is always returned in the current release. [UCR's 89-234, 90-378]

13.1.7. $hasFileVersions and $fileInfo

The bit $hasFileVersions is now maintained on a per-device-module basis. $hasFileVersions is still set in $attributes for each system and applies to the default disk device module for that system.

$hasFileVersions may be set in the $fileAttr field of the $fileInfoCls record returned by $fileInfo if the device on which the file resides has file versions (note that $fileAttr is considered a temporary feature). Similarly, a new long bits field, $deviceAttributes, has been added to the file record. Programs may examine this field to determine characteristics of the file device; the only bit that is currently set in it is $hasFileVersions.

[UCR's 86-761, 86-812]

13.1.8. $clearFileCache

$clearFileCache now writes the current buffer of its file argument if the file is not cached. Before, it took no action if the file was not cached. [UCR's 88-210, 88-238]

13.1.9. $flush

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

```
BOOLEAN
PROCEDURE   $flush      (POINTER(file) f);
```

Table 13.1.9-1. $flush

$flush writes all dirty buffers for the specified file f, whether or not f is cached.

[UCR 88-254]

13.1.10. $commandLineArgs

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

```
# system variable
STRING ARRAY(0 TO *) $commandLineArgs;
```

Table 13.1.10-1. $commandLineArgs

The new system variable $commandLineArgs records the arguments passed to the program from the operating system command line in a different format from $getCommandLine. $getCommandLine returns all arguments concatenated into a single string with spaces as separator characters, thereby losing information on some operating systems (such as UNIX) which permit spaces within a single argument. $commandLineArgs records each argument as a separate element of the array, starting with the program name (if available) as argument zero.

Unlike the $getCommandLine mechanism, $commandLineArgs is not modified by prompts given at the MAINEX asterisk prompt or by calls to $setCommandLine or $invokeModule. It records only the line given in response to the operating system prompt at which MAINSAIL was invoked.

The effects of changing $commandLineArgs or its contents are undefined. [UCR 88-156]

13.1.11. $changeAreaParms

```
PROCEDURE    $changeAreaParms
                          (POINTER($area) area;
                           LONG BITS attrBitsToSet;
                           OPTIONAL LONG BITS
                               attrBitsToClear;
                           OPTIONAL LONG INTEGER
                               strSpcChars);
```

Table 13.1.11-1. $changeAreaParms

$changeAreaParms changes the area parameters for area. The valid bits for attrBitsToSet and attrBitsToClear are the same as the valid bits for the attr parameter to $newArea; the resulting attribute bits for the area are given by:

```
(oldAttrBits IOR attrBitsToSet) CLR attrBitsToClear
```

The strSpcChars parameter specifies the size of area's string space and has the same effects as the strSpcChars parameter to $newArea.

13.1.12. $mergeArea

```
PROCEDURE    $mergeArea   (POINTER($area) dstArea;
                           MODIFIES REPEATABLE
                               POINTER($area) srcArea);
```

Table 13.1.12-1. $mergeArea

$mergeArea merges the contents of srcArea into dstArea (i.e., all chunks and strings in srcArea wind up in dstArea), then gets rid of the srcArea area and sets the srcArea parameter to nullPointer.

The attribute bits for dstArea are unaffected by the attribute bits for srcArea, except that the following bits are cleared in dstArea if they are not set for srcArea:

```
$noCollectablePtrs
$noCompactablePtrs
$noCollectableStrs
```

In other words, if either the source or destination area might contain collectable or compactible data before the call to $mergeArea, the destination area is treated as if it might contain such data after the call to $mergeArea.

13.1.13. $reclaim

```
$ALWAYSINLINE
PROCEDURE    $reclaim     (STRING s;
                           POINTER($area) ap);
```

Table 13.1.13-1. $reclaim

$reclaim is a low-level string space manipulation procedure that can be used to enhance efficiency in certain situations. However, incautious use of $reclaim can corrupt MAINSAIL's string space; this procedure should be used only by those who have a good understanding of what it does.

$reclaim reclaims the characters of a string s if s is at the top of the area ap's string space.

The behavior of $reclaim is undefined unless it conforms to the following restrictions:

- The string s must have been constructed using a system procedure that does not take any strings as uses or modifies arguments (such as non-string forms of cvs). Do not call $reclaim if s has been constructed with, for example, the concatenation operator ($concat), string forms of cWrite, string forms of scan, or $dup, since all of these take strings as uses or modifies arguments. If s has been constructed with a procedure that takes a string uses or modifies argument, it is possible to free strings that were not intended, thereby corrupting string space.

- The string s must be in a user-created area, not in $defaultArea or in any other area created by the runtime system. System procedures may put strings in $defaultArea (or other system areas, if any) at unpredictable times, and reclaiming these strings may have undefined effects. $inArea should be called to verify that s is in the area ap before $reclaim is called.

Example:

```
s := cvs(lr,'0,myArea);
...
IF $inArea(s,myArea) THEN $reclaim(s);
```

13.1.14. Constant Array Pseudo-Fields

The bounds pseudo-fields of an array ary (i.e., ary.lb1, ary.ub1, ary.lb2, etc.) are now evaluated at compiletime if declared as a constant. This is particularly useful when an array is being allocated in a specified area, since in that case the bounds must be specified as arguments to new. Before, the constant bounds had to be repeated in the call to new. If the bounds were changed in the program, all the corresponding calls to new had to be found and changed also (unless macro constants were used). Now it is possible to do the following:

```
INTEGER ARRAY(1 TO 10) ary;
. . .
new(ary,ary.lb1,ary.ub1,area); # no need to use constants 1 and 10
```

ary.$arrayType and ary.$dimension are also evaluated at compiletime if the type and number of dimensions, respectively, are specified in ary's array declaration. The effect is undefined if an array variable is declared one way but then made to point to an array with different characteristics.

ary.name is not evaluated at compiletime.

13.1.15. VAX/VMS-Specific Procedures

13.1.15.1. $trnLnm

```
BOOLEAN
PROCEDURE   $trnLnm     (LONG BITS attr;
                         STRING tabNam,logNam;
                         PRODUCES STRING name;
                         PRODUCES OPTIONAL STRING msg);
```

Table 13.1.15.1-1. $trnLnm

$trnLnm translates a logical name. attr controls the search for the logical name, as documented in the "VAX/VMS System Services Reference Manual". tabName is the name of the table to search and logNam is the logical name to look up. If an error occurs, $trnLnm returns false and msg is set to indicate the error. Otherwise, $trnLnm returns true and name is the translation for the logical name.

Example of use:

```
$trnLnm('0L,"LNM$JOB","SYS$LOGIN",homeDir,msg)
```

[UCR's 86-579, 87-174]

13.1.15.2. CMS-Specific Procedures

13.1.15.2.1. $lookupSynonym

```
STRING
PROCEDURE   $lookupSynonym
                        (STRING synonymFile,symbol;
                         PRODUCES OPTIONAL STRING msg);
```

Table 13.1.15.2.1-1. $lookupSynonym

$lookupSynonym looks up a symbol in a CMS synonym file. synonymFile is the name of a synonym file to search, and symbol is the symbol to look up. The synonym file must be a valid synonym file, i.e., one which could be specified in the CMS "SYNONYM" command. If an error occurs, $lookupSynonym returns the null string and msg is set to indicate the error. Otherwise, $lookupSynonym returns the translation for the symbol. [UCR's 86-579, 87-174]

13.2. Runtime System

13.2.1. cvcs

The implementation of cvcs has been changed so that it no longer puts new text into string space (the area parameter to cvcs is therefore now ignored). This means a call to cvcs cannot cause a garbage collection.

13.2.2. UNIX Shell Variables in File Names

On UNIX, MAINSAIL has been enhanced to allow file names to begin with an environment variable. If the file name begins with the dollar character ("$"), MAINSAIL scans the file name for either the end of the file name string or for "/", whichever comes first. It looks up the resulting identifier in the envp array (passed to the C main program). If the identifier is found, the environment variable specification in the file name is replaced with value of the environment variable; otherwise, the file name remains unaltered.

File names beginning with dollar signs may appear in bootstrap files produced by CONF or in command files so that a MAINSAIL bootstrap can be made in such a way that it does not need to be reinstalled if moved to another directory.

[UCR's 84-630, 86-593, 87-206, 88-23]

13.2.3. MAINSAIL Revision

The MAINSAIL revision time is now displayed as part of the MAINSAIL herald when MAINSAIL comes up. A new MAINEX subcommand, "REVISION", also displays the current revision time. [UCR 88-258]

13.3. Compiler

13.3.1. Code Generator Errors

Code generator errors now prevent an object module from being produced. Previously, if the only errors during compilation occurred during the code generation phase, an object module would still be produced even though it would not run correctly. [UCR's 89-163, 90-213, 90-215]

13.3.2. VAX/VMS C FLI Labels

The VAX/VMS C FLI no longer prefixes an underscore ("_") to the default label for a C FLI procedure, since the VAX/VMS C compiler does not prefix an underscore to function names.

13.4. Utilities

13.4.1. MAINEX "DEFINE" Subcommand

A new MAINEX subcommand, "DEFINE", allows a MAINEX command to be defined as a macro. The syntax of the subcommand is:

```
DEFINE <identifier> <macroBody>
```

Whenever the identifier is encountered as the first word of a MAINEX command line, the macro body is substituted for it. For example:

```
DEFINE xxx compil
```

allows the MAINSAIL compiler to be invoked as "xxx". The MAINEX command line:

```
xxx foo.msl
```

invokes the compiler on the source file "foo.msl".

Since the macro body is just arbitrary text, it can include parameters in addition to the actual module name; e.g.:

```
DEFINE foo edit foo.msl
```

would edit the file "foo.msl" whenever "foo" was typed to the MAINEX "*" prompt. The macro body (with leading and trailing blank space removed) is substituted in place of the macro name in the original command line (separated from the rest of the command line by a single space if the command line contains text after the macro name).

For example, with the above definition, the command line "foo wy43" expands to "edit foo.msl wy43". Macro substitution is recursive, so that if you issue "DEFINE foo bar" and "DEFINE bar baz", then the command line "foo" ultimately expands to "baz". There is a recursion depth limit (its exact value is subject to change), so that an error message is issued if an infinite recursion occurs.

The motivation for the introduction of the "DEFINE" command is that there are some XIDAK module names that have a leading "$", and yet are documented for invocation by users: $DEBUG, $INTCOM, $MM, $OBJCOM, $STAMP, $STRTXT. Without a new mechanism, the user would have to type the full name, including the dollar sign, to the MAINEX prompt in order to invoke one of these modules. To allow users to continue to type the familiar names without dollar signs, the "DEFINE" subcommand is introduced. By default, XIDAK includes the subcommands:

```
DEFINE debug $debug
DEFINE intcom $intcom
DEFINE mm $mm
DEFINE objcom $objcom
DEFINE stamp $stamp
DEFINE strtxt $strtxt
```

in the system subcommand file during standard installation of MAINSAIL.

13.4.2. MAINEX "REVISION" Subcommand

A new MAINEX subcommand, "REVISION", displays the revision time of the current version of MAINSAIL. [UCR 88-258]

13.4.3. INTLIB/MODLIB DEFINE command

MODLIB and INTLIB each have a new "DEFINE" command:

```
DEFINE <identifier> <libraryFileName>
```

The defined identifier can later be used anywhere a library file name would be used as part of a MODLIB or INTLIB command, e.g.:

```
DEFINE foo /fs3/usr/fxi/fxi74200.40/src/src/fgc.ilb
ADD foo mod1 mod2 mod3
ADD foo mod4 mod5 mod6
DIR foo
```

The motivation for these commands should be clear from the above example.

13.4.4. Opening Intmod and Objmod Libraries

XIDAK utility commands that open intmod and objmod libraries (e.g., MAINEX's "OPENEXELIB" ("OPENLIBRARY"), "OPENINTLIB", and "OPENOBJLIB" commands) now move the library named by their argument to the front of the library list, if the library is already open. This affects the search order for intmods and objmods; a module of the same name earlier in the library list hides one later in the list.

13.4.5. MM Area Reference Commands

13.4.5.1. "ri" Command

The new MM command "ri f a1 ... an" ("reference info") writes information about references to areas with titles a1 ... an to file f. This command is a debugging aid which shows all pointer and string references into an area from outside the area. For example, if a program is disposing of an area at a certain point, and the user suspects that this is causing a "dangling" pointer or string bug, this command, given (e.g., from the debugger) just before the area is disposed, may help to determine what string or pointer is causing the problem.

If the ai (and possibly f) are omitted, MM prompts for them. If given on the same line as "ri", separate f and ai with one or more blanks and/or tabs.

The ai are area titles; each title is assumed to be a single "word" (no spaces) unless it starts with a double quote, in which case it extends to the next double quote (double quote pairs as in string constants are not recognized). If no ai are specified, MM prompts for them, in which case each response is taken as the area title (blanks and double quotes play no special role, so that arbitrary titles (except that embedded eols are not allowed) can be specified).

The file f is opened with the bits random!input!output!$unBuffered, so that "TTY" is not suitable. Use ">>f" instead of just "f" to append to the end of an existing file f (in which case the new info is preceded with an eop character if f is not initially empty).

MM first marks each area in a1 ... an, and then finds each pointer and string which references a marked area, but is not itself in a marked area. The identity of each such pointer or string is written to the file f, one line per reference. An example of some lines from such a file f is (the formats are subject to change):

```
Pointers and strings which reference the following areas
(from outside these areas):
compilArea

POINTER FIELD   KERSTRSPC.$AREAREC
POINTER LOCAL   $MM.REFINFO.-48
POINTER LOCAL   PASS1.GETSTA.$THISDB
POINTER LOCAL   PASS1.BEGINSTA.$THISDB
POINTER LOCAL   PASS1.ITERATIVESTA.Q
POINTER LOCAL   PASS1.GETPROCEDURE.MP
POINTER LOCAL   PASS1.GETPROCEDURE.-134
STRING  LOCAL   PASS1.GETPROCEDURE.PROCNAME = "INITIALPROC"
STRING  LOCAL   PASS1.GETDCLS.S = "SYNCHRONIZE"
POINTER LOCAL   PASS1.GETDCLS.DCLSYM
POINTER LOCAL   PASS1.GETDCLS.-90
POINTER FIELD   KERAREA.$NEXTAREA [pass1Area]
POINTER OWN     TREEIO.STRAREA [pass1Area]
POINTER OWN     PAKMOD.36 [pass1Area]
POINTER OWN     PAKMOD.0 [pass1Area]
STRING  FIELD   $PARSENODE.STR [pass1Area] = "44"
STRING  FIELD   $PARSENODE.KEY [pass1Area] = "S"
STRING  FIELD   $PARSENODE.STR [pass1Area] = "600 146"
STRING  FIELD   $PARSENODE.KEY [pass1Area] = "TEXTOPEN"
STRING  FIELD   $PARSENODE.KEY [pass1Area] = "S"
POINTER FIELD   $RECMANAGER.$AREAREC [keptCompilArea]
POINTER FIELD   $RECMANAGER.$AREAREC [keptCompilArea]
POINTER OWN     PAKMOD.36 [keptCompilArea]
POINTER IFIELD  INTMGR.LASTOPENMODULE [keptCompilArea]
POINTER IFIELD  INTMGR.VISIBLESYMTBLNODELIST [keptCompilArea]
POINTER OWN     INTMGR.STRAREA [keptCompilArea]
```

Each informational line starts out with "type kindOfVar" where type is POINTER or STRING, and kindOfVar is LOCAL, OWN, SHARED, FIELD, IFIELD, ARYELEM, or VECELEM. The exact format of the rest of the line depends on these first two fields.

For all but LOCAL, the title of the area which contains the referencing pointer or string is written in square brackets after the information identifying the reference (this is omitted if the title is null string).

If type is STRING, the entry ends with = "value", where value is the first 20 characters of text for the referenced string (if longer than 20 characters, "..." is appended to the end, e.g., "value...").

13.4.5.1.1. LOCAL

kindOfVar is LOCAL for a local variable (more precisely, a value in a stack frame). A local reference has the following format:

```
type LOCAL moduleName.procedureName.frameDspl [referencingCoroutine]
```

[referencingCoroutine] is omitted if it is the root coroutine "MAINSAIL" in which MAINSAIL starts execution.

An example of a line written for a LOCAL reference is:

```
POINTER LOCAL    $MM.REFINFO.-48
```

This means that one of the areas was referenced by a local pointer at displacement -48 in a stack frame for the procedure REFINFO in the module $MM.

The "local" variable $THISDB refers to the data section pointer that is stored in every frame, for example:

```
POINTER LOCAL    $MAINEX.$MAINSAILEXEC.$THISDB
```

In some rather obscure cases it is possible for <unknown> to be written for moduleName. If the referencing module's control section was not in memory when the reference was written, then the procedure name could not be found, and in its place is written the displacement in the control section to which control will return to the referencing procedure. If the displacement is not even known, <unknown> is written for procedureName. Use the "fr" command (described below) to translate frameDspl to the local variable name; e.g., the first example line above might be translated to:

```
POINTER LOCAL    $MM.REFINFO.AREALIST
```

13.4.5.1.2. OWN

kindOfVar is OWN for an own variable (more precisely, a value in a data section (a normal data section, not a shared data section) that is not an interface field). An own reference has the following format:

```
type OWN moduleName.dataSecDspl [referencingArea]
```

An example of a line written for a OWN reference is:

```
POINTER OWN      PAKMOD.36 [pass1Area]
```

This means that one of the areas is referenced by an own pointer at displacement 36 in a data section for the module PAKMOD, and the referencing data section is in an area with title "pass1Area".

Use the "fr" command described below to translate dataSecDspl to the own variable name; e.g., the example line above might be translated to:

```
POINTER OWN      PAKMOD.THEREC [pass1Area]
```

13.4.5.1.3. SHARED

kindOfVar is SHARED for a shared variable (i.e., a value in a shared data section). A shared reference has the following format:

```
type SHARED moduleName.sharedDataSecDspl [referencingArea]
```

An example of a line written for a SHARED reference is:

```
POINTER SHARED  FOO.20 [pass1Area]
```

This means that one of the areas is referenced by a shared pointer at displacement 20 in the shared data section for the module FOO, and the referencing shared data section is in an area with title "pass1Area".

Use the "fr" command described below to translate sharedDataSecDspl to the shared variable name; e.g., the example line above might be translated to:

```
POINTER SHARED  FOO.XRECNEXT [pass1Area]
```

13.4.5.1.4. FIELD

kindOfVar is FIELD for a field of a record or descriptor. A field reference has the following format:

```
type FIELD className.fieldName [referencingArea]
```

An example of a line written for a FIELD reference is:

```
STRING  FIELD   $PARSENODE.KEY [pass1Area] = "$RAISEFRAME"
```

This means that one of the areas is referenced by field KEY of a record of class $PARSENODE, and the referencing record is in an area with title "pass1Area". The referenced string is "$RAISEFRAME".

13.4.5.1.5. IFIELD

kindOfVar is IFIELD for an interface field of a data section. An IFIELD reference has the following format:

```
type IFIELD moduleName.fieldName [referencingArea]
```

An example of a line written for an IFIELD reference is:

```
POINTER IFIELD  INTMGR.FIRSTOPENMODULE [keptCompilArea]
```

This means that one of the areas is referenced by the field FIRSTOPENMODULE of a data section for the module INTMGR, and the referencing data section is in an area with title "keptCompileArea".

13.4.5.1.6. ARYELEM

kindOfVar is ARYELEM for an element of an array. An ARYELEM reference has the following format:

```
type ARYELEM arrayName[i1{,i2{,i3}}] [referencingArea]
```

An example of a line written for an ARYELEM reference is:

```
STRING  ARYELEM MODULENAMEMAP[1] [compilArea] = "$SYS"
```

This means that one of the areas is referenced by MODULENAMEMAP[1], and the referencing data section is in an area with title "compileArea". The referenced string is "$SYS".

13.4.5.1.7. VECELEM

kindOfVar is VECELEM for an element of a vector (an internal data structure used by the runtime system). A VECELEM reference has the following format:

```
type VECELEM [index] [referencingArea]
```

An example of a line written for a VECELEM reference is:

```
POINTER VECELEM [3] [compilArea]
```

This means that one of the areas is referenced by a vector element with index 3, and the referencing vector is in an area with title "compileArea". Note: vectors do not have names.

13.4.5.2. "fr" Command

The command "fr f g" ("fix references") copies the file f (created by "ri f...") to a new file g, except that an attempt is made to replace local and own displacements with the corresponding variable names. The file f may consist of any number of pages of output created by separate "ri" commands (which specified ">>f", i.e., append mode).

This translation is not carried out automatically by the "ri" command since the user may not want to disturb memory in the midst of program execution to the extent that "fr" does, and the supporting intmods (see below) may not be available in the execution context.

To replace a local variable displacement for a procedure in some module FOO, or an own variable displacement in FOO, FOO must have been compiled with the "DEBUG" option and the resulting intmod for module FOO must be accessible. "fr" complains when it cannot find an intmod or when the intmod has insufficient information, but in such cases it simply copies the original line from f to g.

There are other situations when a local or own displacement does not correspond to a variable name, in which case the "fr" command gives a message such as:

```
COMPIL.INITIALPROC.-16: no id for it
```

The original line is copied in this case also. A common case is a local variable inherited from an inline procedure of which the body was expanded in the calling procedure: the names for such inherited locals are not available from the intmod. Also, various own pointers are created by the compiler and do not have variable names.

13.4.6. New DVIEW Commands

New DVIEW commands have been added:

```
D           Reverse direction
I           Set value type to integer (default)
L           Set value type to long integer
P           Prompt <=> no prompt
```

After the "D" command has been given, the direction moved by the <eol> command changes from forward to backward or vice versa.

The "I" command causes DVIEW to examine the file one integer at a time; the "L" command, one long integer at a time.

The "P" command toggles the "DVIEW>" prompt.

Some improvements have also been made to DVIEW's output format.

13.4.7. MEM Buffer Size

The device module MEM has been generalized to handle different buffer sizes. Its argument is the buffer size (in number of characters); i.e., a file name can now be specified as "mem(2048)>foo.msl" to make MEM's memory buffers be 2048 characters. The default buffer size is 4096. [UCR 88-117]

13.5. MAINEDIT

13.5.1. Recovery of Strings at Command Line Prompts

Typing <ECM> ("enter command mode") to any prompt on the MAINEDIT command line pops the top element of the line delete buffer and inserts it at the current place on the message row. If the line buffer is empty, <ECM> beeps and aborts the prompt. Several <ECM>'s to the same prompt keep popping more text onto the end of the text so far. You can type your own text before, in between, and after any of the <ECM>-generated text.

For example, suppose you have a bunch of files you want to look at with the editor. Get all the file names in a buffer, one per line. To look at the next file, delete the line with its name, and issue a ".F" command; at the prompt,

type <ECM>, which inserts the deleted line onto the message row. Then type <eol> as usual. This works very nicely as part of macros.

Note: this mechanism works for any command line prompt, even, e.g., in MAINDEBUG when invoked from MAINEDIT. It does mean that if you have been using <ECM> to abort a prompt, you will have to change to use <ABORT> for this purpose.

13.5.2. New MAINED Search String Commands

New MAINED commands have been provided for manipulating the search strings used by the "T" command. These commands allow you to insert the current search strings into a buffer and to set the current search strings from text in a buffer that has been stored in one of the delete stacks.

13.5.2.1. "RS" (Recall SearchStrings)

The "RS" command inserts the current search strings above the current line, one per line. The search strings are uppercase.

Example: if you have just issued the command "Tabc<eol>" to search for "abc", then "RS" inserts the string "ABC" on a new line above the current line.

Example: if you have just issued the command "QTabc<eol>def<eol>ghi<eol><eol>", then the "RS" command inserts three lines above the current line and sets the first to "ABC", the second to "DEF", and the third to "GHI", since these are the current three search strings.

13.5.2.2. "+R{C,W,L}" (Recall {Characters, Words, Lines} into Search Strings)

These commands are like the corresponding commands without the "+", except instead of inserting the text into the buffer, they use it to set the current search strings. The various modifiers that can be used with the standard "R{C,W,L}" command, such as ".", "Q", and a count, also apply here.

Example: if you have just issued "DL" on a line consisting of "foobar", then "+RL" sets the current search strings to the single search string "FOOBAR".

Example: if you issue "3DL" to delete three lines, say:

```
this is the first line
followed by the second line
then the third
```

then "+RL" results in the three search strings "THIS IS THE FIRST LINE", "FOLLOWED BY THE SECOND LINE", and "THEN THE THIRD".

Example: "DW" which deletes, say, "THEN", followed by "+RW", makes "THEN" the current search string.

Example: Suppose you have two buffers A and B on the screen. A is a buffer of search strings, one per line, that you want to find in buffer B. Initially, put the cursor at the start of the first search string in A, then use ".B" to move to buffer B (anywhere in it). The following macro can then be used to find the first occurrence in B of the next word in the list:

```
.B<eol>.DW+RW<eol>.B<eol>1G.T<eol>
```

- ".B<eol>" switches from buffer B to buffer A
- ".DW" stores the next search word in the word delete buffer
- "+RW" pops that word into the search string
- <eol> moves the cursor to the next search word in buffer A
- ".B<eol>" switches from buffer A to buffer B
- "1G" goes to the start of buffer B
- ".T<eol>" searches for the current search string(s), which is the just set up by +RW

Only the final screen is seen after the entire macro has been executed, so the effect is for the cursor to move to the first occurrence in B of the next search word. Additional occurrences of the same word can be found using ".T<eol>".

13.5.3. Buffer-Specific "INITIALIZE" Macros

If a named macro is given a name of the form "INITIALIZE:...", where "..." represents a pattern that may contain the wildcard character "*", then the macro is executed whenever a buffer is created with a name that matches the pattern. The "*" wildcard character matches any sequence of zero or more characters; any other character in the pattern represents itself. For example, a macro named "INITIALIZE:*.MSL" would be executed upon starting to edit any file with a name ending in ".msl". [UCR's 85-39, 90-32]

13.5.4. Multiple Screens in a Single Edit Session

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

MAINEDIT now supports multiple screens during an edit session. The motivation for these changes was to allow users who have applications that use display modules to debug those applications with the display debugger. The application program can run on one screen and the debug session on another.

In the editor, when a file or buffer name is followed by a comma, the editor prompts for the front and back end names to use for the buffer, e.g., ".bfoo,". Now, if you specify an additional comma, e.g., ".bfoo,," or ".ffoo,,", the editor prompts for display information in addition to the front and back end names. The display information includes the name of a display module, the name of a terminal port, and a baud rate. The new buffer is created using the new display.

Whenever you do see the terminal port prompt, just type <eol> to the prompt unless you want to run on another display. If you do want to run on another display, type the display file name (e.g., something like "/dev/ttya" on UNIX).

In the debugger, the 'E' command now has the forms "{-}{.}E". The "." modifier means prompt for display information before running the module and then run the module on the specified display. The user interacts with the executed module on its display (and keyboard) and with the debugger on the display from which the debugger was invoked (the debug display). The debugger can be running either in line or display mode. The "-" modifier means do not put breakpoints in before running the module. This gives the user control over whether or not breakpoints are hit when a module is executed. Implementing these new forms of the "E" command has eliminated the need for the $lineOrientedDebug bit.

The display module interface has changed slightly. The only change that affects user programs is that initializeTerminal used to take an optional long integer baudrate; it now takes an optional string terminalPortName followed by the optional long integer baudrate.

13.5.5. Display Module and Baud Rate Prompts

MAINEDIT has been modified so that the default behavior is not to prompt for the terminal port and baud rate (after prompting for a display module name). In order to get these prompts, you must now type comma after the display module name (it is OK to have a comma after a display module name in the "eparms" file). The "eparms" keyword "DONOTPROMPTFORBAUDRATE" is no longer used, but the editor does not complain if it finds the keyword in the file.

A new "eparms" keyword "PROMPTFORDISPLAYMODULE" has been added. If specified in the "eparms" file, MAINEDIT always prompts for a display module name (but offers a default if there is one); if the keyword is not specified, it prompts for a display module name only if "," is specified after the initial file name. [UCR 86-399]

13.6. MAINPM

13.6.1. "TRACECHUNKS" Command

A new MAINPM command, "TRACECHUNKS {f}", traces the allocation history of each chunk (record, array, or data section) allocated in a monitored area. The resulting printout is an aid in determining the origin of chunks that are garbage collected or still in existence at program termination.

f is the name of the output "trace file" into which the trace data are written; if omitted, MAINPM chooses a unique name. The data in the trace file are not human-readable; it is only for analysis by MAINPM. By default, this analysis is done upon return to MAINPM, in which case the trace file is deleted if f was not explicitly specified. The "NOTRACEREPORT" command suppresses this report so that it can be done in a later run of MAINPM. Since trace data are written at some sensitive points during memory management, low-level runtime routines are used to write f on the assumption that f is a normal disk file; therefore, a device prefix should not be specified as part of f's file name.

To use MAINPM just to analyze a trace file created during an earlier run, give MAINPM the "TRACECHUNKS f" command, where f is the name of the trace file created earlier, then give it the "LIST g" command where g is the name of the output file. The trace file is written in PDF ("portable data format") so that it can be analyzed on a different kind of computer from that on which the run occurred. This may be necessary since the trace file may become quite large, in which case creation of the summary report uses a lot of memory, perhaps more than is available on the traced machine.

Data are written to the trace file every time a traced chunk is allocated, disposed, garbage collected, or reclaimed by $clearArea or $disposeArea. Data are also written for certain memory management events such as the movement of a traced chunk or chunk page in a traced area. The potentially large size of the trace file may prevent the use of the "TRACECHUNKS" command in some cases due to limitations on available disk space. Of course, the exact size for a given program execution depends on the number of traced events that occur.

An example of part of the report generated by MAINPM from the trace file is:

```
module{.proc}.offset                                          $clear/
{file filePos}                                                $dispose    still
{source text}                  new's    dispose's  collects   Area      allocated
LPIOPR.GENERATE.7984           67073          0       3936          0      63137
/usr/clark/xp.new/lpiopr.ms 16351
pi := new(layoutPin);

LNIOPR.GENERATE.8104           36081          0       2120          0      33961
/usr/clark/xp.new/lniopr.ms 16161
ni := new(layoutNode);

SPKOPR.GENERATE.13146          16323          0        961          0      15362
/usr/clark/xp.new/defopr.hdr 2046
newRec := new(defaultClass);

CFGIO.STRUCTUREIN.4652         11407        133      10353          0        921
/usr/clark/xp.new/cfgio.ms 22604
RETURN($structureRead(ctlFileAccess.ctlDataFile,

SEM0.OBJECTREAD.1514            4986        754          0       4210         22
/usr/clark/xp.new/sem0.ms 7623
RETURN($structureRead(filePtr,objAddr,objSize,

PCKOPR.GENERATE.4604            3267          0        193          0       3074
/usr/clark/xp.new/pckopr.ms 13355
pck := new(pckClass);

PCKOPR.GENERATE.4722            3266          0        193          0       3073
/usr/clark/xp.new/pckopr.ms 13448
new(pck.subUnit,0,pck.subUnitCnt);

NODLIB.GETNEXTTOKEN.6954        3177          0          0       3177          0
/usr/clark/xp.new/nodlib.ms 30373
token := new(tokenClass,processNodeLabelArea);
```

The entries are separated by a blank line. module.proc.offset shows where the allocations occurred. This point could have directly caused the allocation of a chunk, such as with new, $createRecord, $newRecords, or $structureRead, or indirectly, such as with a call to an unmonitored module that caused chunks to be allocated.

file and filePos indicate the source that corresponds to the statement that caused the allocation. source text is the actual text for the statement (from the start of the statement to the end of the first line of the statement; more of the statement could be on subsequent lines).

proc, file, filePos, and source text are collectively referred to as the trace source information. An intmod for the module must be available in order for the trace source information to be computable; source text also requires the original source file. This information can be suppressed in the trace report by specifying the "NOTRACESRC" subcommand to MAINEX, e.g., if you know that the intmods or source files are not available. The "TRACESRC"

command undoes the "NOTRACESRC" command (if you change your mind). IF "NOTRACESRC" is not in effect, MAINPM leaves out any information it cannot determine; e.g., if the intmod cannot be found, all of the trace source information is omitted; if the source file cannot be open, the source text is omitted.

Only chunks allocated in monitored areas are traced. Use the "{NO}MONITORAREA" commands to indicate which areas are to be monitored. By default, all areas are monitored.

Only monitored modules are credited with chunk allocations. Use the "{NO}MONITORMODULE" and "{NO}MONITORLIB" commands to specify which modules are to be monitored. Whenever a chunk is allocated in a monitored area, the procedure call stack is traversed to find the first monitored caller, and the chunk allocation is credited to that caller (if there is no monitored caller, the chunk allocation is not traced). By default, all modules are monitored. This default is not very useful to a typical user, since it simply shows places in the MAINSAIL runtime modules that are ultimately responsible for chunk allocations. At the minimum, a typical user should issue a "NOMONITORLIB s" command, where s is the name of the MAINSAIL runtime library file, or if all the modules to be monitored are in a single user library, issue a "MONITORLIB s" command where s is the name of that library file. Be careful to make the file name s exactly match the name of the library file as it is known to MAINSAIL during execution. The utility module OPENF can be used during an application to see the name of all open files, including open libraries.

The first line of output in the example shows that 67073 chunks were allocated due to a call at offset 7984 from the start of module LPIOPR, in procedure GENERATE. The source for the statement that allocated the chunks is in the file "/usr/clark/xp.new/lpiopr.ms" at position 16351. The text for the statement is "pi := new(layoutPin);". The MAINED editor front end can be used to view the source by opening the indicated file and using the command "V16352" (use 1 greater than the indicated value since MAINED uses 1-origin positions, whereas MAINPM reports 0-origin positions). Alternately, if LPIOPR is compiled debuggable, the source code at this location can be examined by using the MAINSAIL debugger as follows:

```
Debugger
Command     Description
m LPIOPR    set the current module context to be LPIOPR
o 7984      put the cursor at the source code corresponding to the
            indicated code displacement
```

The above debugger "o" command may put the cursor on the statement after the one that caused the allocation since the offset shown is actually the offset referenced by the return address to the procedure call that caused the chunks to be allocated, rather than to the call instruction itself. However, it is usually clear from the source which statement is responsible for the allocation. In order to use the debugger this way to view the source at the indicated offsets, the modules must have been compiled debuggable. This approach is useful if the "NOTRACESRC" command was in effect, or the trace source information could not be obtained when the report was generated.

The "new's" column indicates how many chunks were allocated (directly or indirectly) due to a call at the location given by the first two columns. The output is sorted by ascending order of this column.

The "dispose's" column indicates how many of the chunks were explicitly disposed.

The "collects" column indicates how many of the chunks were reclaimed by the garbage collector. When "TRACECHUNKS" is in effect, MAINPM triggers a garbage collection (if $collectLock = 0) immediately upon

return from the executed program so that this column accurately reflects how many chunks became "garbage" through the end of program execution.

The "$clear/$disposeArea" column indicates how many of the chunks were deallocated due to their area being cleared or disposed.

The "still allocated" column indicates how many of the chunks were still allocated upon return to MAINPM (after the garbage collection forced by MAINPM). This column is just equal to:

```
new's - dispose's - collects - $clear/$disposeArea
```

13.7. STREAMS

13.7.1. Bound Remote Modules

A new bit, $bindRemoteModules, is valid in the ctrlBits argument to $newRemoteModule. When this bit is used, calls to remote procedures using the remote module call the bound instance of the remote module rather than an unbound instance. [UCR 89-425]

13.7.2. New Instance of $waitForDescendants

```
LONG INTEGER
PROCEDURE   $waitForDescendants
                        (POINTER($coroutine)
                            ARRAY(1 TO *) children;
                         PRODUCES POINTER($coroutine)
                            childThatDied;
                         OPTIONAL LONG INTEGER timeOut);
```

Table 13.7.2-1. $waitForDescendants (New Instance Only)

A new instance of $waitForDescendants has been added. This instance of the call returns when any one of the descendant coroutines in the children array has died; the deceased coroutine is produced in childThatDied. The other forms wait either for all coroutines in array to die or for a single coroutine to die. [UCR 89-428]

13.7.3. $canonicalUserID

```
STRING
PROCEDURE    $canonicalUserID
                              (STRING userID,
                                   hostName);
```

Table 13.7.3-1. $canonicalUserID

$canonicalUserID takes a string userID (as returned from the $userID system procedure) and a string hostName (as returned from the $myHostName STREAMS system procedure) and returns a user ID string unique at the current site (provided an appropriate entry has been set up in the XIDAK service protocol table). This unique user ID is not guaranteed to be in a human-readable format.

The user ID is mapped by an entry in the XIDAK service protocol table. The service protocol table entry format is:

```
USERID canonicalName userID1/hostName1 ... userIdN/hostNameN
```

$canonicalUserID returns canonicalName if userID equals userIDi and hostName equals hostNamei. If there is no entry for the userID/hostName pair in the service protocol table, $canonicalUserID returns userID.

13.8. MAINKERMIT

13.8.1. "SET RETRIES n"

A new "SET" variable, "RETRIES", allows the user to set the number of retries for a bad packet before a transfer times out. The syntax is: "SET RETRIES n", where n is the new number of retries.

14. Known Problems

14.1. $descendantKilledExcpt and the MAINSAIL STREAMS Scheduler

At present, it is especially important to handle informational exceptions carefully (as described in Section 11.2) when running in conjunction with the MAINSAIL STREAMS Scheduler. In particular, a handler handling $descendantKilledExcpt raised because a scheduled coroutine is being killed should not attempt any action (such as file I/O) that might invoke the Scheduler. This is because the Scheduler is in an unusual state during $descendantKilledExcpt. This problem may be corrected in some future release of MAINSAIL.

14.2. SOCPRO and PTYPRO on IBM's AIX on IBM RISC System/6000

Because of problems with process termination on AIX, the STREAMS facilities SOCPRO and PTYPRO do not currently work on IBM's AIX on IBM RISC System/6000. IBM has been informed of the problem.

14.3. PC Monitoring Status

14.3.1. AIX

On AIX on both the IBM System/370 Extended Architecture and the IBM RISC System/6000, PC monitoring is supported only on recent releases of the operating system. On the IBM System/370 Extended Architecture, the supported releases are release 1.2.600 and later; on the IBM RISC System/6000, release 03.01.0001.0003 and later.

One way to find out version status and upgrade information on AIX is to run the command:

```
lslpp -h
```

which generates a table with the columns option name, state, event, date, release number, and user name. Look for a state of ACTIVE and an event of COMMIT to find the most recent upgrade for that option. For that option there will be a release number that looks something like 03.01.0001.0003. The 03.01 corresponds to the the AIX version 3.1 that is displayed when you log in. The 0001 and the 0003 correspond to upgrades.

14.3.2. PRISM

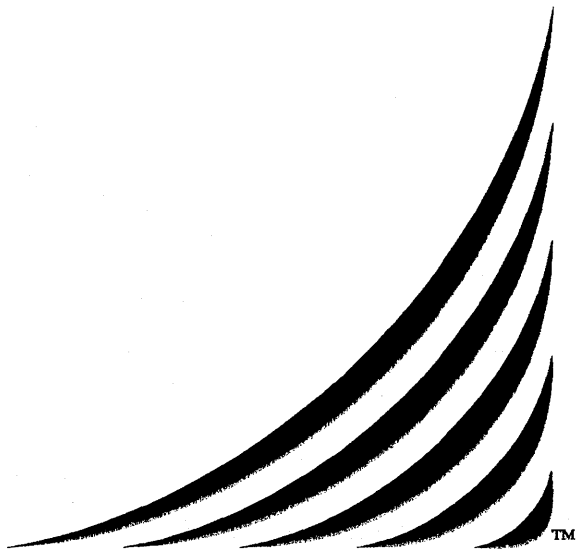
Due to a bug in the operating system, PC monitoring does not work on the PRISM. Apollo has been informed of the problem. Currently, attempting to use PC monitoring on this system has undefined effects and may crash MAINSAIL.

14.3.3. IBM's VM/XA SP CMS on IBM System/370

PC monitoring is not yet implemented on IBM's VM/XA SP CMS on IBM System/370. Currently, attempting to use PC monitoring on this system has undefined effects and may crash MAINSAIL.

14.3.4. Intergraph's System V UNIX on Interpro 32C and SCO's UNIX on HP Vectra with Intel 80386

On Intergraph's System V UNIX on Interpro 32C and SCO's UNIX on HP Vectra with Intel 80386, it is not possible to implement PC monitoring. Attempting to use PC monitoring on these systems issues an error message.



MAINSAIL® Release Notes

Version 11.30 Release

11 July 1991

15. Introduction

This release note documents the changes made between Versions 11.28 and 11.30 of MAINSAIL. If you have received the Version 12.10 manual set, you may also wish to consult the document entitled "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27", which describes how the MAINSAIL software described in the Version 12.10 documentation differs from Versions 11.27 and 11.28 (the differences described for 11.28 apply to 11.30 as well).

It was necessary to change some of the runtime system's internal interfaces in Version 11.30. This means that 11.30 object modules cannot be run under previous Version 11 releases, and that previous Version 11 object modules cannot be run under Version 11.30; you must rebuild all intmods and recompile all object modules.

16. Recent and Proposed Changes

16.1. $formParagraph

$formParagraph no longer puts eol at the end of the last line of a paragraph. Calls in existing code that depend on an eol at the end of $formParagraph should therefore be changed to:

```
$formParagraph(...) & eol
```

17. Clarification of Documentation

17.1. Making a Bootstrap on SPARC SunOS

The instructions in the "UNIX MAINSAIL User's Guide" for assembling bootstraps on the SPARC are wrong. The correct instructions are:

```
   There are two different "as" commands, one for SunOS
   versions before 4.0 and one for 4.0 and after.  For
   versions before 4.0:

% as -P -DoldRegNames -o mainsa.o mainsa.s<eol>

   For 4.0 and after:

% as -P -o mainsa.o mainsa.s<eol>
```

17.2. pageDispose

If numPages is not specified to pageDispose, or is less than zero, it defaults to one page.

17.3. Initializing Time Zone Parameters

If there is a problem in MAINSAIL's date and time procedures with conversion from the local time zone to GMT or vice versa, the first thing to check is whether the MAINEX time zone subcommands have been set properly. You can check whether they have been set at all by examining the variable $timeSubcommandsSet; e.g., enter the debugger and issue the command:

```
v $timeSubcommandsSet
```

If $timeSubcommandsSet is FALSE, the MAINEX time zone subcommands (DEFINETIMEZONE, DSTSTARTRULE/DSTENDRULE, DSTNAME, GMTOFFSET, STDNAME) were never issued. These subcommands should be given the correct arguments for your site (as described in the "MAINSAIL Utilities User's Guide"), or conversion to and from GMT will not work correctly on many operating systems. The subcommands are usually set in a file "site.cmd" on the MAINSAIL directory; if this file exists, it is read automatically by MAINSAIL whenever MAINSAIL starts.

17.4. The SUN3 Display Module and CommandTool

The SUN3 display module must be run in a ShellTool window, not a CommandTool window. Sun may enhance CommandTool to support display functions in the future, but currently CommandTool does not.

17.5. MAINED [+|-]..{n}[B|F] Commands

In MAINED, "+" or "-" may be prefixed to the "..B" and "..F" commands. The "..{n}B" and "..{n}F" commands edit the specified buffer or file, making the window 1/mth of the screen, where m is the number of windows (but no window is allowed to be smaller than n lines). The "+" and "-" versions create a new window, adding it to the bottom or top of the screen, respectively. Thus, if you are editing three windows, and want to edit a file "foo" in a fourth window at the top of the screen, and also to adjust each window to occupy a quarter of the screen, you would issue the command:

```
-..Ffoo<eol>
```

18. Bugs Fixed

18.1. Runtime System

18.1.1. $ioSize

On 11.28, if $ioSize was used in a module that didn't explicitly do a RESTOREFROM on PDFMOD's intmod, the use of $ioSize would result in the internal macro whichIntmod being undefined. This has been fixed. [UCR 89-417]

18.1.2. Duplicated Character from Terminal Input

The last character of the input buffer was incorrectly duplicated if the TTY input overflowed the buffer. The bug showed up when running MAINSAIL in batch and the batch file had an input line longer than the size of the TTY buffer.

18.1.3. Sun Installation Scripts

The Sun installation scripts did not understand more than one minor version number for the Sun operating system version number; i.e., they could understand "4.0" but not "4.0.3". They now handle the latter format correctly. [UCR 89-361]

The Sun installation got an undefined symbol error while installing "mainsab". [UCR 89-433]

18.2. Compiler

18.2.1. Bad Code for RETURN($exceptionStringArg1)

If a handler contains a Return Statement, then before the RETURN can be done, the handler must be terminated as if it had been fallen out of. There was a bug in which handlers were being terminated before the Return Statement's expression (for typed procedures) was evaluated. The handlers shouldn't have been terminated until after the RETURN expression was evaluated. This caused, e.g., "RETURN($exceptionStringArg1)" to return the wrong value. [UCR 89-382]

18.2.2. Incremental Recompilation of Modules Containing Calls to $createCoroutine

A compiler bug sometimes occurred when modules with calls to $createCoroutine were incrementally recompiled. The bug would cause the compiler to get confused when reading in the module's string constants, upon which it would either blow up with some sort of fatal error or, with no warning, corrupt the string constant space in the new object module.

18.2.3. MC68020/MC68881 Floating Point Comparison Bug

Bad code was sometimes generated on the MC68020/MC68881 for chained floating point comparisons or two floating point comparisons connected by "AND", e.g.:

```
9.L GEQ x AND x > 0.L
```

[UCR's 89-401, 89-454]

18.2.4. Intel 30386/387 Floating Point Comparison Bug

On the Intel 80386/387, floating point comparisons that should have been equivalent sometimes had different results, depending on whether or not the operands being compared happened to have been stored in memory and then reloaded into registers between the time that their values were initially calculated and the time they were compared.

The problem is that the registers on the 387 (Intel's floating point chip) have more bits of precision than single or double precision values in memory. When values are stored in memory, they are rounded to whatever precision the memory location has. When they are reloaded, this rounding is, of course, preserved. Usually the rounding has no effect on subsequent comparisons, but if two unrounded numbers are unequal but so close in value that their rounded values are the same, the results of comparing them would be different.

The fix was to make sure that operands of floating point comparisons are always rounded. To keep from generating a lot of extraneous rounding code (where the operand being rounded was already rounded), the code generator keeps track of whether or not each register is known to contain a rounded value. It then generates the extra rounding code for comparisons only for those operands that aren't guaranteed to be rounded already. [UCR 90-20]

18.2.5. SPARC Code Generator Bug in Floating Point Absolute Value

The SPARC code generator produced incorrect code for the floating point absolute value function (abs). [UCR 90-26]

18.2.6. MIPS R2000 Overflow Detection

Extra code must be generated on the MIPS R2000 to detect arithmetic overflow for multiplication and division. For multiplication, if one of the operands is a constant whose binary representation has exactly two 1-bits, the sense of the conditional branch around the call to the error routine was wrong, so that the error routine was being called when overflow did NOT occur. [UCR 89-400]

18.2.7. VAX-11 copy of NullCharadr or NullAddress

Bad code was generated for the VAX-11 for the system procedure "copy" if either of its first two arguments was the constant "NULLCHARADR" or "NULLADDRESS". Bad code could also have been generated for the address and charadr forms of the system procedures clear, load, and store. [UCR 89-244]

18.2.8. Bad Code for "0 DIV expr" and "0 MOD expr" on PRISM

Bad code was generated on the PRISM when the constant 0 was the first argument to the DIV or MOD operator. [UCR 89-474]

18.2.9. "countNumBrhsToEachIns: brh target = 0" on PRISM

The code generator issued the error message:

```
countNumBrhsToEachIns: brh target = 0
```

if "ACHECK" was set when code was generated for a (long) integer multiplication by a constant power of two.

18.2.10. Optimizer and Unitialized Floating Point

Because of the way code is rearranged by the optimizer, optimized programs sometimes access produces (long) real parameters even when the source text does not make it appear that it should do so. This resulted in errors on some systems which trap accesses to invalid floating point values. The optimizer has been changed to initialize produces (long) real parameters to zero. [UCR 89-385]

18.2.11. Optimizer and Long Arrays

When the optimizer replaced a long array common subexpression with an equivalent temporary, it didn't mark the temporary as also being a long array. On most machines, this made no difference in the code that was eventually generated when the long array was subscripted. However, on some machines, such as the MC68020/MC68881, the bug caused only the low-order 16 bits of the subscript to be used as an index instead of the full 32 bits. [UCR 89-433]

18.2.12. MIPS R2000 FLI to C

The MIPS R2000 FLI to C did not pass strings correctly. [UCR 89-409]

18.3. Utilities

18.3.1. $truncateFile on LIB File

LIB sometimes issued the spurious error:

```
$truncateFile: f not opened for random!input
```

18.4. MAINEDIT

18.4.1. MAINVI ":map" Command

":map" and related commands were taking only the first word after the macro key to be defined as the macro body. Real vi seems to take the entire rest of the line; MAINVI has been changed to work this way too. [UCR 89-394]

18.4.2. DOMAIN/IX FRAME Display Module

The FRAME display module on DOMAIN/IX has had a number of minor bug fixes. The most annoying bugs were that:

- When the cursor moved out of and then back into the FRAME window, the display module sent the letters "QN" to the application ("QN" is the MAINED command to refresh the window, but this is not appropriate when MAINED is not the application using FRAME). FRAME now uses the auto-refresh mode for its window, which avoids the problem.

- The RETURN key was not treated as end-of-line in CMDLOG from MAINVI, and ABORT/EXIT was not treated as a synonym of <esc> in MAINVI's insert mode. This can be fixed by the following entry in the ".mainvirc" file (see Section 19.3.1):

```
:mapcode! 148 <esc>
:mapcode! 150 <CTRL-M>
```

[UCR's 88-273, 88-297, 89-227, 89-393, 89-394, 89-445]

18.5. STREAMS

18.5.1. $openStream of "SOCPRO" File Did Not Fail If File Could Not Be Run

$openStream of "SOCPRO>Node:file" did not return FALSE (failure) if the specified program could not be run. This has been fixed. [UCR 89-330]

18.5.2. STREAMS Programs Invoked under SunView

STREAMS programs invoked under SunView often blew up because the version of the select system call under SunView did not examine the width of the bit array before clearing it (it always cleared 256 bits' worth, sometimes clobbering MAINSAIL data structures). MAINSAIL now always allocates at least 256 bits for the file descriptor bit array. [UCR 89-414]

18.5.3. More than One NETSTR Stream

A spurious error message was issued if an application opened more than one stream using "NETSTR".

18.5.4. Writing a Null String to a Memory Stream

Writing a null string to a memory stream caused a fatal error. It now does nothing. [UCR 89-420]

19. Enhancements

19.1. Language

19.1.1. Display Module Code dpyRefreshScreen

The display module interface procedure dpycRead can now return a new code, dpyRefreshScreen, which tells the caller that the screen needs to be repainted because the display module has discovered that the screen has been corrupted for some reason (e.g., a system message).

19.2. Compiler

19.2.1. "RPC C" Subcommand

The 11.30 compiler supports the "RPC C" subcommand. However, $compileTimeValue("RPC") was not upgraded as described in the Version 12.10 manuals. In Version 12, $compileTimeValue("RPC") returns one of "", "C", or "MAINSAIL"; under 11.30, $compileTimeValue("RPC") returns one of "" or "TRUE". [UCR 90-34]

19.2.2. Better Error Message When Combined Field Names of Class Are Too Long

When the name of a class and all its fields could not be fit into a string, the compiler previously generated the standard "string will be too long" error message, which was not very informative. It now gives the message:

```
Combined lengths of field names in class <c> are too long
```

[UCR 88-172]

19.2.3. Larger Procedure Bodies Allowed by PRISM Code Generator

The PRISM code generator now allows procedure bodies to be larger before issuing the "procedure too large" error message. [UCR 89-416]

19.2.4. Floating Point Overflow Trapped on Intel 387

On systems using the Intel 387 floating point coprocessor, MAINSAIL's initialization code has been changed to force the 387 to trap floating point overflow, division by zero, and invalid operation. Now when these errors occur, MAINSAIL raises an arithmetic exception. [UCR's 89-408, 90-9]

19.3. MAINEDIT

19.3.1. New MAINVI Commands

Arbitrary character codes can now be mapped using the ":mapcode" and ":mapcode!" commands, which are like the ":map" and ":map!" commands, except that the first word is not the keystroke itself that is to be mapped but the decimal character code of the keystroke to be mapped. In the case of ":mapcode", this is a code produced by dpycRead; for ":mapcode!", by unmappedDpycRead. You can use the program of Figure 19.3.3-1 to determine key codes produced by a display module; answer "Y" to the initial prompt for mapped (dpycRead) codes or "N" or unmapped (unmappedDpycRead) codes.

19.3.2. New MAINVI Initialization File

Since the new ":mapcode" and ":mapcode!" commands, along with other MAINVI-specific commands, would not be compatible with the real vi commands normally placed in a ".exrc" file, MAINVI now reads an additional command file in your home directory (if present). This file is called ".mainvirc". It is read immediately after ".exrc".

19.3.3. DOMAIN/IX FRAME Display Module Initialization

You can now create a file called ".framerc" in your home directory to specify the value to be passed to GPR_$SET_OBSCURED_OPT when the FRAME (or FFRAME) display module is initialized and the key codes not to be caught by the FRAME display module. If ".framerc" does not exist, the defaults are the values now used by the display module.

Each line of ".framerc" consists of a keyword followed by one or more arguments. The two keywords now allowed are "OBSCUREDOPT" and "DONOTCATCH". Case is ignored in keywords. The arguments allowed to these keywords are:

- OBSCUREDOPT: An integer corresponding to the value to be passed to GPR_$SET_OBSCURED_OPT, one of:

```
0       gpr_$ok_if_obs
1       gpr_$error_if_obs
2       gpr_$pop_if_obs
3       gpr_$block_if_obs
4       gpr_$input_ok_if_obs
```

- DONOTCATCH: One or more integers which are the codes of keys not to be caught by the display module. More than one DONOTCATCH line may appear in the file if there are more codes than is convenient to place on one line.

For example, a ".framerc" that specifies GPR_$POP_IF_OBS and that codes 1 and 147 (CTRL-A and SAVE/EDIT) are not to be caught would look like:

```
OBSCUREDOPT 2
DONOTCATCH 1 147
```

The reading of ".framerc" stops with the first blank line or with end-of-file.

The program of Figure 19.3.3-1 may be useful in determing the codes generated by various keys. For the purposes of ".framerc", use the unmapped (N) response. Exit the program by typing "q" or "Q".

```
BEGIN "keyCod"

REDEFINE $scanName = "dpyHdr"; SOURCEFILE "(system library)";

INITIAL PROCEDURE;
BEGIN
BOOLEAN mapped;
INTEGER ch;
STRING s;
POINTER(dpyCls) dpy;
mapped := confirm("Y for mapped key codes, N for unmapped");
write(logFile,"Display module: "); read(cmdFile,s); dpy := new(s);
dpy.initializeTerminal; dpy.clearScreen;
DOB ch := IF mapped THEN dpy.dpycRead EL dpy.unmappedDpycRead;
    dpy.clearScreen; dpy.setCursorOnScreen(0,0);
    dpy.overstrikeChars(cvs(ch)) END UNTIL ch = 'q' OR ch = 'Q';
dpy.deInitializeTerminal;
dispose(dpy);
END;

END "keyCod"
```

Figure 19.3.3-1. A Program to Determine Mapped/Unmapped Key Codes Generated by a Display Module

19.3.4. Buffer-Specific Initialization Macros

MAINEDIT now executes a named macro of the form form INITIALIZE:<bufferNamePattern> whenever setting up a buffer whose name matches the specified pattern. The buffer name pattern may include the same wildcards as used in a MAINEX "SEARCHPATH" subcommand. For example, the macro named "INITIALIZE:*.MSS" is executed every time a buffer name ending in ".MSS" is created.

19.4. STREAMS

19.4.1. New Form of $waitForDescendants

```
LONG INTEGER
PROCEDURE   $waitForDescendants
                        (POINTER($coroutine) ARRAY
                             (1 TO *) children;
                         PRODUCES POINTER($coroutine)
                             childThatDied;
                         OPTIONAL LONG INTEGER timeOut);
```

Figure 19.4.1-1. $waitForDescendants (New Generic Instance Only)

The new instance of $waitForDescendants differs from the other instances in that:

- The coroutine array parameter is not optional.
- The new instance returns as soon as ANY of the specified coroutines dies; the other array instance waits for ALL specified coroutines to die before returning.
- The new instance sets the parameter childThatDied to the array entry for the coroutine that died, and then sets that array entry to nullPointer. Since nullPointer array entries are ignored (this is true of the other array instance of $waitForDescendants as well), this makes it convenient to call the new form of $waitForDescendants in a loop:

```
new(ary,1,n);
        .
        .
        .
FOR i := 1 UPTO n DOB
    $waitForDescendants(ary,childThatDied);
    ... code to handle death of childThatDied ... END;
```

19.4.2. User-Definable Timeouts

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

Two new user-visible fields, $maxExec and $nto, have been added to $remoteModuleCls. These fields specify timeouts for the remote module.

$maxExec is the longest expected execution time for the remote procedure. If the remote procedure executes for longer than $maxExec, the client will time out and abort execution of the remote procedure. The default value for $maxExec is quite long, since it is difficult to predict how long a remote procedure may require in order to finish.

$nto is a short timeout used when setting up a remote procedure call. Programmers will not usually need to adjust $nto. However, where client-server communication is unexpectedly slow, it is possible that $nto may be unnecessarily exceeded and the remote procedure call aborted by a timeout. In this case, $nto should be increased to avoid the timeout.

To set the a timeout, set the appropriate field of $remoteModuleCls immediately after allocating a remote module; e.g., for $maxExec:

```
m := $newRemoteModule(...);
m.$maxExec := <new timeout value>;
```

You must set the $maxExec field before making any calls to the remote module; the effect of setting it after the first call is made is undefined.

19.4.3. SOCPRO and Searchpaths

SOCPRO now examines the "ENTER" and "SEARCHPATH" subcommands specified in the parent process to map the executable file name specified in the SOCPRO stream name. On UNIX, it also uses the parent process's PATH environment variable. Note that if a SOCPRO stream is opened through NETSTR, the parent process is the "gensrv" server process, and not the process opening the NETSTR stream. [UCR 89-78]

20. Known Problems

20.1. SPARCstation Operating System Version Numbers

MAINSAIL runs on both versions 3.x and 4.x of the SunOS operating system on SPARC-based workstations. There are some incompatibilities between these two versions of SunOS that make it more difficult to built a MAINSAIL bootstrap on networks that contain both versions of the operating system. If you have only one version of SunOS for the SPARC (either all 3.x or all 4.x) on your network, then you will not have any of the problems described below, and you do not need to read this section any further.

The incompatibilites are two-fold:

1. MAINSAIL bootstraps built on 3.x will run under 4.x, but not vice versa.

2. All pieces of a bootstrap must be compiled, assembled, and linked under the same version of the operating system; bootstraps that contain pieces built under different versions of the operating system will not work correctly.

The first problem means that if you have a MAINSAIL system that will be used from both 4.x and 3.x nodes, then it must be installed on a 3.x node. The second problem means that when you are building any piece of a MAINSAIL bootstrap or code that is linked with a MAINSAIL bootstrap, you must always run "as", "cc", or "ld" (or whatever the local equivalents of those programs may be called) on the same version of the operating system as on the node where the MAINSAIL system is installed. If you follow the above recommendation to install your MAINSAIL systems on 3.x nodes, then you must always issue any bootstrap-making commands on 3.x nodes as well. If you accidentally build a bootstrap with pieces from different versions, the symptoms are unpredicatble, but are often manifested as file I/O errors.

20.2. MAXMEMORYSIZE under AIX

In an AIX bootstrap configured to run in 24-bit mode, the system stack starts at the address 0x7fffff and grow towards low memory. Hence, the amount of memory that can be allocated with sbrk is limited to between 7M and 8M.

We recommend a value of no more than 6M for the CONF parameter, MAXMEMORYSIZE, for boots that run in 24-bit mode. This will prevent your process from growing into the system stack.

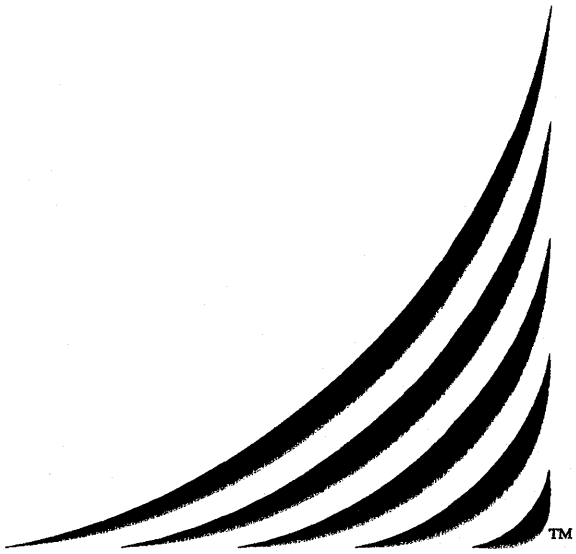
On AIX 0.1, to configure your executable boots to run in XA mode instead of 24-bit mode, issue the command:

```
chmodxa <executableBootFileName>
```

chmodxa is an AIX 0.1 command which toggles the execution mode of the specified executable file. When the command is issued, it prints out information about how the executable file is configured, i.e., 24-bit (370 mode) vs. 31-bit (XA mode).

On AIX 0.2, you must give a "cc" switch, "-Hxa", to specify 31-bit mode when a bootstrap is linked. Otherwise, the bootstrap will be built in 24-bit mode. For example:

```
cc -Hxa -o mainsa mainsa.s /usr/mainsail/11.30/m.o
```



MAINSAIL® Release Notes

Version 11.28 Release

11 July 1991

XIDAK

21. Introduction

This release note documents the changes made between Versions 11.27 and 11.28 of MAINSAIL. If you have received the Version 12.10 manual set, you may also wish to consult the document entitled "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27", which describes how the MAINSAIL software described in the Version 12.10 documentation differs from Versions 11.27 and 11.28.

It was necessary to change some of the runtime system's internal interfaces in Version 11.28. This means that 11.28 object modules cannot be run under previous Version 11 releases, and that previous Version 11 object modules cannot be run under Version 11.28; you must rebuild all intmods and recompile all object modules.

22. Recent and Proposed Changes

CMS/EDAC is no longer supported in 11.28.

23. Clarification of Documentation

23.1. Arrays Declared with Constant Bounds

If an array is declared with constant bounds, then its bounds must always have the values declared. The compiler may take advantage of the constant bound information, and arrays that do not match their compiletime declarations at runtime will not be correctly accessed and have undefined effects. In particular, it is the programmer's responsibility to ensure that if an array with variable bounds is assigned to (or passed as an argument corresponding to) an array with constant bounds, the bounds of the variable-bounded array match the declaration of the constant-bounded array. For example, although the compiler permits the following:

```
INTEGER ARRAY(1 TO 10) ary1;
INTEGER ARRAY(*) ary2;
...
new(ary2,0,20);
...
ary1 := ary2;
```

the use of ary1 after the Assignment Statement has undefined effects.

23.2. $initRand

The parameters to $initRand (newX and newX2) constitute a single 56-bit seed for the $rand generator (only the low-order 28 bits of each are examined). A way to use $initRand to set the seed depending on the current second is:

```
$initRand($date,$time)
```

23.3. MAINPM "LIST" Command

The MAINPM "LIST" command does not immediately close the listing file; the file is not closed until MAINPM exits.

23.4. Apollo's DOMAIN/IX on Apollo PRISM Considerations

MAINSAIL on Apollo's DOMAIN/IX on Apollo PRISM behaves like most other flavors of UNIX, but because the Apollo operating system must be compatible with Aegis, Apollo's previous operating system, some special considerations must be taken into account.

23.4.1. Operating System Environment

MAINSAIL on Apollo's DOMAIN/IX on Apollo PRISM must be run in a BSD environment. In particular, all directories on which MAINSAIL writes files must have protections that are consistent with a BSD environment. By default, a directory's protection is inherited from its parent directory, which may be inappropriate for BSD. Observed symptoms of an "inappropriate" protection are the "ld" and "cc" commands failing with the message "ld fatal: can't create output file", and occasional, seemingly random MAINSAIL error messages to the effect that a file cannot be created or opened for output.

A directory's protection may be made consistent with BSD by means of the "chacl" (change access control list) command with the "-B" option. Typically, this is necessary only for higher-level directories such as users' login directories. Any subdirectories created after their parent directories' protection have been corrected will also have the correct protection. Consult the UNIX documentation for further information.

23.4.2. Foreign Language Interface

Calls from MAINSAIL to procedures written in other languages, and from procedures written in other languages to MAINSAIL, are supported as long as the procedures meet the interface for C procedures without function prototypes, as described in the "Calling Conventions" chapter of Apollo's "Series 10000 Assembler Reference" manual and subsequently corrected in the "prasm Software Release Document, Software Release 10.0.p", dated 13 October 1988.

Calls between MAINSAIL and Pascal are not supported. Contact XIDAK if such support is desired.

24. Bugs Fixed

24.1. Runtime System

24.1.1. closeLibrary

A bug in closeLibrary eventually caused a nullPointer data access in $swapIn. The bug occurred in the following situation:

- A module (say, FOO) resided in a library (say, "foo.olb").
- A data section for FOO was allocated.
- FOO was swapped out of memory.
- FOO's data section was disposed.
- The library "foo.olb" was closed and then reopened.
- A data section for FOO was allocated.

24.1.2. Aegis Runtime System

$fileInfo set the $createDate, $createTime, $modifyDate, and $modifyTime fields in GMT even when local time was specified. [UCR 89-81]

24.2. Compiler

24.2.1. PRISM Code Generator

Bad code was sometimes generated in accessing elements of one-dimensional string arrays with constant lower bounds. [UCR 89-269]

24.2.2. VAX-11 Code Generator

Bad code was generated for the integer forms of copy and clear and the long integer form of clear if the length specified was not a constant. If the length was negative, no storage should have been affected, but instead a non-zero amount of memory was affected.

25. Enhancements

25.1. Language

25.1.1. $length

As documented in the "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27", the Version 12.10 procedure $length is available in Version 11.28.

```
INTEGER
PROCEDURE   $length     (BOOLEAN v);

INTEGER
PROCEDURE   $length     (INTEGER v);

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 25.1.1-1. $length (Generic)

$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>
```

25.1.2. $minInteger and $minLongInteger

As documented in the "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27", the Version 12.10 identifiers $minInteger and $minLongInteger are available in Version 11.28.

```
COMPILETIME
INTEGER
<macro>     $minInteger;

COMPILETIME
LONG INTEGER
<macro>     $minLongInteger;
```

Table 25.1.2-1. $minInteger and $minLongInteger

Analogous to $maxInteger and $maxLongInteger, $minInteger and $minLongInteger are the smallest integer and long integer, respectively, representable on the host processor. An arithmetic expression that attempts to create an integer value less than $minInteger or a long integer value less than $minLongInteger may lead to overflow, which has undefined effects.

25.2. Runtime System

25.2.1. New Memory Management Heuristics

As documented in the "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27", the Version 12.10 memory management heuristics (as governed by "COLLECTMEMORYPERCENT") are available in Version 11.28.

The new memory management heuristics frequently perform better than the old ones. [UCR's 87-309, 87-313, 87-314, 88-144, 88-217, 88-299] MAINSAIL no longer uses a parameter that tells it what percentage of CPU time is to be used in memory management; instead, MAINSAIL decides to do memory management if it predicts that n percent of memory would be reclaimed by doing a garbage collection (where n is the value of the new CONF parameter "COLLECTMEMORYPERCENT"; the old CONF parameter "ALLOWEDPERCENT" no longer exists). Thus a higher value for "COLLECTMEMORYPERCENT", such as 10%, means that MAINSAIL collects garbage less often since more garbage has to build up before a garbage collection occurs. A smaller value, such as 1%, means more garbage collections since only 1% of memory is allowed to be garbage. The heuristic depends on an estimate that MAINSAIL makes of how much of memory is garbage; this estimate changes as garbage collections occur. Use of "COLLECTMEMORYPERCENT" instead of the old "ALLOWEDPERCENT" leads to more reproducible behavior since garbage collections no longer depend on CPU time, which can be different from one run to the next even with identical inputs. Note that MAINSAIL might garbage collect regardless of the value of "COLLECTMEMORYPERCENT" when it cannot get more memory from the operating system.

Large values for "COLLECTMEMORYPERCENT" may lead to unnecessary paging if a process's virtual size is larger than the physical memory available. Values for "COLLECTMEMORYPERCENT" larger than about 10% have not been shown to be useful. A value of about 1% may be reasonable for a large batch job.

The percentage of CPU time spent in memory management overhead is still calculated, and MAINSAIL still exits if the overhead exceeds $overheadPercentExitValue.

On some operating systems, MAINSAIL now returns memory to the operating system in certain cases, thereby reducing its process size.

25.3. MAINEDIT

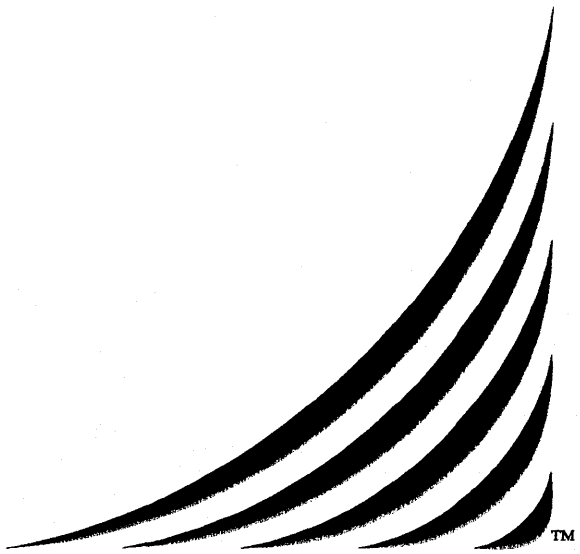
25.3.1. MAINVI

As documented in the "MAINSAIL Inverse Release Notes, Version 12.10 - Version 11.27", most of the bug fixes to MAINVI documented in the "Version 12.10 Release Notes" are available in Version 11.28.

25.4. Structure Blaster

25.4.1. $structureCopy

$structureCopy no longer makes duplicates of the string text of a structure; both old and new structures share the same string text.



MAINSAIL® Inverse Release Notes

Version 12.10 - Version 11.27

11 July 1991

XIDAK

25.5. Introduction

This "inverse release note" describes the status of new features in Version 12.10 of MAINSAIL. Some of these features are present in Version 11.27 and subsequent releases of Version 11, but some are absent. Manual sets current for Version 12.10 are now being shipped to MAINSAIL users, but Version 12 of MAINSAIL has not yet been released to customers. This note is provided so that the Version 12.10 manual set may be used with Version 11 releases of MAINSAIL.

Some minor enhancements made to Version 12.10 of MAINSAIL were omitted from this list; all such enhancements either do not affect user code or are present in Version 11.27 of MAINSAIL, so that the Version 12.10 documentation is accurate for Version 11.27 for these items.

25.6. Language

25.6.1. New Conditional Compilation Directives

The conditional compilation directives "$EFC", "$CASEC", "$BEGINC", "$DOC", "$DONEC", "$CONTINUEC", and "$FORC" are all present in Version 11.27 of MAINSAIL.

25.6.2. New Argument to "MESSAGE" Directive

The new optional second arguments to "MESSAGE" ("warning" or "error") are not available in Version 11.27 of MAINSAIL.

25.6.3. New Compiletime Pseudo-Procedures

The compiletime pseudo-procedures "$TYPEOF", "$CLASSOF", and "$ISCONSTANT" are all present in Version 11.27 of MAINSAIL.

25.6.4. Repeatable Macro Parameters, $numArgs, $arg, and $sArg

Repeatable macro parameters, $numArgs, $arg, and $sArg are all present in Version 11.27 of MAINSAIL.

25.6.5. New Arguments to $compileTimeValue

The $compileTimeValue arguments:

```
RESTOREFROM <file or module name>
SOURCEFILE
SOURCEFILE <file name>
```

are not available in Version 11.27 of MAINSAIL. The argument "RPC" is available in Version 11.27 of MAINSAIL.

25.6.6. New System Procedures and Macros

25.6.6.1. $atan2

$atan2 is not available in Version 11.27 of MAINSAIL.

25.6.6.2. $classDscrFor

$classDscrFor is not available in Version 11.27 of MAINSAIL.

25.6.6.3. $cot

$cot is not available in Version 11.27 of MAINSAIL.

25.6.6.4. $def

$def is available in Version 11.27 of MAINSAIL.

25.6.6.5. $disposeDataSecsInArea

$disposeDataSecsInArea is not available in Version 11.27 of MAINSAIL.

25.6.6.6. $length

$length is not available in Version 11.27 of MAINSAIL. However, it will be available in Version 11.28 and subsequent Version 11 releases of MAINSAIL.

25.6.6.7. $log2

$log2 is available in Version 11.27 of MAINSAIL.

25.6.6.8. $mainsailExec

$mainsailExec is available in Version 11.27 of MAINSAIL.

25.6.6.9. $minInteger and $minLongInteger

$minInteger and $minLongInteger are not available in Version 11.27 of MAINSAIL. However, they will be available in Version 11.28 and subsequent Version 11 releases of MAINSAIL.

25.6.6.10. Platform Macros

The platform macros $platformNameFull, $platformNameAbbreviation, and $platformNumber are not available in Version 11.27 of MAINSAIL; neither are the predefined platform identifiers (e.g., $aix, $alnt, ...).

25.6.7. System Procedures with New Parameters and Instances

25.6.7.1. $dateToStr and $dateAndTimeToStr

The $dateToStr and $dateAndTimeToStr parameter ctrlBits2 is not available in Version 11.27 of MAINSAIL.

25.6.7.2. displace

The pointer instance of displace is available in Version 11.27 of MAINSAIL.

25.7. Runtime System

25.7.1. New Memory Management Heuristics

The new memory management heuristics (as governed by "COLLECTMEMORYPERCENT") are not available in Version 11.27 of MAINSAIL. However, they will be available in Version 11.28 and subsequent Version 11 releases of MAINSAIL.

25.7.2. Date and Time Facilities

The restrictions that the number of hours be less than or equal to 24, minutes less than or equal to 60, and seconds less than or equal to 60 have not been removed for time differences passed to $strToTime in Version 11.27 of MAINSAIL.

The change to $strToDate causing it no longer to prompt:

```
Do you mean the year to be in the current century?
```

is available in Version 11.27 of MAINSAIL.

25.7.3. $disposeArea and $clearArea and Data Sections

The prohibition against calling $disposeArea and $clearArea on areas containing data sections has not been lifted in Version 11.27 of MAINSAIL.

25.7.4. Chunk Alignment

The alignment of chunks remains four times the size of a long integer in Version 11.27 of MAINSAIL (it is twice the size of a long integer in Version 12.10).

25.7.5. $ttyEofExcpt

The predefined exception $ttyEofExcpt is not available in Version 11.27 of MAINSAIL.

25.7.6. "DEFINETIMEZONE"

The MAINEX subcommand "DEFINETIMEZONE" is not available in Version 11.27 of MAINSAIL.

25.7.7. File Cache Procedure Parameters

The parameter attributes to $queryFileCacheParms and $setFileCacheParms is a long bits in both Version 11.27 and Version 12.10, not a bits as in previous Version 11 MAINSAIL releases.

25.7.8. VAX/VMS File Formats

The device prefix "BS(n)", where n is the record size, is available on VAX/VMS in Version 11.27 of MAINSAIL.

25.8. Utilities

25.8.1. CALLS

The command line argument to CALLS is not available in Version 11.27 of MAINSAIL.

25.8.2. CONF

25.8.2.1. "PLATFORM" Command

The CONF "PLATFORM" command is not available in Version 11.27 of MAINSAIL.

25.8.2.2. "COLLECTMEMORYPERCENT" vs. "ALLOWEDPERCENT"

In Version 11.27 and earlier Version 11 releases of MAINSAIL, the old CONF command "ALLOWEDPERCENT" is supported, but not the new command "COLLECTMEMORYPERCENT". In Version 11.28 and subsequent Version 11 releases, "COLLECTMEMORYPERCENT" will be available, but not "ALLOWEDPERCENT".

25.8.3. INTCOM

INTCOM is not available in Version 11.27 of MAINSAIL.

25.8.4. MAINEX

25.8.4.1. Long Subcommand Lines

The feature allowing long MAINEX subcommand lines to be continued by ending them with the backslash ("\") character is available in Version 11.27 of MAINSAIL.

25.8.4.2. "CSUBCOMMANDS <fn>"

The "CSUBCOMMANDS <fn>" subcommand is available in Version 11.27 of MAINSAIL.

25.8.5. MAINKERMIT

The MAINKERMIT command "PTEXT" is available in Version 11.27 of MAINSAIL.

25.8.6. MM

All MM features described in the Version 12.10 documentation are available in Version 11.27 of MAINSAIL.

25.8.7. OBJCOM

OBJCOM is not available in Version 11.27 of MAINSAIL.

25.9. Debugger

The "+Q" (unconditional exit) command is not available in Version 11.27 of MAINSAIL.

25.10. MAINEDIT

The display modules WY50, WY5043, WY43, WY75, and HP300H are all available in Version 11.27 of MAINSAIL.

25.10.1. MAINVI

Most bug fixes and enhancements made to MAINVI over the course of the past year are not available in Version 11.27 of MAINSAIL. However, they will be available in Version 11.28 and subsequent Version 11 releases of MAINSAIL.

25.11. MAINPM

Neither the "SPACE" nor the "PC" command is available in Version 11.27 of MAINSAIL.

25.12. Structure Blaster

25.12.1. $structureCompare

The procedure $structureCompare is available in Version 11.27 of MAINSAIL.

25.12.2. Structures Written Only from Specified Areas

The temporary feature allowing structures to be written only from areas marked with the $markedArea bit is not available in Version 11.27 of MAINSAIL.

25.13. Summary

The following is a summary of the availability of Version 12.10 features in Versions 11.27 and 11.28:

Feature	11.27	11.28
$EFC, $CASEC, $BEGINC, $DOC, $DONEC, $CONTINUEC, $FORC	YES	YES
MESSAGE second argument	NO	NO
$TYPEOF, $CLASSOF, $ISCONSTANT	YES	YES
Repeatable macro parameters, $numArgs, $arg, $sArg	YES	YES
$compileTimeValue arguments RESTOREFROM, SOURCEFILE	NO	NO
$compileTimeValue argument RPC	YES	YES
$atan2	NO	NO
$classDscrFor	NO	NO
$cot	NO	NO
$def	YES	YES
$disposeDataSecsInArea, $disposeArea and $clearArea when there are data sections in area	NO	NO
$length	NO	YES
$log2	YES	YES
$mainsailExec	YES	YES
$minInteger, $minLongInteger	NO	YES
Platform macros, CONF PLATFORM cmd	NO	NO
$dateToStr/$dateAndTimeToStr parameter ctrlBits2	NO	NO
Pointer version of displace	YES	YES
New memory management (COLLECTMEMORYPERCENT)	NO	YES
Time differences > 24 hr or > 60 min/sec	NO	NO
No more "current century" prompt from $strToDate	YES	YES
Chunk alignment 2 long integers	NO	NO
$ttyEofExcpt	NO	NO
DEFINETIMEZONE MAINEX subcommand	NO	NO
$queryFileCacheParms/ $setFileCacheParms attributes parameter long integer	YES	YES
BS(n) on VAX/VMS	YES	YES
CALLS command line argument	NO	NO
INTCOM, OBJCOM	NO	NO
MAINEX long lines split with \	YES	YES
CSUBCOMMANDS MAINEX subcommand	YES	YES
MAINKERMIT PTEXT command	YES	YES
MM features	YES	YES
MAINDEBUG +Q command	NO	NO
New MAINEDIT display modules	YES	YES
MAINVI fixes	NO	YES
MAINPM SPACE and PC commands	NO	NO
$structureCompare	YES	YES
Structures written only from marked areas	NO	NO

Index