MAINSAIL®

System-Specific User's Guides

MAINSAIL®

System-Specific User's Guides

24 March 1989

Table of Contents

Appendices

List of Examples

List of Figures

List of Tables

1. System-Specific User's Guides

This document contains descriptions of system-specific features of MAINSAIL on all operating systems on which MAINSAIL is available as of March, 1989.

Implementations exist for all the operating systems for which MAINSAIL is described in this document. The current support level of MAINSAIL on each supported platform is shown in Table 1-1; consult a current "XIDAK Product Catalog" for more information.

1.1. Conventions Used in This Document

1.1.1. User Interaction

Throughout the examples in this document, characters typed by the user are underlined. "<eol>" symbolizes the end-of-line key on a terminal keyboard; this key is marked "RETURN" or "ENTER" on most keyboards. In Example 1.1.1-1, "Prompt:" is written by the computer; the user types "response" and then presses the end-of-line key.

```
Prompt: response<eol>
```

Example 1.1.1-1. How User Input Is Distinguished

1.1.2. Syntax Descriptions

Specifications of syntax often contain descriptions enclosed in angle brackets ("<" and ">"). Such descriptions are not typed literally, but are replaced with instances of the things they describe. For example, a specification of the syntax of the address on an envelope might appear as in Example 1.1.2-1.

Optional elements in command or syntax descriptions are often enclosed in curly brackets ("{" and "}"). For example, a string of characters specified as "{A}B{C}" could have any one of the forms "B", "BC", "AB", and "ABC". Alternatives may be enclosed in square brackets (or curly brackets, if all alternatives are optional) and separated by vertical bars ("|"); "[A|B|C]" means "A", "B", or "C"; "{A|B}" means "A", "B", or nothing.

Platform Abbrev.	Platform Name	Support Level
aeg	Apollo's Aegis on Motorola M68000	custom
aix	IBM's AIX on IBM System/370	custom
alnt	Alliant's CONCENTRIX on Motorola M68000	custom
cms	IBM's VM/SP CMS on IBM System/370	custom
hp20	HP's HP-UX on Motorola MC68020/MC68881	standard
hp38	SCO's XENIX on HP Vectra with Intel 80386	custom
hpux	HP's HP-UX on Motorola M68000	custom
ip32c	Intergraph's System V UNIX on Interpro 32C	custom
ipsc2	Intel's iPSC/2 System V UNIX on Intel 80386	custom
ix20	Apollo's DOMAIN/IX on Motorola MC68020/MC68881	standard
ixfpa	Apollo's DOMAIN/IX on Motorola MC68020/Weitek FPA	custom
ixpri	Apollo's DOMAIN/IX on Apollo PRISM	custom
sun2	Sun Microsystems' SunOS on Motorola M68000	custom
sun3	Sun Microsystems' SunOS on Motorola MC68020/MC68881	standard
sun38	Sun Microsystems' SunOS on Intel 80386	custom
sun4	Sun Microsystems' SunOS on SPARC	standard
ultrx	DEC's ULTRIX-32 on VAX-11	standard
vms	DEC's VAX/VMS on VAX-11	standard
xcms	IBM's VM/XA SP CMS on IBM System/370 Extended Architecture	custom

Table 1-1. Supported Platforms

```
<name of addressee>
<street number> <street name>
<town or city name>, <state abbreviation>  <zip code>
```

Example 1.1.2-1. Syntax of a Mailing Address

1.1.3. Temporary Features

Temporary features that have not acquired a final form are marked as follows:

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

Temporary features are subject to change or removal without notice. Programmers who make use of temporary features must be prepared to modify their code to accommodate the changes in them on each release of MAINSAIL. It is recommended that code that makes use of temporary features be as isolated from normal code as possible and thoroughly documented.

™

Aegis MAINSAIL®

User's Guide

24 March 1989

xidak™

2. Introduction

This document describes the MAINSAIL implementation for Aegis, the Apollo operating system for the M68000-based Apollo Domain computer. It describes only Aegis-specific MAINSAIL features. It assumes that the reader is familiar with the "MAINSAIL Language Manual" and other machine-independent documentation.

2.1. Version

This version of the "Aegis MAINSAIL User's Guide" is current as of Version 12.10 of MAINSAIL. It incorporates the "Aegis MAINSAIL Version 5.9 Release Note" of October, 1982; the "Aegis Version 7.4 Release Note" of May, 1983; the "Aegis MAINSAIL Release Note, Version 8" of January, 1984; the "Aegis MAINSAIL Release Note, Version 9" of February, 1985; the "Aegis MAINSAIL Release Note, Version 10" of March, 1986; and the "Aegis MAINSAIL Release Note, Version 11" of July, 1987.

3. General Operation

3.1. Invoking MAINSAIL

In order to run a MAINSAIL program, the Aegis link "m" must be defined in the naming directory to be the MAINSAIL directory. Example 3.1-1 shows how to define this link and run MAINSAIL. It assumes that the MAINSAIL directory is "/foo/bar/mainsail". If the link is already defined, the "crl" command is not needed.

```
$ crl ~m /foo/bar/mainsail<eol>
$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
```

Example 3.1-1. Creating the Link "~m" and Invoking MAINSAIL

MAINSAIL begins execution and types a herald identifying itself and the version of MAINSAIL being used. It then types "*" as a prompt and waits for input. The "*" prompt and possible responses to it are described in the MAINEX section of the "MAINSAIL Utilities User's Guide".

4. CONF, the MAINSAIL Configurator

This chapter assumes familiarity with the CONF section of the "MAINSAIL Utilities User's Guide".

The default CONF parameters are normally kept in the file "~m/aeg.cnf". Aegis systemwide changes should always be made to this file.

The output of CONF running under Aegis for Aegis is an Aegis executable binary file. When run under another operating system, Aegis CONF produces an assembly language file, which may be assembled using the Aegis assembler. The Aegis assembler must be obtained by special agreement with Apollo. In the example, the output file is called "mainsa.bin" and is the Aegis-dependent part of the MAINSAIL bootstrap. In order to run the new bootstrap, it must be bound with the file "~m/m.bin", which contains the portion of the bootstrap independent of the configuration values specified.

Example 4-1 shows a sample session with CONF and how to assemble and link the resulting bootstrap. Default values are restored from the file "~m/aeg.cnf" and the bootstrap is written to the file "mainsa.bin".

```
$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ~m/aeg.cnf
CONF: <eol>
Bootstrap written in file MAINSA.BIN
*<eol>
$ bind mainsa.bin ~m/m.bin -bin mainsa<eol>
All globals are resolved.
$ mainsa<eol>

  (mainsa executes)
```

Example 4-1. Using CONF to Make a Bootstrap

4.1. Errors from the Aegis "bind" Command

The Aegis binder (invoked by the shell's "bind" command) may give "unresolved global" errors if the list of Aegis system calls known to the MAINSAIL Uniform System Caller (see Chapter 6) contains calls not known to the binder. This can happen if the MAINSAIL Uniform System Caller is more up-to-date than the version of Aegis under which MAINSAIL is running, or if an Aegis system call library is not installed (the "GM_$" library is usually not installed by default; issue the command:

```
$ inlib /lib/gmrlib<eol>
```

to correct this). Unresolved global errors from the binder are completely harmless unless a program actually attempts to use the unresolved system call, in which case the operating system error "reference to undefined global" is generated.

5. Memory Usage under Aegis

MAINSAIL's use of the Aegis address space is complicated by the fact that Aegis allocates memory that is not directly under MAINSAIL's control. The part of the address space directly under MAINSAIL's control is limited to the size specified by CONF's "MAXMEMORYSIZE" command, and is contained in that part of memory displayed by MAINEX's "MAP" subcommand (except that free pages at the high and low ends of the normal MAINSAIL address space may not be displayed). Refer to the "MAINSAIL Utilities User's Guide" for more information on CONF and MAINEX. Mapped disk I/O buffers, mapped libraries, the MAINSAIL kernel, the MAINSAIL bootstrap, and a variety of things mapped by the operating system are not stored within the part of the address space under MAINSAIL's direct control.

5.1. Mapped Libraries

MAINSAIL module libraries under Aegis may be mapped or unmapped. Mapped libraries result in substantial execution efficiencies but may consume a great deal of memory. To specify that a library file is to be mapped, precede the file name with an asterisk in the call to openLibrary. For example:

```
openLibrary("*foo.lib")
```

maps all of "foo.lib" into the MAINSAIL process's address space, while:

```
openLibrary("foo.lib")
```

uses a less efficient form of I/O to read individual modules from "foo.lib".

5.2. GCCHP and the Disk File Cache

The utility GCCHP may be used to set the file cache size for non-REC random disk I/O. Aegis MAINSAIL does not use the standard caching mechanism when mapping files is enabled (the default), so GCCHP functions differently from the way it is described in the "MAINSAIL Utilities User's Guide". Only the "requestedMaxSize" parameter is used by the Aegis disk module. This parameter is measured in 1K pages. Since buffers tend to be 32 pages long, the value for requestedMaxSize should be a multiple of 32. The amount of buffer space allocated for sequential disk I/O is not directly controllable by the user, although a maximum of 64 pages is mapped for each sequential non-REC file.

5.3. LAS Utility

An Aegis-specific utility, LAS, invokes the Aegis LAS command from within MAINSAIL. This command shows the contents of the Aegis address space and is sometimes useful in debugging. Information on the format of LAS output may be found in the Apollo manuals, which are not supplied by XIDAK.

XIDAK will support the LAS module only as long as Apollo supports the LAS command.

5.4. Aegis Stack Size

The initial coroutine "MAINSAIL" on Aegis uses the system stack, which is quite large; the configuration "STACKSIZE" parameter is ignored when the initial coroutine is allocated.

6. Aegis System Calls

Aegis system calls are Pascal procedure calls. The Uniform System Caller is available as an alternative method to the Foreign Language Interface for making an Aegis system call.

The Uniform System Caller allows Aegis system calls to be included in a MAINSAIL program without going through all the steps necessary to use the Foreign Language Interface. To call an Aegis system call with the Uniform System Caller, the user makes a call with the appropriate parameters to the Aegis-specific system procedure "$sysCall", "$lbSysCall", or "$aSysCall".

XIDAK attempts to keep the list of Aegis system calls callable through the Uniform System Caller up-to-date; however, Apollo adds new system calls from time to time, and the newest Aegis calls may not yet have been installed in MAINSAIL. Uninstalled calls must be made through the Foreign Language Interface (see Chapter 7).

6.1. Making a System Call

A system call is made in one of three ways depending on the type of the return value of the system call, if any. If the system call returns a Pascal pointer value (a MAINSAIL address), use "$aSysCall". If the call's return value is something other than a Pascal pointer or Pascal double, use "$lbSysCall". If a system call returns a Pascal double, treat the return value as an implicit extra "VAR" parameter at the end of the Pascal parameter list. If the call is an untyped Pascal procedure, use "$sysCall". "$sysCall", "$lbSysCall", and "$aSysCall" take n + 1 parameters, where n is the number of parameters in the Pascal calling sequence of the system call. The first parameter is used to determine which system call is being made; it is an identifier formed by taking the Apollo name for the system call and removing all the occurrences of the characters "_" and "$" and then adding "$" to the front and the word "Call" to the end. For example, GPR_$PIXEL_BLT becomes "$gprPixelBltCall". The remaining parameters are the addresses of the parameters being passed to the Apollo system call.

For example, assume a call is to be made to GPR_$MOVE. This call has three parameters and no return value. Assume there are three address variables called "a1", "a2", and "a3", which have been set up to point to the parameters as Pascal expects them. Example 6.1-1 shows the call to this procedure.

"$sysCall" was used (rather than "$aSysCall" or "$lbSysCall") because GPR_$MOVE does not return a value. An example of a system call that does return a value is PFM_$CLEANUP.

```
$sysCall($gprMoveCall,a1,a2,a3);
```

Example 6.1-1. Calling GPR_$MOVE

This call returns a non-address value, so it uses "$lbSysCall", as in Example 6.1-2. The status record returned by PFM_$CLEANUP is 32 bits and can be mapped to a MAINSAIL long bits.

```
lb := $lbSysCall($pfmCleanupCall,a1);
```

Example 6.1-2. Calling PFM_$CLEANUP

An example of a call that returns an address value is MS_$REMAP. MS_$REMAP, therefore, requires the use of "$aSysCall". The call is shown in Example 6.1-3.

```
a1 := $aSysCall($msReMapCall,a2,a3,a4,a5,a6);
```

Example 6.1-3. Calling MS_$REMAP

6.2. Parameter Passing

To pass a value to a system routine, the value must be stored in memory and the address of the value passed to the uniform system caller. Likewise, to receive a value from a system call, a place in memory must be allocated for the return value and then upon return from the call the value must be loaded from memory. To simplify the process of loading and restoring the parameters, macros like those in Example 6.2-1 are helpful.

These macros and the address variable "a" must be declared in any module that uses them. "startParms" should be called before each system call to initialize "a", which functions as a pointer into the scratch page. "put" is used with input and input-output parameters and "skip" is used with output parameters. Example 6.2-2 shows a "wrapper procedure" for

```
DEFINE
    startParms =
        [BEGIN a := <address of scratch space> END],
    put(what,where) =
        [BEGIN where := a; write(a,(what)) END],
    skip(howMany,where) =
        [BEGIN where := a;
         a := displace(a,(howMany)) END];
```

Example 6.2-1. Useful Macros

GPR_$SET_BITMAP, which presents a simpler MAINSAIL calling sequence than invoking the Uniform System Caller directly.

```
LONG BITS PROCEDURE gprSetBitMap
    (LONG INTEGER bitMapDesc);
BEGIN
ADDRESS a,a1,a2;                # To use the above macros,
                                # there must always be an
                                # ADDRESS variable called
                                # "a".  The other variables
                                # can have any name.
startParms;
put(bitMapDesc,a1);             # Load bitMapDesc into
                                # memory at address a1.
skip(size(longBitsCode),a2);    # Allocate a location in
                                # memory for the status to
                                # be written to at address
$sysCall                        # a2. Call GPR_$SET_BITMAP.
    ($gprSetBitMapCall,a1,a2);  # Retrieve the value at a2
RETURN(lbLoad(a2));             # (the status) and use this
                                # as the return value.
END;
```

Example 6.2-2. Calling GPR_$SET_BITMAP

As used in these examples, Pascal parameters of types corresponding to the MAINSAIL types integer, long integer, real, long real, bits, long bits, pointer, address, and charadr each correspond to exactly one invocation of the "put" and "skip" macros. The amount of space to skip with the "skip" macro is based on the size of the MAINSAIL type corresponding to the Pascal type of the output parameter (see Table 7.2-1). It may be convenient to use the MAINSAIL procedure "size" as documented in the "MAINSAIL Language Manual".

6.3. Common Pitfalls

Boolean variables are not represented the same in MAINSAIL as they are in Pascal. Pascal uses a byte that is all 0's for FALSE and all 1's for TRUE. MAINSAIL uses a 16-bit word that is 1 for TRUE and 0 for FALSE. Example 6.3-1 shows how to pass a MAINSAIL BOOLEAN "bo" as an input parameter to Pascal.

```
put(IF bo THEN -1 EL 0,a1)
```

Example 6.3-1. How to Pass a MAINSAIL BOOLEAN to Pascal

Since the M68000 uses twos'-complement form to represent negative numbers, this sets all the bits in the byte at "a1" (as well as at "a1 + 1").

String parameters are also represented differently between MAINSAIL and Pascal. Follow the format used in Example 6.3-2.

There are many ways to pass records and arrays, one of which is shown in Example 6.3-3. The second parameter of GPR_$PIXEL_BLT is a two-dimensional array with a total of four elements. In this example each element of the array is passed to the MAINSAIL wrapper procedure as a separate parameter. When a MAINSAIL array corresponds to a Pascal array, $adrOfFirstElement may be used to find the address of the start of the MAINSAIL array.

```
LONG BITS PROCEDURE gprText (STRING text);
BEGIN
ADDRESS a,a1,a2,a3;
startParms;
store(a,cvc(text));           # Convert the string to a
                              # CHARADR and store it at
                              # location "a".
a1 := aLoad(a);               # Convert the stored CHARADR
                              # into the address "a1".
put(length(text),a2);         # The APOLLO system calls
                              # need to be passed the
                              # length of the string.
skip(size(longBitsCode),a3);
$sysCall($gprTextCall,a1,a2,a3);
RETURN(lbLoad(a3));
END;
```

Example 6.3-2. Calling GPR_$TEXT

```
LONG BITS PROCEDURE gprPixelBlt (
    LONG INTEGER     sourceBitmapDesc;
    INTEGER          xSource,ySource,
                     windowWidth,windowHeight;
    INTEGER          xDest,yDest);
BEGIN
ADDRESS a,a1,a2,a3,a4,ax;        # GPR_$PIXEL_BLT has 4
                                 # parameters.  ax is just
                                 # a placeholder.

startParms;
put(sourceBitmapDesc,a1);
put(xSource,a2);                 # The first element of the
                                 # Pascal array (the
                                 # address to be given to
                                 # GPR_$PIXEL_BLT).
put(ySource,ax);                 # The next three
put(windowWidth,ax);             # parameters are the next
put(windowHeight,ax);            # three array elements,
                                 # stored in consecutive
                                 # locations.
put(xDest,a3);                   # This is a 2-element
put(yDest,ax);                   # array done as above.
skip(size(longBitsCode),a4);
$sysCall($gprPixelBltCall,a1,a2,a3,a4);
RETURN(lblLoad(a4));
END;
```

Example 6.3-3. Calling GPR_$PIXEL_BLT

7. Foreign Language Interface

This chapter contains Aegis-specific information for the MAINSAIL Foreign Language Interface (FLI). Refer to the "MAINSAIL Compiler User's Guide" for a general description of the FLI.

Both the Foreign Call Compiler ("FCC") and MAINSAIL Entry Compiler ("MEC") are available for Aegis.

The default transformation used to derive a Pascal procedure name from its MAINSAIL procedure name (as declared in a module compiled by one of the FLI compilers) is to strip off the leading "$" character, if any, from the MAINSAIL name, and use the resulting string as the name of the Pascal routine. In the case of the MAINSAIL-to-Pascal compiler, this is the name of the external routine called by MAINSAIL; in the case of the Pascal-to-MAINSAIL compiler, it is the name to be used by Pascal in calling MAINSAIL. With either compiler, it is possible to override the default name by using the "ENCODE" directive.

On Aegis, FORTRAN uses the same calling convention as Pascal, so the Pascal FLI is used to interface to both languages; i.e., use the compiler subcommand "FLI TP" or "FLI FP" when interfacing to Aegis FORTRAN.

7.1. Output File Names

The default output file name for an Aegis FLI compiler is "<module name>.BIN" if running on Aegis, or "<module name>.ASM" if running on another system.

7.2. Data Types

MAINSAIL parameter types are mapped into Pascal parameter types according to Table 7.2-1. MAINSAIL parameter types are mapped into FORTRAN parameter types according to Table 7.2-2. These mappings apply both to the FCC and to the MEC.

The MAINSAIL procedure declaration corresponding to the Pascal routine may return any type but string, pointer, or array. A typed MAINSAIL procedure maps into a Pascal FUNCTION of the appropriate type.

MAINSAIL parameter	Pascal parameter
USES BOOLEAN	non-VAR boolean
MODIFIES or PRODUCES BOOLEAN	VAR boolean
USES INTEGER or BITS	non-VAR integer16
MODIFIES or PRODUCES INTEGER or BITS	VAR integer16
USES REAL	non-VAR single
MODIFIES or PRODUCES REAL	VAR single
USES LONG REAL	non-VAR double
MODIFIES or PRODUCES LONG REAL	VAR double
USES STRING	ARRAY OF char, integer16
MODIFIES STRING	VAR ARRAY OF char, VAR integer16
PRODUCES STRING	VAR POINTER to ARRAY OF char, VAR integer16
USES POINTER	VAR or non-VAR RECORD
MODIFIES or PRODUCES POINTER	not allowed
USES ARRAY	VAR or non-VAR ARRAY
MODIFIES or PRODUCES ARRAY	not allowed
USES ADDRESS or CHARADR	non-VAR pointer
MODIFIES or PRODUCES ADDRESS or CHARADR	VAR pointer

Table 7.2-1. MAINSAIL and Pascal Parameter Types (continued)

```
"VAR" is equivalent to "OUT" or "IN OUT" in Aegis
Pascal parameter qualifications; "non-VAR" means "IN" or
no parameter qualifier.
```

Table 7.2-1. MAINSAIL and Pascal Parameter Types (end)

MAINSAIL parameter	FORTRAN parameter
BOOLEAN, INTEGER, or BITS	INTEGER*2
REAL	REAL*4
LONG REAL	REAL*8
USES or MODIFIES STRING	CHARACTER*n, INTEGER*2
PRODUCES STRING	INTEGER*4 (pointer to character array), INTEGER*2
USES POINTER	no corresponding data type
MODIFIES or PRODUCES POINTER	not allowed
USES ARRAY	array
MODIFIES or PRODUCES ARRAY	not allowed
ADDRESS or CHARADR	INTEGER*4 (pointer)

Table 7.2-2. MAINSAIL and FORTRAN Parameter Types

A single MAINSAIL string parameter corresponds to two Pascal parameters. Both Pascal parameters (an array or array pointer parameter and an integer16 parameter) must be declared in the Pascal procedure header so that Pascal can figure out the length of the MAINSAIL string. Only one string parameter appears in the MAINSAIL procedure header. In Pascal, the character array (or array pointer) must be declared first, immediately followed by the integer16 length. For Pascal routines returning strings as a pointer to a character array and a length, use a MAINSAIL produces string variable.

In the case of a MAINSAIL uses string (or a MAINSAIL modifies string not constructed with a call to the MAINSAIL system procedure "newString") the Pascal routine must not modify the contents of the array, which may actually be in MAINSAIL's string space. For a Pascal routine that uses a Pascal VAR ARRAY to return a character string, use a MAINSAIL modifies string argument, where the string was constructed with the MAINSAIL procedure "newString". The charadr passed to newString should point into empty memory (e.g., memory obtained by a call to newScratch), and the length of the string should be the same as the length of the Pascal array. A MAINSAIL uses string must also be allocated in scratch space. A string in MAINSAIL string space should be passed as a MAINSAIL modifies string to foreign code, but the foreign code must not actually modify a string in MAINSAIL string space. Pascal must not access or modify characters in the array beyond the end of a MAINSAIL uses or modifies string. A MAINSAIL modifies string may be the null string when it is passed, but the Pascal routine must not access the array part or change the length part in this case. Pascal can detect a null string by observing that the integer16 length is 0.

If a MAINSAIL pointer or address parameter is classified, care must be taken to ensure that analogous fields in the Pascal record type are at the same offset as the MAINSAIL fields, especially if the Pascal record type is a packed record. Refer to Apollo's Pascal reference manuals for descriptions of how Pascal arranges data structures in memory, and to Chapter 12 in this document for the corresponding information about MAINSAIL.

If a MAINSAIL pointer is passed it must not be nullPointer.

Pascal may modify the contents of an array, even if it was passed as a uses parameter from MAINSAIL. Care must be taken not to access any value outside the bounds of the MAINSAIL array, since Pascal cannot automatically check the bounds of the MAINSAIL array. It is never legal to pass a MAINSAIL nullArray to Pascal. A MAINSAIL boolean array must be treated as an array of integer16 in Pascal (or an INTEGER*2 array in FORTRAN), where 0 represents false and 1 represents true, since the representation of booleans differs among the three languages.

If a MAINSAIL array is passed to Pascal as a non-VAR array, the MAINSAIL array must be at least as large as the Pascal array is declared to be. Pascal copies non-VAR arrays on procedure entry; if the MAINSAIL structure is smaller than the Pascal structure, and the MAINSAIL structure is located near the end of the memory MAINSAIL has acquired from Aegis, then an access violation may result as Pascal attempts to copy the structure.

FORTRAN multidimensional arrays are stored differently from MAINSAIL and Pascal arrays. In FORTRAN, the leftmost subscript of a multidimensional array varies most rapidly. In MAINSAIL and Pascal, the rightmost subscript varies most rapidly.

7.3. Foreign Call Compiler Example

Suppose that a MAINSAIL module FOOSUB is to call a Pascal procedure PROC1. Figure 7.3-1 shows the module FOOSUB, the Pascal subroutine PROC1, and the "dummy" MAINSAIL module FLISUB that defines this foreign procedure.

Examples 7.3-2 and 7.3-4 show the steps necessary before the module FOOSUB can be run.

```
Compile FOOSUB for Aegis with the MAINSAIL
compiler.

Compile FLISUB with the "FLI TP" (FLI to Pascal) compiler
subcommand:

$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): flisub.msl,<eol>
> fli tp<eol>
> <eol>
Opening intmod for $SYS...

flisub.msl 1
Output for FLISUB on flisub.bin
Intmod for FLISUB not stored

compile (? for help): <eol>
```

Example 7.3-2. Compiling FLI Modules

```
MAINSAIL module FOOSUB (in "foosub.msl"):
----------------------------------------------------------
BEGIN "fooSub"

CLASS c1 (INTEGER i; LONG REAL rr);

MODULE fliSub (
    PROCEDURE proc1 (PRODUCES INTEGER i;
                     INTEGER j,k;
                     MODIFIES STRING s;
                     MODIFIES BOOLEAN bo;
                     BOOLEAN ARRAY(1 TO 2) ary;
                     POINTER(c1) p;
                     CHARADR c; INTEGER len;
                     MODIFIES ADDRESS(c1) a);
);

INITIAL PROCEDURE;
BEGIN
BOOLEAN bo;
INTEGER i,j,k,len;
STRING s;
POINTER(c1) p;
ADDRESS(c1) a;
BOOLEAN ARRAY(1 TO 2) ary;
s := "Hello there"; bo := TRUE;
new(ary); ary[1] := TRUE; ary[2] := FALSE;
p := new(c1); p.i := 1; p.rr := 2.0L;
c := cvc(s); len := length(s);
a := cva(p);
proc1(i,1,2,s,bo,ary,p,c,len,a);
ttyWrite("i = ",i,eol &
    "bo = ",IF bo THEN "TRUE" EL "FALSE",eol &
    "a.i = ",a.i,"; a.rr = ",a.rr,eol);
END;

END "fooSub"
----------------------------------------------------------
```

Figure 7.3-1. Declarations for FCC Example (continued)

```
MAINSAIL "dummy" module FLISUB (in "flisub.msl"):
-----------------------------------------------------------
BEGIN "fliSub"

MODULE fliSub (
    PROCEDURE proc1 (PRODUCES INTEGER i;
                     INTEGER j,k;
                     MODIFIES STRING s;
                     MODIFIES BOOLEAN bo;
                     BOOLEAN ARRAY(1 TO 2) ary;
                     POINTER(c1) p;
                     CHARADR c; INTEGER len;
                     MODIFIES ADDRESS(c1) a);
);

PROCEDURE proc1 (PRODUCES INTEGER i;
                 INTEGER j,k;
                 MODIFIES STRING s;
                 MODIFIES BOOLEAN bo;
                 BOOLEAN ARRAY(1 TO 2) ary;
                 POINTER(c1) p;
                 CHARADR c; INTEGER len;
                 MODIFIES ADDRESS(c1) a);;

END "fliSub"
-----------------------------------------------------------



Pascal subroutine PROC1 (in "psub.pas"):
-----------------------------------------------------------
TYPE
    charArray = ARRAY[1..32767] OF char;
    charArrayPtr = ^charArray;
    int16Array = ARRAY[1..2] OF integer16;
    c1 = RECORD
            i: integer16;
            rr: double;
        END;
    c1Ptr = ^c1;
```

Figure 7.3-1. Declarations for FCC Example (continued)

```
PROCEDURE proc1 (VAR i: integer16; j,k: integer16;
                 VAR ch: charArray;
                 VAR len: integer16;
                 VAR bo: boolean;
                 ary: int16Array;
                 p: c1;
                 ch2: charArrayPtr;
                 len2: integer16;
                 VAR a: c1Ptr);
VAR n: integer16;
BEGIN
    i := j + k;
    writeln('String parameter is:');
    FOR n := 1 TO len DO write(ch[n]); writeln;
    bo := NOT bo;
    { Swap the boolean array elements: }
    n := ary[1]; ary[1] := ary[2]; ary[2] := n;
    writeln('p.i = ',p.i,'; p.rr = ',p.rr);
    writeln('Second string parameter is:');
    FOR n := 1 TO len2 DO write(ch2^[n]); writeln;
    new(a); a.i := p.i; a.rr := p.rr;
END;
---------------------------------------------------------
```

Figure 7.3-1. Declarations for FCC Example (end)

7.4. MAINSAIL Entry Compiler Example

Suppose that the Pascal procedure callms is to call the MAINSAIL procedure proc1. Figures 7.4-1, 7.4-2, 7.4-3, and 7.4-4 show the Pascal procedure callms, the MAINSAIL module MSMOD that contains the procedure proc1, the MAINSAIL FLI module TOPAS, and the MAINSAIL module CALPAS that calls the Pascal procedure callms, respectively. Example 7.4-5 shows how to compile and run callms.

```
Compile the Pascal file "psub.pas" with the Pascal
compiler (PAS command from the Aegis shell) to produce
the object file "psub.bin".

Run the MAINSAIL utility CONF.

$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ~m/aeg.cnf
CONF: foreignmodule flisub<eol>
CONF: <eol>
Bootstrap written in file MAINSA.BIN
```

Example 7.3-3. Compiling the Pascal File and Running CONF

```
Link the new bootstrap with the FLI code and the Pascal
object module into an executable bootstrap called "X":

$ bind mainsa.bin ~m/m.bin flisub.bin psub.bin -bin x<eol>

Run the MAINSAIL module FOOSUB:

$ x<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*foosub<eol>

  (FOOSUB executes, calling the Pascal routine)
```

Example 7.3-4. Linking and Running an FLI Module

```
Pascal module (in file "callms.pas"):
MODULE callms_module;

FUNCTION proc1 (i1,i2: integer32; VAR bo: integer16):
    integer32;
EXTERN;

PROCEDURE callms;
{
    MAINSAIL proc1 does the following:
    (1) Adds its first two arguments and returns
        the result
    (2) Sets bo to be TRUE
}
VAR bo: integer16;
    result,li1,li2: integer32;
BEGIN
bo := 0; li1 := 1; li2 := 2;
result := proc1(li1,li2,bo);
writeln('Result is ',result);
IF bo = 0 THEN
    writeln("FAILURE: bo should be nonZero");
END;
```

Figure 7.4-1. Pascal Procedure That Calls MAINSAIL Procedure proc1

```
MAINSAIL Module MSMOD (in file "msmod.msl"):

BEGIN "msMod"

MODULE msMod (
    LONG INTEGER PROCEDURE procl (
        LONG INTEGER        lil,li2;
        PRODUCES BOOLEAN    bo);
    );

LONG INTEGER PROCEDURE procl (
    LONG INTEGER        lil,li2;
    PRODUCES BOOLEAN    bo);
BEGIN
bo := TRUE;
RETURN(lil + li2);
END;

END "msMod"
```

Figure 7.4-2. MAINSAIL Module MSMOD Called by Pascal Procedure callms

```
MAINSAIL Module TOPAS (in file "topas.msl"):

BEGIN "toPas"

MODULE toPas (PROCEDURE callMs);

PROCEDURE callMs;;

END "toPas"
```

Figure 7.4-3. MAINSAIL Foreign Language Interface Module TOPAS

```
MAINSAIL Module CALPAS (in file "calpas.msl"):

BEGIN "calPas"

MODULE toPas (PROCEDURE callMs);

INITIAL PROCEDURE;
callMs;

END "calPas"
```

Figure 7.4-4. MAINSAIL Module CALPAS That Calls Pascal Procedure callms

```
(1) Compile MSMOD and CALPAS with the Aegis MAINSAIL
    compiler.  Compile MSMOD with the MEC from Pascal (by
    specifying the compiler subcommand "fli fp").  Compile
    TOPAS with the FCC to Pascal (by specifying the
    compiler subcommand "fli tp").

$ mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): msmod.msl<eol>

msmod.msl 1 ...

compile (? for help): calpas.msl<eol>

calpas.msl ...
```

Example 7.4-5. Pascal to MAINSAIL Example (continued)

```
compile (? for help): msmod.msl,<eol>
>fli fp<eol>
><eol>

msmod.msl 1 ...

compile (? for help): topas.msl,<eol>
>fli tp<eol>
><eol>

topas.msl ...

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

(control returns to the Aegis shell)

(2) Make a new MAINSAIL bootstrap that declares TOPAS to
    be a foreign module.

$ mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ~m/aeg.cnf
CONF: bootfilename mec.bin<eol>
CONF: foreignmodules<eol>
FOREIGNMODULES is
...
Should be:
=<eol>
TOPAS<eol>
<eol>
CONF: <eol>
Bootstrap written in file mec.bin
*<eol>

(control returns to the Aegis shell)
```

Example 7.4-5. Pascal to MAINSAIL Example (continued)

```
(3) Compile the Pascal code with the Pascal compiler.

$ pas callms<eol>

(4) Bind the new MAINSAIL bootstrap.


$ bind mec.bin msmod.bin topas.bin callms.bin ~m/m.bin
        -bin mec<eol>

(5) Run the new executable MAINSAIL bootstrap and call
    the foreign procedure.

$ mec<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*calpas<eol>
Result is          3
*<eol>
```

Example 7.4-5. Pascal to MAINSAIL Example (end)

7.5. $foreignCodeStartsExecution

If a foreign-language program initiates execution instead of MAINSAIL, the configuration bit $foreignCodeStartsExecution must be set in the MAINSAIL bootstrap. Consult the description of the FLI in "MAINSAIL Compiler User's Guide" and the description of CONF in the "MAINSAIL Utilities User's Guide" for details.

Example 7.5-1 shows a Pascal program that calls the MAINSAIL module of Example 7.5-2. The steps necessary to compile, configure, link, and execute the program are shown in Example 7.5-3 ('10 is the $foreignCodeStartsExecution bit). When a foreign module calls a MAINSAIL module, there is no need for a "fake" module that imitates the foreign module's interface; it is the MAINSAIL module called from the foreign language that is compiled with the MEC.

```
PROGRAM call;

VAR i,j: integer;

FUNCTION mslProc (i,j: integer16): integer16;
EXTERN;

BEGIN
i := 33;
j := 17;
writeln('mslProc(i,j) is ',mslProc(i,j));
END.
```

Example 7.5-1. Pascal Main Program (in "call.pas")

```
BEGIN "mslMod"

MODULE mslMod (
    INTEGER PROCEDURE mslProc (INTEGER i,j);
);

INTEGER PROCEDURE mslProc (INTEGER i,j);
RETURN(i + j);

END "mslMod"
```

Example 7.5-2. MAINSAIL Module Called from Pascal (in "mslmod.msl")

```
$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): mslmod.msl<eol>
Opening intmod for $SYS...

mslmod.msl 1
Objmod for MSLMOD stored on mslmod-aeg.obj
Intmod for MSLMOD not stored

compile (? for help): mslmod.msl,<eol>
> fli fp<eol>
> <eol>
Opening intmod for $SYS...
```

Example 7.5-3. Calling MAINSAIL from a Pascal Main Program (continued)

```
mslmod.msl 1
Opening intmod for $SYS...
Output for MSLMOD stored on MSLMOD.BIN
Intmod for MSLMOD not stored

compile (? for help): <eol>
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ~m/aeg.cnf
CONF: bootfilename fli<eol>
CONF: configurationbits<eol>
CONFIGURATIONBITS is 'H0, should be: '10<eol>
CONF:
Bootstrap written in file FLI.BIN
*<eol>
$ pas call<eol>
No errors, no warnings, Pascal Rev 6.1413
$ bind call.bin fli.bin mslmod.bin ~m/m.bin -bin fli<eol>
All globals are resolved.
$ fli<eol>
mslProc(i,j) is        50
$
```

Example 7.5-3. Calling MAINSAIL from a Pascal Main Program (end)

8. Terminal I/O

8.1. Line-Oriented Mode

In the line-oriented mode of operation, MAINSAIL buffers terminal output until:

1. A linefeed character (<eol>) is output,

2. The output buffer is full, or

3. A terminal read occurs.

In each case, the buffer is output as a line to the terminal. Thus, output characters are not seen on the terminal until one of the above events occurs.

8.2. Aegis and MAINEDIT

The display modules BORRO, FRAME, FBORRO, and FFRAME are used with the Aegis bitmap display. These display modules are documented in the appendices to the "MAINEDIT User's Guide".

9. File I/O

9.1. Disk I/O

MAINSAIL supports several disk file formats under Aegis: UASC (standard Aegis format for text files), REC (old Aegis record-structured text file format), UNDEF (Aegis HDRU byte stream format), OBJ (Aegis object file format, identical for all practical purposes to UNDEF), and NIL (non-stream disk file format). The default format for new text files created by MAINSAIL is UASC; for new data files it is UNDEF. Data files and random output files may not be in REC format. When MAINSAIL replaces an existing file, the new file is in the default format unless the format is explicitly specified.

To specify a format (REC, for example) to be used for a file FOO, use the syntax "REC>FOO" for the file name parameter to the MAINSAIL procedure "open". "HDRU>" is accepted as a synonym for "UNDEF>" in such a format specification. "VAR>" is a synonym for "REC>", and "BS>" is a synonym for "UASC>" for a text file, or for "HDRU>" for a data file.

Consider the case in which a file "FOO" is a UASC text file created with the Apollo editor, and you wish to copy it to REC format file. The MAINSAIL module COPIER can be run as shown in Example 9.1-1. Refer to the "MAINSAIL Utilities User's Guide" for a complete description of COPIER. Example 9.1-1 also shows the reverse process. In the first part of the example, "FOO.NEW" is in REC format because the format has been explicitly specified. In the second part of the example, "FOO" is a UASC file because the default for new sequential text files is the UASC format.

Although it is possible to perform random access input on a REC file, it is not possible to perform random output on such a file. Explicitly specifying the REC device prefix for a random output file results in an error.

9.2. Disabling Aegis Mapping

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

The $disableAegisMapping configuration bit (value 'H8000) prevents Aegis MAINSAIL from employing mapping calls for disk I/O. Instead, Aegis stream calls are used. Users may desire

```
Copying a UASC file to REC format file.

$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*copier<eol>
Input file: foo<eol>
Output file: rec>foo.new<eol>
Input file: <eol>
*<eol>

------------------------------------------------

Copying a REC format file to a UASC file (assume
a file called "foo" does not presently exist).

$ ~m/mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*copier<eol>
Input file: foo.new<eol>
Output file: foo<eol>
Input file: <eol>
*<eol>
```

Example 9.1-1. Converting Between File Formats

to set this bit because the Aegis operating system contains a bug which causes Aegis to run out of memory after enough mapped I/O has taken place.

After Apollo fixes the bug in Aegis and the versions of Aegis with the bug become obsolete, the $disableAegisMapping configuration bit will be eliminated.

9.3. MAINSAIL and Aegis Links

Aegis standard utilities do not permit a file to be created or deleted if it is referred to using a link pointing to the file. MAINSAIL also obeys this convention and gives an error if a link name is given in an open with the create or delete bit set. If it is necessary to create or delete a file, specify the name of the file, not the name of a link pointing to the file.

9.4. MAINSAIL and the Serial Ports

It is possible to run MAINSAIL on an ordinary computer terminal connected to one of the serial ports available on the back of the Apollo cabinet. Make sure the terminal is correctly connected and use the Apollo shell's TCTL command in order to configure the serial line to the appropriate baud rate, parity, and so on. If you do not give the "-ECHO" switch to TCTL, your keystrokes will not echo as you type them to MAINSAIL. Also, many terminals attempt to send XON and XOFF to the Apollo, so that both the "-SYNC" and "-INSYNC" TCTL commands should usually be in effect.

When the terminal and serial line have been correctly configured, run MAINSAIL, directing standard input and standard output to the serial line. Figure 9.4-1 shows how this is done using port 2.

```
$ ~m/mainsa </dev/sio2 >/dev/sio2<eol>
```

Figure 9.4-1. Directing Standard I/O to the Serial Line

If the MAINSAIL editor is available for the type of terminal you are using, it may be run over the serial line. It is also possible to use the serial line to run a debugging session for a program that borrows the Apollo bitmap display, since the MAINSAIL debugger can communicate with the user over the serial line.

MAINSAIL can use the serial lines as regular sequential input and output files. Example 9.4-2 shows how to copy a text file "FOO" to serial line 2. The prefix "SIO>" is required for SIO line files.

```
*copier
Input file: foo<eol>
Output file: sio>/dev/sio2<eol>
*
```

Example 9.4-2. Copying a File to a Serial Port

10. System Information Procedures

10.1. $homeDirectory

$homeDirectory returns the string provided by PM_$GET_HOME_TXT.

10.2. Command Line and $programName

$programName is set to the first element of the argument vector returned by PGM_$GET_ARGS. The command line is formed from the remaining elements, separated by spaces.

10.3. Exit Codes

The exit code is converted to a two-byte integer and passed to PGM_$SET_SEVERITY before MAINSAIL exits. $successExitCode is PGM_$OK ('0L) and $failureExitCode is PGM_$ERROR ('3L).

10.4. $currentDirectory

$currentDirectory returns the name of the current working directory, as given by NAME_$GET_WDIR.

10.5. $directory (for Aegis Disk Files)

$directory returns a list of files as given by NAME_$READ_DIR. $reportAllVersions is ignored.

10.6. $fileInfo (for Aegis Disk Files)

$fileInfo fills in the fields of $fileInfoClass as follows:

- $fullPathName is obtained from NAME_$GET_PATH.

- $OSDSize is the number of 1024-byte blocks occupied by the file.

- $createDate, $createTime, $modifyDate, and $modifyTime are as shown in an Aegis directory listing.

10.7. $userID

$userID returns the ID of the current user as a string of the form "<name>.<project>.<org>".

10.8. $cpuID

$cpuID returns a string that is the node ID in hexadecimal.

11. Aegis Faults

The faults intercepted by MAINSAIL as of March, 1989 are shown in Table 11-1. This list is subject to revision as Apollo defines new faults.

11.1. Determining Which Faults to Catch

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

If you wish MAINSAIL to intercept an Aegis fault not on the default list, you may write a foreign language procedure "catchFault". The MAINSAIL declaration and corresponding Pascal code for catchFault are shown in Figure 11.1-1.

Call catchFault with the bit pattern of the fault status code corresponding to each fault which you want MAINSAIL to intercept. The return value from catchFault is the error status from pfm_$establish_fault_handler.

Hex Code	Fault: Standard Aegis Error Message
00040004	MST manager: reference to illegal address
00040005	MST manager: reference to out-of-bounds address
00120001	fault handler: odd address error
00120002	fault handler: illegal instruction
00120003	fault handler: integer divide by zero
00120004	fault handler: CHK instruction trapped...
00120005	fault handler: arithmetic overflow
00120006	fault handler: privileged instruction violation
00120007	fault handler: invalid SVC code
00120008	fault handler: invalid SVC procedure name
00120009	fault handler: undefined TRAP instruction
0012000A	fault handler: unimplemented instruction
0012000B	fault handler: protection boundary violation
00120011	fault handler: access violation
00120016	fault handler: invalid user-generated fault...
00120017	fault handler: fault in user-space interrupt handler for pbu device
0012001C	fault handler: unimplemented SVC
0012001D	fault handler: invalid stack format
00120023	fault handler: floating point divide by zero
00120025	fault handler: floating point operand error
00120026	fault handler: floating point overflow
00240000	PEB manager: all faults
03030009	loader: reference to undefined global
05090000	floating point manager: all faults
09010004	AUX: illegal instruction fault
09010005	AUX: trace trap fault
09010006	AUX: IOT instruction fault
09010007	AUX: EMT instruction fault
09010008	AUX: floating point exception fault
0901000A	AUX: bus error fault
0901000B	AUX: segmentation violation fault
0901000C	AUX: bad argument to system call fault
0901000D	AUX: broken pipe fault

Table 11-1. Aegis Faults Intercepted by MAINSAIL

```
The MAINSAIL catchFault declaration:

LONG BITS PROCEDURE catchFault (LONG BITS faultCode);

The Pascal code for catchFault:

%INCLUDE '/sys/ins/base.ins.pas';
%INCLUDE '/sys/ins/pfm.ins.pas';

        .
        .
        .

FUNCTION PascalFaultHandler (IN rec: pfm_$fault_rec_t):
    pfm_$fh_func_val_t; EXTERN;

FUNCTION catchFault (fault: integer32): status_$t;
VAR
    pfmHandle: pfm_$fh_handle_t;
    status: status_$t;
BEGIN
pfmHandle := pfm_$establish_fault_handler
    (fault,[],addr(PascalFaultHandler),status);
catchFault := status;
END;
```

Figure 11.1-1. How to Specify Which Fault to Catch

12. M68000 and MC68020 Processor-Dependent Information

This chapter describes M68000 Family-specific information.

12.1. M68000 vs. MC68020 Code Generation

XIDAK's M68000 code generator produces code that runs on the entire line of processors supporting Motorola's M68000 Family architecture, including the MC68000, MC68010, and MC68020. The M68000 code generator does not take advantage of any of the special instructions available on the latter two processors. The MC68020-specific code generators produce code that runs more efficiently on the MC68020 and the accompanying floating point processors included by various manufacturers with the MC68020 (the supported floating point processors at present are the Motorola MC68881 and the Weitek FPA). Code produced for the MC68020 does not run on earlier processors in the M68000 Family.

Except where otherwise specified, features described for the M68000 apply to the MC68020 as well.

12.2. Procedure Size

There is no well-defined limit for the size of a procedure on the M68000. However, the compiler almost always successfully compiles procedures under 32K bytes in length, and may handle larger procedures, depending on the code. Procedures longer than approximately 32K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

12.3. M68000 Data Types

Refer to Table 12.3-1. A storage unit on the M68000 is one byte (8 bits).

Data type	Representation
boolean	1 word (2 bytes)
integer	Standard M68000 integer format (1 word, 2 bytes)
long integer	Standard M68000 long integer format (1 longword, 4 bytes)
real	Depends on operating system, usually 1 longword (4 bytes)
long real	Depends on operating system, usually 2 longwords (8 bytes)
bits	1 word (2 bytes)
long bits	1 longword (4 bytes)
string	2 longwords (8 bytes): first longword (low address) is length, second longword (high address) is charadr of first character
address	Standard M68000 address: 1 longword (4 bytes)
charadr	Same as address
pointer	Same as address

Table 12.3-1. M68000 Data Types

12.4. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

12.5. Program Counter at Processor Exception

The location of an M68000 processor exception as reported by the early M68000 CPU's may be as much as four bytes beyond the code which actually produced the error. This may lead

MAINDEBUG to position incorrectly on certain M68000 processor exceptions. The MC68020 does not have this problem.

VM/SP CMS and VM/XA SP CMS MAINSAIL®

User's Guide

24 March 1989

xidak™

13. Introduction

This document describes the MAINSAIL implementations for VM/SP CMS and VM/XA SP CMS. VM/SP CMS is the IBM operating system for the IBM System/370 and compatible processors; VM/XA SP CMS is the IBM operating system for the IBM System/370 Extended Architecture and compatible processors. This document describes only VM/SP CMS-specific and VM/XA SP CMS-specific MAINSAIL features. It assumes that the reader is familiar with the "MAINSAIL Language Manual" and other machine-independent documentation.

13.1. Version

This version of the "VM/SP CMS and VM/XA SP CMS MAINSAIL User's Guide" is current as of Version 12.10 of MAINSAIL. It incorporates the "VM/CMS Version 5.10 Release Note" of October, 1982; the "VM/CMS Version 7.4 Release Note" of May, 1983; the "CMS MAINSAIL Release Note, Version 8" of January, 1984; the "VM/SP CMS MAINSAIL Release Note, Version 9" of February, 1985; the "VM/SP CMS MAINSAIL Release Note, Version 10" of March, 1986; and the "VM/SP CMS MAINSAIL User's Guide" and "VM/SP CMS MAINSAIL Release Note, Version 11" of July, 1987.

"VM/SP CMS and VM/XA SP CMS" are sometimes shortened to "CMS", or the two different names to "CMS" and "XCMS", respectively (when the context is clear).

In examples of interaction with CMS, the "." character terminates each prompt. This character appears on ASCII terminals when a terminal input request is made; it does not appear on 3270 terminals.

14. General Operation

14.1. Invoking MAINSAIL

To run a MAINSAIL program, first invoke MAINSAIL by typing "mainsa<eol>". MAINSAIL begins execution and types a herald identifying itself and the version of MAINSAIL being used. It then types "*" as a prompt and waits for input. The "*" prompt and possible responses to it are described in the MAINEX section of the "MAINSAIL Utilities User's Guide".

14.2. Intmod and Objmod File Names

On CMS and XCMS, the searchpaths for intmod and objmod file names in bootstraps distributed by XIDAK are:

```
SEARCHPATH *-int:* *2.*1int
SEARCHPATH *-obj:* *2.*1obj
```

Thus, for example, the default intmod and objmod file names for a module FOO compiled for CMS on CMS are "foo.cmsint" and "foo.cmsobj" and the default intmod and objmod file names for a module FOO compiled for XCMS on XCMS are "foo.xcmint" and "foo.xcmobj".

15. CONF, MAINSAIL Configurator

This chapter assumes familiarity with the CONF section of the "MAINSAIL Utilities User's Guide".

15.1. STACKSIZE Command

On both CMS and XCMS, the "STACKSIZE" command applies to the stack size for the initial coroutine ("MAINSAIL") as well as to the stack size for each new coroutine. The standard configuration file automatically defines a default value for this parameter.

15.2. VM/SP CMS-Specific Information

In addition to the machine-independent commands described in the CONF section of the "MAINSAIL Utilities User's Guide", the CMS implementation of MAINSAIL provides the additional command shown in Table 15.2-1.

CONF Command	Meaning
CMSBITS	Set CMS-specific attributes.

Table 15.2-1. VM/SP CMS-Specific CONF Command

The "CMSBITS" command allows the user to specify CMS-specific attributes. The values currently available are shown in Table 15.2-2. The "CMSBITS" values '2 and '4 are used together to govern which, if any, FORTRAN initialization routine is called from the boot if there are foreign modules. Figure 15.2-3 shows the relationship between these "CMSBITS" values and the FORTRAN initialization routine called.

The default CONF parameters are normally kept in the file "CMS CNF". Systemwide changes should always be made to this file.

The output of CONF is a System/370 assembly language file that is assembled to make a new bootstrap. Before the bootstrap can be assembled, the CMS command "GLOBAL MACLIB

Value	Meaning
'2	Interface to FORTRAN 77
'4	Do not initialize FORTRAN

Table 15.2-2. Currently Available CMSBITS Values

Foreign Module(s) Specified	CMSBITS Value '2 Specified	CMSBITS Value '4 Specified	FORTRAN IV Init Routine Called	FORTRAN 77 Init Routine Called
No	N/A	N/A	No	No
Yes	N/A	Yes	No	No
Yes	No	No	Yes	No
Yes	Yes	No	No	Yes

```
The initialization routines called are "IBCOM#" (for
FORTRAN IV) and "VFEIN#" (for FORTRAN 77).
```

Figure 15.2-3. Relationship between CMSBITS and FORTRAN Initialization Routines

DMSSP CMSLIB OSMACRO" must be given to define the standard CMS and OS macro simulation macro libraries.

Example 15.2-4 shows a sample session with CONF and how to assemble and link the resulting bootstrap. Default values are restored from the file "CMS CNF" and the bootstrap is written to the file "MAINSA ASSEMBLE". Note that the CMS "GLOBAL" statement is needed in order to assemble the bootstrap, and that it must be assembled with the "NOALIGN" option.

15.3. VM/XA SP CMS-Specific Information

In addition to the machine-independent commands described in the CONF section of the "MAINSAIL Utilities User's Guide", the XCMS implementation of MAINSAIL provides the additional commands shown in Table 15.3-1.

```
.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename mainsa.assemble<eol>
CONF: .<eol>
Bootstrap written in file FIX>MAINSA.ASSEMBLE
*.<eol>

    (Control returns to CMS)

.global maclib dmssp cmslib osmacro<eol>
.assemble mainsa (noalign<eol>
.load mainsa (clear<eol>
.genmod mainsa (all<eol>
.mainsa<eol>

    (MAINSAIL is invoked)
```

Example 15.2-4. CONF Session

CONF Command	Meaning
XCMSBITS	Set XCMS-specific attributes.

Table 15.3-1. VM/XA SP CMS-Specific CONF Commands

The "XCMSBITS" command allows the user to specify XCMS-specific attributes. The values currently available are shown in Table 15.3-2. The "XCMSBITS" values '1 and '2 are used together to govern which, if any, FORTRAN initialization routine is called from the boot if there are foreign modules. Figure 15.3-3 shows the relationship between these "XCMSBITS" values and the FORTRAN initialization routine called.

Value	Meaning
'1	Interface to FORTRAN 77
'2	Do not initialize FORTRAN

Table 15.3-2. Currently Available XCMSBITS Values

Foreign Module(s) Specified	XCMSBITS Value '1 Specified	XCMSBITS Value '2 Specified	FORTRAN IV Init Routine Called	FORTRAN 77 Init Routine Called
No	N/A	N/A	No	No
Yes	N/A	No	No	No
Yes	No	No	Yes	No
Yes	Yes	No	No	Yes

The initialization routines called are "IBCOM#" (for FORTRAN IV) and "VFEIN#" (for FORTRAN 77).

Figure 15.3-3. Relationship between XCMSBITS and FORTRAN Initialization Routines

The default CONF parameters are normally kept in the file "XCMS CNF". Systemwide changes should always be made to this file.

The output of CONF is a System/370 assembly language file that is assembled to make a new bootstrap. Before the bootstrap can be assembled, the CMS command "GLOBAL MACLIB DMSSP CMSLIB OSMACRO" must be given to define the standard CMS and OS macro simulation macro libraries.

Example 15.3-4 shows a sample session with CONF and how to assemble and link the resulting bootstrap. Default values are restored from the file "XCMS CNF" and the bootstrap is written to the file "MAINSA ASSEMBLE". Note that the CMS "GLOBAL" statement is needed in order to assemble the bootstrap, and that it must be assembled with the "NOALIGN" option. The amode and rmode options in the "GENMOD" command are necessary for the resulting bootstrap to run in 31-bit mode. These options may be omitted, in which case the resulting bootstrap will run in 24-bit mode.

```
.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file XCMS.CNF
CONF: .bootfilename mainsa.assemble<eol>
CONF: .<eol>
Bootstrap written in file FIX>MAINSA.ASSEMBLE
*.<eol>

    (Control returns to CMS)

.global maclib dmssp cmslib osmacro<eol>
.hasm mainsa (noalign<eol>
.load mainsa (clear<eol>
.genmod mainsa (amode 31 rmode 24<eol>
.mainsa<eol>

    (MAINSAIL is invoked)
```

Example 15.3-4. CONF Session

In order to make use of the 31-bit address capability of XCMS, the virtual machine size must be greater than 16M. Use the CMS "DEFINE STORAGE" command to set the virtual machine size.

It is important that the virtual machine size be larger than the value of the MAINSAIL "MAXMEMORYSIZE" CONF parameter. This is to ensure that MAINSAIL does not allocate all of the available memory and is important because certain CMS routines called by MAINSAIL may themselves allocate memory. If MAINSAIL has allocated all of the available memory, all CMS procedures that attempt to allocate memory will fail.

On XCMS, memory is allocated from high addresses to low addresses. Thus, the $memoryGrowsDown bit (bit 'H1) in the CONF "CONFIGURATIONBITS" parameter must be set.

16. System Calls

16.1. Introduction

The SVC System Calls provide access to the SVC 202 and SVC 203 instructions from MAINSAIL. Since VM/SP CMS system macros are invoked through the SVC 202 instruction, the SVC System Calls provide a way to invoke system macros from MAINSAIL. The SVC System Calls are available on CMS but not on XCMS.

The CMSCALL System Call provides access to the CMSCALL macro from MAINSAIL. The CMSCALL macro invokes a CMS command, CMS function, EXEC, or user MODULE. Refer to the "VM/XA SP CMS Application Program Development Reference" for more information about the CMSCALL macro. The CMSCALL System Call is available on XCMS but not on CMS.

The Diagnose Call provides access to the diagnose instruction from MAINSAIL. The Diagnose Call is available both on CMS and on XCMS.

The SVC System Calls, the CMSCALL System Call, and the Diagnose Call should be avoided if possible since the resulting code is CMS-dependent or XCMS-dependent. If it is necessary that an application program use these capabilities, the code that does so should be isolated so that it can easily be changed when the program is ported.

16.2. SVC System Calls

System calls of SVC types 202 and 203 are currently supported. CMS macros that expand to either an SVC 202 or an SVC 203 call can be invoked directly from MAINSAIL. Other SVC types are not presently supported but may be supported upon request.

The procedure headers for the SVC System Calls are shown in Figure 16.2-1. For the $svc202 call, parmString is the charadr of the tokenized parameter list and ePList is the charadr of the extended parameter list. The user is responsible for the information in these lists. value is the value to be placed in the high-order byte of GPR1. Two procedures, $svc203a and $svc203b, are provided for SVC 203 calls. If the SVC 203 call expects a parameter list, use $svc203a; if the SVC 203 call expects its parameters to be in registers, use $svc203b.

```
GENERIC
PROCEDURE   $svc        "$svc202,$svc203a,$svc203b";

INTEGER
PROCEDURE   $svc202     (CHARADR parmString;
                         OPTIONAL INTEGER value;
                         OPTIONAL CHARADR ePList);

INTEGER
PROCEDURE   $svc203a    (INTEGER SVCCode;
                         CHARADR parmString);

INTEGER
PROCEDURE   $svc203b    (INTEGER SVCCode;
                         OPTIONAL LONG BITS R0In;
                         OPTIONAL LONG BITS R1In;
                         OPTIONAL LONG BITS R0Out;
                         OPTIONAL LONG BITS R1Out);
```

Figure 16.2-1. Procedure Header for SVC System Calls

16.2.1. SVC System Call Example

This section describes how to use the SVC System Calls to invoke a CMS macro.

Suppose that a program needs to invoke the CMS macro TAPECTL. Refer to the "IBM VM/370 System Programmer's Guide" for more information about this macro. The format of the parameter list expected must first be determined. To do this, write a short assembly language program that invokes the macro, as shown in Figure 16.2.1-1. Assemble this program and examine the macro expansion in the listing file produced. The macro expansion for TAPECTL is shown in Figure 16.2.1-2. MAINSAIL must set up this parameter list before calling $svc. Table 16.2.1-3 shows one way to do this.

```
PRINT   GEN
BALR    15,0
USING   *,15
TAPECTL REW,TAP1,MODE=(9,6250)
BR      14
END
```

Figure 16.2.1-1. Assembly Language to Expand CMS Macro TAPECTL

```
            TAPECTL REW,TAP1,MODE=(9,6250)
+           CNOP    0,4
+           BAL     1,DMS0001A
+           DC      CL8'TAPEIO'
+           DC      CL8'REW'
+           DC      CL4'TAP1'
+           DC      BL1'11010011',AL3(0)
+           DC      2F'0'
+DMS0001A   SVC     202
```

Figure 16.2.1-2. Macro Expansion for CMS Macro TAPECTL

```
PROCEDURE callTapectl;
BEGIN
INTEGER i;
CHARADR c,svcBuffer;
c := svcBuffer := $newScratchChars(32);
cWrite(c,'T','A','P','E','I','O',' ',' ');
cWrite(c,'R','E','W',' ',' ',' ',' ',' ');
cWrite(c,'T','A','P','1');
cWrite(c,cvi('HD3),0,0,0);
cWrite(c,0,0,0,0,0,0,0,0);
IF i := $svc(svcBuffer) THEN
    errMsg("TAPECTL: REW function error ",cvs(i));
scratchDispose(svcBuffer) END;
```

Table 16.2.1-3. Code to Invoke Procedure $svc for CMS Macro TAPECTL

Some CMS system macros return data in registers R0 and/or R1. For example, the macro FSSTATE returns the address of the file status table (FST) in register R1. The macro expansion for FSSTATE is shown in Figure 16.2.1-4. After the SVC instruction is executed, register R1 is loaded with a pointer to the FST, located at offset 28 from the beginning of the parameter list passed to the supervisor call. MAINSAIL code that invokes this SVC 202 call will have set up a buffer and passed its address to the procedure $svc. After $svc has been invoked, the MAINSAIL statement:

```
a := cva(lbLoad(svcBufAdr,28));
```

can be used to obtain the address of the the FST.

```
              FSSTATE  (R3),FORM=E
+             CNOP     0,4
+             BAL      1,DMS0170A
+             DC       CL8'ESTATE'
+             DC       CL8' '
+             DC       CL8' '
+             DC       CL8' '
+             DC       CL2' '
+             DC       CL2' '
+             DC       AL4(0)
+DMS0170A     EQU      *
+             MVC      8(18,1),0(R3)
+             SVC      202
+             DC       AL4(*+8)
+             L        1,28(,1)
```

Figure 16.2.1-4. Macro Expansion for CMS Macro FSSTATE

16.3. CMSCALL System Call

The procedure header for the CMSCALL System Call is shown in Figure 16.3-1. pList and epList are the charadrs for the tokenized parameter list and extended parameter list, respectively. The user is responsible for the information in these lists. callTyp, uFlags, doCopy, doModify, and fence specify values for the CALLTYP, UFLAGS, COPY, MODIFY, and FENCE arguments in the CMSCALL macro.

```
INTEGER
PROCEDURE    $cmsCall    (CHARADR pList,epList;
                          INTEGER callTyp,uFlags;
                          BOOLEAN doCopy,doModify,fence);
```

Figure 16.3-1. Procedure Header for CMSCALL System Call

16.3.1. CMSCALL System Call Example

The example shown in Table 16.3.1-1 invokes CMSCALL to execute a CMS or CP command.

```
INTEGER PROCEDURE executeCmsOrCpCmd (STRING s);
BEGIN
INTEGER     i;
CHARADR     c;
OWN CHARADR scratchBuf;
# s contains the tokenized command string
IF NOT scratchBuf THEN
    scratchBuf := $newScratchChars($pageSize);
c := scratchBuf;
copy(cvc(s),c,length(s)); c := displace(c,length(s));
FOR i := 1 UPTO 8 DO cWrite(c,cvi('HFF));
RETURN(
    $cmsCall(
        scratchBuf,NULLCHARADR,0,0,TRUE,FALSE,TRUE));
END;



PROCEDURE doListCmd (OPTIONAL STRING diskName);
BEGIN
INTEGER i;
STRING  r,s;
s := "LISTFILE*       *       ";
r := IF NOT diskName THEN "A" EL cvu(r);
cWrite(s,cRead(r));
cWrite(s,IF r THEN cRead(r) EL ' ');
write(s,"      (       EXEC    ");

IF i := $executeCmsOrCpCmd(s) THENB
    ...
END;
```

Table 16.3.1-1. Sample Code That Invokes CMSCALL

16.4. Diagnose System Call

The Diagnose Call allows the execution of the diagnose instruction from MAINSAIL.

The procedure header for the Diagnose Call is shown in Figure 16.4-1. diagnoseCode is the code indicating the diagnose routine to be executed. RxIn, RxPlus1In, RyIn, RyPlus1In, and R15 are input parameters for the diagnose instruction. Register assignment is handled by the Diagnose Caller. Most diagnose instructions require only some of the parameters. Those not required can either be passed as '0L or be omitted if none of the following parameters is coded. RxOut, RyOut, RyPlus1Out, and completionCode are the diagnose output parameters. completionCode is the value of register 15 after execution of the diagnose instruction. The condition code is returned as the procedure result and should be checked only if the diagnose instruction sets the condition code.

```
INTEGER
PROCEDURE $diagnose (
    INTEGER                                diagnoseCode;
    OPTIONAL LONG BITS                     RxIn;
    OPTIONAL LONG BITS                     RxPlus1In;
    OPTIONAL LONG BITS                     RyIn;
    OPTIONAL LONG BITS                     RyPlus1In;
    OPTIONAL LONG BITS                     R15;
    PRODUCES OPTIONAL LONG BITS            RxOut;
    PRODUCES OPTIONAL LONG BITS            RyOut;
    PRODUCES OPTIONAL LONG BITS            RyPlus1Out;
    PRODUCES OPTIONAL LONG INTEGER         completionCode);
```

Figure 16.4-1. Header for Diagnose Call

16.4.1. Diagnose Call Example

Suppose that a program needs to execute the diagnose instruction for the pseudo timer (diagnose code 12). RxIn must be the address of a 32-byte area on a doubleword boundary where the date and time information is stored. Refer to the "IBM VM/370 System Programmer's Guide" for more information about this diagnose instruction.

Example 16.4.1-1 shows how the storage area may be created and how the diagnose call is invoked.

```
BEGIN
ADDRESS a;
DEFINE
    pseudoTimer     =   12;

# a must be on a double word boundary.  Allocate 5
# doublewords (the pseudo timer uses only 4) and then
# adjust the address by displacing by the length of a
# doubleword  - 1 and and then rounding down (clearing the
# low order bits).
IF NOT a THEN
    a := cva(cvlb(displace(newScratch(5 * 8),7)) CLR '7L);
# $diagnose returns date and time at address a as follows:
# mm/dd/yy
# hh:mm:ss
# virtual cpu time
# total cpu time
$diagnose(pseudoTimer,a);
END;
```

Example 16.4.1-1. Invoking the Pseudo Timer Diagnose Instruction

17. Foreign Language Interface

This chapter contains CMS-specific information for the MAINSAIL Foreign Language Interface (FLI). Refer to the "MAINSAIL Compiler User's Guide" for a general description of the FLI.

17.1. MAINSAIL to FORTRAN Compilers

There are two MAINSAIL-to-FORTRAN compilers; one interfaces to FORTRAN IV, the other to FORTRAN 77 (VS FORTRAN). Both forms of FORTRAN are available on CMS; only FORTRAN 77 is available on XCMS. These compilers may also be used to interface to assembly language that follows the FORTRAN calling convention.

The default output file name for the MAINSAIL-to-FORTRAN compilers is "FIX(80)><module name>.assembler" when compiling on CMS and "<module name>.asm" when cross-compiling from another operating system.

17.1.1. Data Types

FORTRAN IV parameters are mapped into MAINSAIL parameters as shown in Table 17.1.1-1.

FORTRAN	MAINSAIL	Representation Passed
LOGICAL*1	BOOLEAN	byte
INTEGER*2	INTEGER or BITS	halfword
INTEGER*4	LONG INTEGER, LONG BITS	fullword
REAL*4	REAL	fullword
REAL*8	LONG REAL	doubleword
LOGICAL*1 array	STRING	address of first character
array	ARRAY or ADDRESS	address of first element

Table 17.1.1-1. Mapping FORTRAN IV Data Types to MAINSAIL Data Types When Using the FCC

FORTRAN 77 parameters are mapped into MAINSAIL parameters as shown in Table 17.1.1-2.

FORTRAN	MAINSAIL	Representation Passed
LOGICAL*1	BOOLEAN	byte
INTEGER*2	INTEGER or BITS	halfword
INTEGER*4	LONG INTEGER, LONG BITS	fullword
REAL*4	REAL	fullword
REAL*8	LONG REAL	doubleword
LOGICAL*1 array	CHARADR	address of first character
CHARACTER*n	STRING	address of first character, length of string
array	ARRAY or ADDRESS	address of first element

Table 17.1.1-2. Mapping FORTRAN 77 Data Types to MAINSAIL Data Types When Using the FCC

Pointer variables may not be passed to FORTRAN. Modifies and produces parameters of the types string and array are also not allowed.

Uses arguments are passed to FORTRAN by value; modifies and produces arguments are passed by value result. Produces arguments are initialized to Zero.

17.2. Foreign Language Interface Example, MAINSAIL to FORTRAN IV

Suppose that the MAINSAIL module CALFIV is to call the FORTRAN IV subroutine PROC1. Figures 17.2-1, 17.2-2, and 17.2-3 show the module CALFIV, the MAINSAIL FLI module TOFIV, and the FORTRAN IV subroutine PROC1, respectively. Example 17.2-4 shows how to compile and run CALFIV.

```
MAINSAIL Module CALFIV (in file CALFIV MSL):

BEGIN "calFiv"

MODULE toFiv (
    PROCEDURE proc1 (
        PRODUCES LONG INTEGER   tt;
        LONG INTEGER            uu,vv;
        PRODUCES BOOLEAN        bo;
        REAL ARRAY(1 TO 2)      rAry;
        STRING                  s;
        PRODUCES INTEGER        sLen);
    );

INITIAL PROCEDURE;
BEGIN
INTEGER             sLen;
LONG INTEGER        tt;
STRING              s;
REAL ARRAY(1 TO 2)  rAry;
# FORTRAN IV subroutine PROC1 does the following:
#    (1) adds uu and vv and returns the result in tt
#    (2) sets bo to be TRUE
#    (3) swaps the 1st and 2nd elements of rAry
#    (4) computes the length of s and returns the
#        result in sLen
new(rAry); rAry[1] := 1.0; rAry[2] := 2.0;
s := "This is a string" & cvcs(0);
proc1(tt,1L,2L,bo,rAry,s,sLen);
write(logFile,
    "tt: ",tt,eol &
    "bo: ",IF bo THEN "TRUE" EL "FALSE" & eol &
    "rAry[1]: ",rAry[1], " rAry[2]: ",rAry[2],eol &
    "s: ",s,eol &
    "sLen: ",sLen,eol);
END;

END "calFiv"
```

Figure 17.2-1. MAINSAIL Module CALFIV That Calls FORTRAN IV Subroutine PROC1

```
MAINSAIL Module TOFIV (in file TOFIV MSL):

BEGIN "toFiv"

MODULE toFiv (
    PROCEDURE procl (
        PRODUCES LONG INTEGER   tt;
        LONG INTEGER            uu,vv;
        PRODUCES BOOLEAN        bo;
        REAL ARRAY(1 TO 2)      rAry;
        STRING                  s;
        PRODUCES INTEGER        sLen);
    );

PROCEDURE procl (
    PRODUCES LONG INTEGER   tt;
    LONG INTEGER            uu,vv;
    BOOLEAN                 bo;
    REAL ARRAY(1 TO 2)      rAry;
    STRING                  s;
    PRODUCES INTEGER        sLen);;

END "toFiv"
```

Figure 17.2-2. MAINSAIL Foreign Language Interface Module TOFIV

```
FORTRAN IV subroutine PROC1 (in file FIV FORTRAN):

      SUBROUTINE PROC1 (TT,UU,VV,RARY,S,SLEN)
      INTEGER*2 SLEN
      INTEGER*4 TT,UU,VV
      LOGICAL*1 BO
      REAL*4 RARY(2),TEMP
      LOGICAL*1 S(512)
      TT = UU + VV
      BO = .TRUE.
      TEMP = RARY(1)
      RARY(1) = RARY(2)
      RARY(2) = TEMP
      SLEN = 0
100   CONTINUE
      IF (.NOT. S(SLEN+1)) GOTO 150
      SLEN = SLEN + 1
      GOTO 100
150   CONTINUE
      RETURN
      END
```

Figure 17.2-3. FORTRAN IV Subroutine PROC1 Called by MAINSAIL

```
(1) Compile CALFIV with the MAINSAIL CMS compiler.
    Compile TOFIV with the FCC to FORTRAN IV (by
    specifying the compiler subcommand "fli tf").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.compil<eol>
```

Example 17.2-4. MAINSAIL to FORTRAN IV Example (continued)

```
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): .calfiv.msl<eol>

calfiv.msl 1 ...

compile (? for help): .tofiv.msl,<eol>
>fli tf<eol>
><eol>

tofiv.msl 1 ...

compile (? for help): .<eol>
*.<eol>
(control returns to CMS)

(2) Make a new MAINSAIL bootstrap that declares TOFIV to
    be a foreign module.  NOTE:  The foreign module names
    are converted to uppercase, as the CMS assembler
    accepts only uppercase characters.

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename fcc<eol>
CONF: .foreignmodules TOFIV<eol>
CONF: .<eol>
Bootstrap written in file FIX>fcc.assemble
*.<eol>
(control returns to CMS)

(3) Compile the FORTRAN IV code with FORTRAN IV
    compiler.
```

Example 17.2-4. MAINSAIL to FORTRAN IV Example (continued)

(4) Assemble and link the new MAINSAIL bootstrap. The "global maclib" command makes available all macro libraries required to assemble the MAINSAIL bootstrap. NOTE: the MAINSAIL bootstrap and all FLI interface code MUST be assembled with the (NOALIGN option. The "global txtlib" command makes available the FORTRAN IV initialization routine called by MAINSAIL.

```
.global maclib dmssp cmslib osmacro<eol>
.assemble fcc (noalign<eol>
.assemble tofiv (noalign<eol>
.global txtlib fortmod2<eol>
.load fcc tofiv fiv (clear reset msent<eol>
.genmod fcc (all<eol>
```

(5) Run the new executable MAINSAIL bootstrap and call the foreign procedure.

```
.fcc<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.calfiv<eol>
```

Example 17.2-4. MAINSAIL to FORTRAN IV Example (end)

17.3. Foreign Language Interface Example, MAINSAIL to FORTRAN 77

Suppose that the MAINSAIL module CALF77 is to call the FORTRAN 77 subroutine PROC1. Figures 17.3-1, 17.3-2, and 17.3-3 show the module CALF77, the MAINSAIL FLI module TOF77, and the FORTRAN 77 subroutine PROC1, respectively. Example 17.3-4 shows how to compile and run CALF77.

```
MAINSAIL Module CALF77 (in file CALF77 MSL):

BEGIN "calF77"

MODULE toF77 (
    PROCEDURE proc1 (
        PRODUCES LONG INTEGER   tt;
        LONG INTEGER            uu,vv;
        PRODUCES BOOLEAN        bo;
        REAL ARRAY(1 TO 2)      rAry;
        STRING                  s;
        PRODUCES INTEGER        sLen);
    );

INITIAL PROCEDURE;
BEGIN
INTEGER             sLen;
LONG INTEGER        tt;
STRING              s;
REAL ARRAY(1 TO 2)  rAry;
# FORTRAN 77 subroutine proc1 does the following:
#   (1) adds uu and vv and returns the result in tt
#   (2) sets bo to be TRUE
#   (3) swaps the 1st and 2nd elements of rAry
#   (4) computes the length of s and returns the
#       result in sLen
new(rAry); rAry[1] := 1.0; rAry[2] := 2.0;
s := "This is a string" & cvcs(0);
proc1(tt,1L,2L,rAry,s,sLen);
write(logFile,
    "tt: ",tt,eol &
    "bo: ",IF bo THEN "TRUE" EL "FALSE" & eol &
    "rAry[1]: ",rAry[1], " rAry[2]: ",rAry[2],eol &
    "s: ",s,eol &
    "sLen: ",sLen,eol);
END;

END "calF77"
```

Figure 17.3-1. MAINSAIL Module CALF77 That Calls FORTRAN 77 Subroutine PROC1

```
MAINSAIL Module TOF77 (in file TOF77 MSL):

BEGIN "toF77"

MODULE toF77 (
    PROCEDURE proc1 (
        PRODUCES LONG INTEGER   tt;
        LONG INTEGER            uu,vv;
        PRODUCES BOOLEAN        bo;
        REAL ARRAY(1 TO 2)      rAry;
        STRING                  s;
        PRODUCES INTEGER        sLen);
    );

PROCEDURE proc1 (
    PRODUCES LONG INTEGER   tt;
    LONG INTEGER            uu,vv;
    BOOLEAN                 bo;
    REAL ARRAY(1 TO 2)      rAry;
    STRING                  s;
    PRODUCES INTEGER        sLen);;

END "toF77"
```

Figure 17.3-2. MAINSAIL Foreign Language Interface Module TOF77

```
FORTRAN 77 subroutine PROC1 (in file F77 FORTRAN):

      SUBROUTINE PROC1 (TT,UU,VV,RARY,S,SLEN)
      INTEGER*2 SLEN
      INTEGER*4 TT,UU,VV
      LOGICAL*1 BO
      REAL*4 RARY(2),TEMP
      CHARACTER S*(*),ZERO
      TT = UU + VV
      BO = .TRUE.
      TEMP = RARY(1)
      RARY(1) = RARY(2)
      RARY(2) = TEMP
      ZERO = CHAR(0)
      SLEN = 0
100   CONTINUE
      IF (S(SLEN+1:SLEN+1) .EQ. ZERO) GOTO 150
      SLEN = SLEN + 1
      GOTO 100
150   CONTINUE
      RETURN
      END
```

Figure 17.3-3. FORTRAN 77 Subroutine PROC1 Called by MAINSAIL

```
(1) Compile CALF77 with the MAINSAIL CMS compiler.
    Compile TOF77 with the FCC to FORTRAN 77 (by
    specifying the compiler subcommand "fli t7").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.compil<eol>
```

Example 17.3-4. MAINSAIL to FORTRAN 77 Example (continued)

```
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): .calf77.msl<eol>

calf77.msl 1 ...

compile (? for help): .tof77.msl,<eol>
>fli t7<eol>
><eol>

tof77.msl 1 ...

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

(control returns to CMS)

(2) Make a new MAINSAIL bootstrap that declares TOF77 to
    be a foreign module and declares that FORTRAN 77 code
    is to be called (by specifying "cmsbits '2").
    The CMSBITS command is necessary in order that
    MAINSAIL invoke the correct FORTRAN initialization
    routine.  NOTE:  The foreign module names are
    converted to uppercase, as the CMS assembler accepts
    only uppercase characters.
```

Example 17.3-4. MAINSAIL to FORTRAN 77 Example (continued)

```
.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename fcc<eol>
CONF: .foreignmodules TOF77<eol>
CONF: .cmsbits '2<eol>
CONF: .<eol>
Bootstrap written in file FIX>fcc.assemble
*.<eol>

    (control returns to CMS)

(3) Compile the FORTRAN 77 code with the FORTRAN 77
    compiler.

(4) Assemble and link the new MAINSAIL bootstrap.  The
    "global maclib" command makes available all macro
    libraries required to assemble the MAINSAIL
    bootstrap.  NOTE:  The MAINSAIL bootstrap and all FLI
    interface code MUST be assembled with the (NOALIGN
    option.  The "global txtlib" command makes available
    the FORTRAN 77 initialization routine called by
    MAINSAIL.  The "global loadlib" command makes
    available FORTRAN 77 routines required at runtime.

.global maclib dmssp cmslib osmacro<eol>
.assemble fcc (noalign<eol>
.assemble tof77 (noalign<eol>
.global txtlib vsf2fort<eol>
.global loadlib vsf2load<eol>
.load fcc tof77 f77 (clear reset msent<eol>

    On CMS, do:

.genmod fcc (all<eol>
```

Example 17.3-4. MAINSAIL to FORTRAN 77 Example (continued)

```
    On XCMS, do:

.genmod fcc (amode 31 rmode 24<eol>

(5) Run the new executable MAINSAIL bootstrap and call
    the foreign procedure.

.fcc<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.calf77<eol>
```

Example 17.3-4. MAINSAIL to FORTRAN 77 Example (end)

17.4. FORTRAN to MAINSAIL Compilers

There are two FORTRAN-to-MAINSAIL compilers; one interfaces to FORTRAN IV, the other to FORTRAN 77 (VS FORTRAN). Both forms of FORTRAN are available on CMS; only FORTRAN 77 is available on XCMS. These compilers may also be used to interface to assembly language that follows the FORTRAN calling conventions.

The default output file name for the FORTRAN-to-MAINSAIL compilers is "FIX(80)><module name>.assembler" when compiling on CMS and "<module name>.asm" when cross-compiling from another operating system.

The $foreignCodeStartsExecution option has been implemented for both FORTRAN IV to MAINSAIL and FORTRAN 77 to MAINSAIL on CMS, and FORTRAN 77 to MAINSAIL on XCMS.

17.4.1. Data Types

FORTRAN IV parameters are mapped into MAINSAIL parameters as shown in Table 17.4.1-1.

MAINSAIL	FORTRAN	Representation Passed
BOOLEAN	LOGICAL*1	byte
INTEGER or BITS	INTEGER*2	halfword
LONG INTEGER, LONG BITS	INTEGER*4	fullword
REAL	REAL*4	fullword
LONG REAL	REAL*8	doubleword
CHARADR	LOGICAL*1 array	address of first character
ADDRESS	array	address of first element

Table 17.4.1-1. Mapping FORTRAN IV Data Types to MAINSAIL Data Types When Using the MEC

FORTRAN 77 parameters are mapped into MAINSAIL parameters as shown in Table 17.4.1-2.

MAINSAIL	FORTRAN	Representation Passed
BOOLEAN	LOGICAL*1	byte
INTEGER or BITS	INTEGER*2	halfword
LONG INTEGER, LONG BITS	INTEGER*4	fullword
REAL	REAL*4	fullword
LONG REAL	REAL*8	doubleword
CHARADR	LOGICAL*1 array	address of first character
ADDRESS	array	address of first element
STRING	CHARACTER*n	address of first character, length of string

Table 17.4.1-2. Mapping FORTRAN 77 Data Types to MAINSAIL Data Types When Using the MEC

Pointer and array variables may not be passed from FORTRAN to MAINSAIL. String parameters may not be passed from FORTRAN IV to MAINSAIL but may be passed from FORTRAN 77 to MAINSAIL. Modifies and produces parameters of the types pointer, string, charadr, and address are also not allowed and the MAINSAIL procedure cannot produce a pointer, string, or array.

17.5. Foreign Language Interface Example, FORTRAN IV to MAINSAIL

Suppose that the FORTRAN IV subroutine CALLMS is to call the MAINSAIL procedure proc1. Figures 17.5-1, 17.5-2, 17.5-3, and 17.5-4 show the FORTRAN IV subroutine CALLMS, the MAINSAIL module MSMOD that contains procedure proc1, the MAINSAIL FLI module TOFIV, and the MAINSAIL module CALFIV that calls the FORTRAN IV subroutine CALLMS. Example 17.5-5 shows how to compile and run CALLMS.

```
FORTRAN IV code for CALLMS (in file CALLMS FORTRAN):

      SUBROUTINE CALLMS
      INTEGER*4 LI1,LI2,RESULT,PROC1
      LOGICAL*1 BO
      EXTERNAL PROC1
C
C MAINSAIL PROC1 DOES THE FOLLOWING:
C   (1) ADDS ITS FIRST TWO PARAMETERS AND RETURNS
C       THE RESULT
C   (2) SETS BO TO BE .TRUE.
C
      BO = .FALSE.
      LI1 = 1
      LI2 = 2
      RESULT = PROC1(LI1,LI2,BO)
      WRITE(6,1000) RESULT
      IF (RESULT .NE. 3) WRITE(6,1010)
      IF (.NOT. BO) WRITE(6,1020)
1000  FORMAT(' Result is ',I6)
1010  FORMAT(' FAILURE: Incorrect result')
1020  FORMAT(' FAILURE: BO should be .TRUE.')
      RETURN
      END
```

Figure 17.5-1. FORTRAN IV Subroutine CALLMS That Calls MAINSAIL Procedure proc1

```
MAINSAIL Module MSMOD (in file MSMOD MSL):

BEGIN "msMod"

MODULE msMod (
    LONG INTEGER PROCEDURE proc1 (
        LONG INTEGER          li1,li2;
        PRODUCES BOOLEAN      bo);
    );

LONG INTEGER PROCEDURE proc1 (
    LONG INTEGER          li1,li2;
    PRODUCES BOOLEAN      bo);
BEGIN
bo := TRUE;
RETURN(li1 + li2);
END;

END "msMod"
```

Figure 17.5-2. MAINSAIL Module MSMOD Called by FORTRAN IV Subroutine CALLMS

```
MAINSAIL Module TOFIV (in file TOFIV MSL):

BEGIN "toFiv"

MODULE toFiv (PROCEDURE callMs);

PROCEDURE callMs;;

END "toFiv"
```

Figure 17.5-3. MAINSAIL Foreign Language Interface Module TOFIV

```
MAINSAIL Module CALFIV (in file CALFIV MSL):

BEGIN "calFiv"

MODULE toFiv (PROCEDURE callMs);

INITIAL PROCEDURE;
callMs;

END "calFiv"
```

Figure 17.5-4. MAINSAIL Module CALFIV That Calls FORTRAN IV Subroutine CALLMS

```
(1) Compile MSMOD and CALFIV with the MAINSAIL CMS
    compiler.  Compile MSMOD with the MEC from FORTRAN IV
    (by specifying the compiler subcommand "fli ff").
    Compile TOFIV with the FCC to FORTRAN IV (by
    specifying the compiler subcommand "fli tf").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): .msmod.msl<eol>

msmod.msl 1 ...

compile (? for help): .calfiv.msl<eol>

calfiv.msl 1 ...
```

Example 17.5-5. FORTRAN IV to MAINSAIL Example (continued)

```
compile (? for help): .msmod.msl,<eol>
>.fli ff<eol>
>.<eol>

msmod.msl 1 ...

compile (? for help): .tofiv.msl,<eol>
>.fli tf<eol>
>.<eol>

tofiv.msl 1 ...

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

 (control returns to CMS)

 (2) Make a new MAINSAIL bootstrap that declares TOFIV to
     be a foreign module.  NOTE:  The foreign module names
     are converted to uppercase, as the CMS assembler
     accepts only uppercase characters.

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename mec<eol>
CONF: .foreignmodules TOFIV<eol>
CONF: .<eol>
Bootstrap written in file FIX>mec.assemble
*.<eol>

 (control returns to CMS)

 (3) Compile the FORTRAN IV code with FORTRAN IV
     compiler.
```

Example 17.5-5. FORTRAN IV to MAINSAIL Example (continued)

```
(4) Assemble and link the new MAINSAIL bootstrap.  The
    "global maclib" command makes available all macro
    libraries required to assemble the MAINSAIL
    bootstrap.  NOTE:  The MAINSAIL bootstrap and all FLI
    interface code MUST be assembled with the (NOALIGN
    option.  The "global txtlib" command makes available
    the FORTRAN IV initialization routine called by
    MAINSAIL.

.global maclib dmssp cmslib osmacro<eol>
.assemble mec (noalign<eol>
.assemble msmod (noalign<eol>
.assemble tofiv (noalign<eol>
.global txtlib fortmod2<eol>
.load mec msmod tofiv callms (clear reset msent<eol>
.genmod mec (all<eol>

(5) Run the new executable MAINSAIL bootstrap and call
    the foreign procedure.

.mec<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.calfiv<eol>
Result is        3
*.<eol>
```

Example 17.5-5. FORTRAN IV to MAINSAIL Example (end)

17.6. Foreign Language Interface Example, FORTRAN IV to MAINSAIL, $foreignCodeStartsExecution

Suppose that the FORTRAN IV program CALLMS is to call the MAINSAIL procedure proc1 and that execution starts in the FORTRAN IV code. The FORTRAN IV program CALLMS is the same as that shown in Figure 17.5-1 except that the SUBROUTINE statement is omitted. The MAINSAIL module MSMOD that contains the procedure proc1 is the same as that shown in Figure 17.5-2. Example 17.6-1 shows how to compile and run CALLMS.

```
(1) Compile MSMOD with the MAINSAIL CMS compiler.  Compile
    MSMOD with the MEC from FORTRAN IV (by specifying the
    compiler subcommand "fli ff").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): .msmod.msl<eol>

msmod.msl 1 ...

compile (? for help): .msmod.msl,<eol>
>.fli ff<eol>
>.<eol>

msmod.msl 1 ...

compile (? for help): .<eol>
*.<eol>
```

Example 17.6-1. FORTRAN IV to MAINSAIL Example, $foreignCodeStartsExecution (continued)

```
(control returns to CMS)

(2) Make a new MAINSAIL bootstrap that declares that
    execution starts in foreign code (by specifying
    "configurationbits '10").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename mec<eol>
CONF: .configurationbits '10<eol>
CONF: .<eol>
Bootstrap written in file FIX>mec.assemble
*.<eol>

(control returns to CMS)

(3) Compile the FORTRAN IV code with the FORTRAN IV
    compiler.

(4) Assemble and link the new MAINSAIL bootstrap.  The
    "global maclib" command makes available all macro
    libraries required to assemble the MAINSAIL
    bootstrap.  NOTE:  The MAINSAIL bootstrap and all FLI
    interface code MUST be assembled with the (NOALIGN
    option.  The "global txtlib" command makes available
    the FORTRAN IV initialization routine called by
    MAINSAIL.

.global maclib dmssp cmslib osmacro<eol>
.assemble mec (noalign<eol>
.assemble msmod (noalign<eol>
.global txtlib fortmod2<eol>
.load mec msmod callms
.genmod mec (all<eol>
```

Example 17.6-1. FORTRAN IV to MAINSAIL Example, $foreignCodeStartsExecution (continued)

```
(5) Run the executable file "mec".

.mec<eol>
Result is       3

(control returns to CMS)
```

Example 17.6-1. FORTRAN IV to MAINSAIL Example, $foreignCodeStartsExecution (end)

17.7. Foreign Language Interface Example, FORTRAN 77 to MAINSAIL

Suppose that the FORTRAN 77 subroutine CALLMS is to call the MAINSAIL procedure proc1. Figures 17.7-1, 17.7-2, 17.7-3, and 17.7-4 show the FORTRAN 77 subroutine CALLMS, the MAINSAIL module MSMOD that contains procedure proc1, the MAINSAIL FLI module TOF77, and the MAINSAIL module CALF77 that calls the FORTRAN 77 subroutine CALLMS. Example 17.7-5 shows how to compile and run CALLMS.

```
FORTRAN subroutine CALLMS (in file CALLMS FORTRAN):

      SUBROUTINE CALLMS
      INTEGER*4 LI1,LI2,RESULT,PROC1
      LOGICAL*1 BO
      CHARACTER*24 S1,S2
      EXTERNAL PROC1
C
C MAINSAIL PROC1 DOES THE FOLLOWING:
C   (1) ADDS ITS FIRST TWO PARAMETERS AND RETURNS
C       THE RESULT
C   (2) COMPARES S1 AND S2; SETS TO .TRUE. IF THE
C       STRINGS ARE EQUAL AND TO .FALSE. IF THE
C       STRINGS ARE NOT EQUAL
C
      LI1 = 1
      LI2 = 2
      BO = .TRUE.
      S1 = 'First string to compare '
      S2 = 'Second string to compare'
      RESULT = PROC1(LI1,LI2,S1,S2,BO)
      WRITE(6,1000) RESULT
      IF (RESULT .NE. 3) WRITE(6,1010)
      IF (BO .EQV. .TRUE.) WRITE(6,1020)
1000  FORMAT(' Result is ',I6)
1010  FORMAT(' FAILURE: Incorrect result')
1020  FORMAT(' FAILURE: String compare failed')
      RETURN
      END
```

Figure 17.7-1. FORTRAN 77 Subroutine CALLMS That Calls MAINSAIL Procedure proc1

```
MAINSAIL Module MSMOD (in file MSMOD MSL):

BEGIN "msMod"

MODULE msMod (
    LONG INTEGER PROCEDURE proc1 (
        LONG INTEGER         li1,li2;
        STRING               s1,s2;
        PRODUCES BOOLEAN     bo);
    );

LONG INTEGER PROCEDURE proc1 (
    LONG INTEGER         li1,li2;
    STRING               s1,s2;
    PRODUCES BOOLEAN     bo);
BEGIN
bo := IF s1 NEQ s2 THEN FALSE EL TRUE;
RETURN(li1 + li2);
END;

END "msMod"
```

Figure 17.7-2. MAINSAIL Module MSMOD Called by FORTRAN 77 Subroutine CALLMS

```
MAINSAIL Module TOF77 (in file TOF77 MSL):

BEGIN "toF77"

MODULE toF77 (PROCEDURE callMs);

PROCEDURE callMs;;

END "toF77"
```

Figure 17.7-3. MAINSAIL Foreign Language Interface Module TOF77

```
MAINSAIL Module CALF77 (in file CALF77 MSL):

BEGIN "calF77"

MODULE toF77 (PROCEDURE callMs);

INITIAL PROCEDURE;
callMs;

END "calF77"
```

Figure 17.7-4. MAINSAIL Module CALF77 That Calls FORTRAN 77 Subroutine CALLMS

```
(1) Compile MSMOD and CALF77 with the MAINSAIL CMS
    compiler.  Compile MSMOD with the MEC from FORTRAN 77
    (by specifying the compiler subcommand "fli f7").
    Compile TOFIV with the FCC to FORTRAN 77 (by
    specifying the compiler subcommand "fli t7").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): .msmod.msl<eol>

msmod.msl 1 ...

compile (? for help): .calf77.msl<eol>

calf77.msl 1 ...
```

Example 17.7-5. FORTRAN 77 to MAINSAIL Example (continued)

```
compile (? for help): .msmod.msl,<eol>
>.fli f7<eol>
>.<eol>

msmod.msl 1 ...

compile (? for help): .tof77.msl,<eol>
>.fli t7<eol>
>.<eol>

tof77.msl 1 ...

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

(control returns to CMS)

(2) Make a new MAINSAIL bootstrap that declares TOF77 to
    be a foreign module and declares that FORTRAN 77
    code is to be called (by specifying "cmsbits '2").
    The CMSBITS command is necessary in order that
    MAINSAIL invoke the correct FORTRAN initialization
    routine.  NOTE:  The foreign module names are
    converted to uppercase, as the CMS assembler accepts
    only uppercase characters.

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename mec<eol>
CONF: .foreignmodules TOF77<eol>
CONF: .cmsbits '2<eol>
CONF: .<eol>
Bootstrap written in file FIX>mec.assemble
*.<eol>
```

Example 17.7-5. FORTRAN 77 to MAINSAIL Example (continued)

```
(control returns to CMS)

(3) Compile the FORTRAN 77 code with the FORTRAN 77
    compiler.

(4) Assemble and link the new MAINSAIL bootstrap.  The
    "global maclib" command makes available all macro
    libraries required to assemble the MAINSAIL
    bootstrap.  NOTE:  The MAINSAIL bootstrap and all FLI
    interface code MUST be assembled with the (NOALIGN
    option.  The "global txtlib" command makes available
    the FORTRAN 77 initialization routine called by
    MAINSAIL.  The "global loadlib" command makes
    available FORTRAN 77 routines required at runtime.

.global maclib dmssp cmslib osmacro<eol>
.assemble mec (noalign<eol>
.assemble msmod (noalign<eol>
.assemble tof77 (noalign<eol>
.global txtlib vsf2fort<eol>
.global loadlib vsf2load<eol>
.load mec msmod tof77 callms (clear reset msent<eol>

    On CMS, do:

.genmod mec (all<eol>

    On XCMS, do:

.genmod mec (amode 31 rmode 24<eol>

(5) Run the new executable MAINSAIL bootstrap and call
    the foreign procedure.
```

Example 17.7-5. FORTRAN 77 to MAINSAIL Example (continued)

```
.mec<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.calf77<eol>
Result is       3
*.<eol>
```

Example 17.7-5. FORTRAN 77 to MAINSAIL Example (end)

17.8. Foreign Language Interface Example, FORTRAN 77 to MAINSAIL, $foreignCodeStartsExecution

Suppose that the FORTRAN 77 program CALLMS is to call the MAINSAIL procedure proc1 and that execution starts in the FORTRAN 77 code. The FORTRAN 77 program CALLMS is the same as that shown in Figure 17.7-1 except that the SUBROUTINE statement is omitted. The MAINSAIL module MSMOD that contains procedure proc1 is the same as that shown in Figure 17.7-2. Example 17.8-1 shows how to compile and run CALLMS.

```
(1) Compile MSMOD with the MAINSAIL CMS compiler.  Compile
    MSMOD with the MEC from FORTRAN 77 (by specifying the
    compiler subcommand "fli f7").

.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): .msmod.msl<eol>

msmod.msl 1 ...

compile (? for help): .msmod.msl,<eol>
>.fli f7<eol>
>.<eol>

msmod.msl 1 ...

compile (? for help): .<eol>
*.<eol>
```

Example 17.8-1. FORTRAN 77 to MAINSAIL Example, $foreignCodeStartsExecution (continued)

```
(control returns to CMS)
```

(2) Make a new MAINSAIL bootstrap that declares that execution starts in foreign code (by specifying "configurationbits '10") and declares that FORTRAN 77 code is being used (by specifying "cmsbits '2"). The "CMSBITS" command is necessary in order that MAINSAIL invoke the correct FORTRAN initialization routine.

```
.mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*.conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file CMS.CNF
CONF: .bootfilename mec<eol>
CONF: .configurationbits '10<eol>
CONF: .cmsbits '2<eol>
CONF: .<eol>
Bootstrap written in file FIX>mec.assemble
*.<eol>

(control returns to CMS)
```

(3) Compile the FORTRAN 77 code with the FORTRAN 77 compiler.

(4) Assemble and link the new MAINSAIL bootstrap. The "global maclib" command makes available all macro libraries required to assemble the MAINSAIL bootstrap. NOTE: The MAINSAIL bootstrap and all FLI interface code MUST be assembled with the (NOALIGN option. The "global txtlib" command makes available the FORTRAN 77 initialization routine called by MAINSAIL. The "global loadlib" command makes available FORTRAN 77 routines required at runtime.

Example 17.8-1. FORTRAN 77 to MAINSAIL Example, $foreignCodeStartsExecution (continued)

```
.global maclib dmssp cmslib osmacro<eol>
.assemble mec (noalign<eol>
.assemble msmod (noalign<eol>
.global txtlib vsf2fort<eol>
.global loadlib vsf2load<eol>
.load mec msmod callms

    On CMS, do:

.genmod mec (all<eol>

    On XCMS, do:

.genmod mec (amode 31 rmode 24<eol>

(5) Run the executable file "mec".

.mec<eol>
Result is        3

(control returns to CMS)
```

Example 17.8-1. FORTRAN 77 to MAINSAIL Example, $foreignCodeStartsExecution (end)

17.9. Caveats

MAINSAIL stores arrays in row-major form whereas FORTRAN stores them in column-major form; the user must take this into consideration for multidimensional arrays.

If FORTRAN violates the boundaries of a MAINSAIL array parameter, or if it changes the characters of a string parameter, the results are unpredictable.

17.10. FORTRAN IV and FORTRAN 77 Compatibility

Within one MAINSAIL session, either FORTRAN IV or FORTRAN 77 routines can be invoked, but not both.

18. Program Exceptions

18.1. Introduction

Under VM/SP CMS and VM/XA SP CMS, a user program may enable or disable interrupts resulting from certain program exceptions. Program exceptions for which the interrupt mechanism may be enabled or disabled are fixed point overflow, decimal overflow, exponent underflow, and significance.

All program exceptions that cause an interrupt and for which the interrupt is enabled are trapped by the MAINSAIL error handling mechanism and reported to the user. The MAINSAIL bootstrap automatically enables interrupts caused by fixed point overflow, decimal overflow, and exponent underflow. The interrupt caused by significance is not enabled, since the code generated for "cvr" and "cvlr" can potentially cause this program exception.

18.2. $spm

A MAINSAIL program running under CMS or XCMS may directly specify whether or not a program exception should cause an interrupt by invoking the procedure $spm. The procedure header for $spm is shown in Figure 18.2-1. $spm issues an SPM (Set Program Mask) instruction with the data passed in the parameter programMask. This instruction sets both the condition code and the program mask bits of the current PSW (Program Status Word). Bits 2 and 3 (0 is the high-order bit) must contain the condition code and bits 4 through 7 the program mask. Figure 18.2-2 shows the correspondence between the program mask bits and interrupts. A "0" indicates that the interrupt is disabled; a "1" indicates that it is enabled. Refer to the "System/370 Principles of Operation" for more information.

```
PROCEDURE $spm (LONG BITS programMask);
```

Figure 18.2-1. Procedure Header for Procedure $spm

Bit	Interrupt
4	Fixed point overflow
5	Decimal overflow
6	Exponent underflow
7	Significance

Figure 18.2-2. Correspondence between Program Mask Bits and Interrupts

18.3. Example of Calling $spm

Example 18.3-1 shows how to call $spm. The $spm call shown disables the fixed point overflow program exception. Decimal overflow, exponent underflow, and significance remain enabled; if any of these occurs, an interrupt is generated and an error reported by the MAINSAIL error handling mechanism.

```
PROCEDURE disableFixedPointOverflow;
$spm('H07000000L);
```

Example 18.3-1. Calling $spm

19. File System

19.1. File Names

VM/SP CMS and VM/XA SP CMS MAINSAIL file names use a single dot rather than spaces to separate the various file name components. For example, the CMS file name of the form:

```
                <file name> <file type> <file mode>
```

is specified to MAINSAIL as:

```
                <file name>.<file type>.<file mode>
```

Since CMS file names must have a file type, MAINSAIL appends ".MDATA" if <file type> is not specified. If <file mode> is omitted, MAINSAIL uses the file mode "A" if the file is being created, otherwise "*".

Lowercase characters in a file name are converted to upper case before they are seen by CMS.

19.2. File Formats

MAINSAIL correctly processes CMS fixed and variable files. The standard MAINSAIL view of a text file is a stream of bytes, with lines separated by <eol> (for EBCDIC, <eol> = linefeed = 37 decimal). This format is called the "byte stream" format. MAINSAIL uses the CMS fix file format with record size 2048 for byte stream files.

19.2.1. Sequential Text Input File Formats

The input records of a text file are automatically translated by the MAINSAIL runtime system into the MAINSAIL byte stream format, which is then made available to the user's program. Thus, the user program need not concern itself with the CMS concept of records except to ensure that the proper translation is applied. Table 19.2.1-1 describes the translation algorithm for each device prefix.

Device Prefix	Translation Algorithm
BS	Byte stream file: Fixed-length records. No translation takes place.
FIX	Fixed-length records: Trailing blanks are discarded and <eol> is appended.
VAR	Variable-length records: <eol> is appended.

Table 19.2.1-1. Input Translation Rules for Text Files

If no device prefix is specified, the device prefix used depends on the CMS file format, as shown in Table 19.2.1-2. MAINSAIL fails to open a file if the device prefix is explicitly specified and there is a conflict between it and the format of the existing file.

CMS Format	Device Prefix Used
FIX(n), n = 2048	BS
FIX(n), n NEQ 2048	FIX
VAR	VAR

Table 19.2.1-2. Default Device Prefixes for Sequential Text Input

19.2.2. Sequential Text Output File Formats

For a file opened for sequential output, the standard MAINSAIL runtime system recognizes the prefixes shown in Table 19.2.2-1. Here, "n" indicates an optional record size.

Device Prefix	Output Format
BS	Fixed-length records of size 2048. Byte stream format. The output stream is packed into the records with no translation. This is the most efficient format for MAINSAIL, but is not understood by CMS text processing utilities since there is no relationship between lines and records.
FIX(n)	Fixed-length records of size n, default n = 80. A new output record is started when either n bytes are output or an <eol> character is output. In the latter case the <eol> is discarded and the record is padded with blanks to make it n bytes.
VAR(n)	Variable-length records of maximum size n, default n = 132. A new output record is started when either n bytes are output or an <eol> character is output. In the latter case the <eol> is discarded. Records are not padded with blanks.

Table 19.2.2-1. Device Prefixes for Sequential Text Output

If a device prefix is specified, the file is created according to the device specification. If a device prefix is not specified, the device prefix "VAR(132)>" is used; i.e., the default format for sequential text output files is V-format with a maximum record length of 132.

19.2.3. Sequential Data File Formats

Any data file can be opened for sequential input. Sequential output data files must be either byte stream or fixed format (the default is byte stream).

19.2.4. Random File Formats

Any text file can be opened for random input. Random text output files must be byte stream format (fixed-length records of 2048 bytes). Random data output files can be either byte stream or fix format (the default is byte stream).

19.2.5. Converting Between File Formats

The MAINSAIL utility module COPIER can be used to convert between file formats by explicitly specifying the device module for the output file. Refer to the "MAINSAIL Utilities User's Guide" for a complete description of this utility.

20. System Information Procedures

All file names returned by the file information procedures are MAINSAIL file names; i.e., "." is included between <file name> and <file type>, and between <file type> and <file mode>, and the file name is converted to lower case.

20.1. $currentDirectory

$currentDirectory always returns "A", meaning the A disk.

20.2. $homeDirectory

$homeDirectory always returns "A", meaning the A disk.

20.3. $directory (for CMS Disk Files)

$directory returns a list of all files on the minidisk specified by directoryName. If directoryName is omitted, the "A" disk is assumed.

If $fullPathNames is specified, the file mode is returned as part of the file name; otherwise it is omitted.

$reportAllVersions is ignored.

20.4. $fileInfo (for CMS Disk Files)

$fileInfo returns information from the FST for the specified file. The extended format FSCB is used.

The file mode is included in p.$fullPathName. p.$osdSize is set to the alternate number of records times the record length. p.$createDate and p.$createTime are both set to 0L. p.$modifyDate and p.$modifyTime are set from the alternate date and time information.

20.5. $userId

$userId executes the diagnose instruction with code '0 and returns as a string the user identification information returned by that instruction.

20.6. $cpuId

$cpuId returns a string of length ten that contains the CPU serial number followed by the CPU model number.

20.7. Command Line

MAINSAIL obtains the command line from CMS when it is invoked. The command line provided by CMS may be in one of two forms, depending on how MAINSAIL is invoked. If MAINSAIL is invoked by typing a program name at the CMS prompt, then CMS provides the command line in the form of an extended parameter list. If MAINSAIL is invoked from within a CMS EXEC file, then CMS provides the command line in the form of a tokenized parameter list. MAINSAIL determines the form of the command line provided by CMS. If it is in the form of an extended parameter list, then MAINSAIL does not modify the string obtained from CMS. If it is in the form of a tokenized parameter list, then MAINSAIL puts one space between each token. In either case, the command line does not include the name of the program invoked.

20.8. $programName

$programName is set to be the first argument in the parameter list or extended parameter list.

20.9. Exit Codes

On CMS, the $successExitCode is defined to be '0L and $failureExitCode is defined to be '1L. The exit code is returned in register 15 when MAINSAIL exits.

20.10. $environment

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

MAINSAIL provides access to operating system dependent environment parameters. CMS MAINSAIL defines a system variable:

```
INTEGER $environment;
```

which is the value of the high order byte of general purpose register 1 (GPR1) upon entry to MAINSAIL. The meaning of these values is described in "VM/SP System Programmer's Guide". For example, if $environment is 13, it means that MAINSAIL was invoked from an CMS EXEC file with "&CONTROL MSG" in effect.

21. Character Set

The EBCDIC character set is used. Table 21-1 shows EBCDIC codes used by MAINSAIL.

Character	EBCDIC Code (decimal)
null character, $nulChar (typically ignored in text input files)	0 (nul)
tab (horizontal tab)	5 (ht)
eol (end-of-line)	37 (linefeed)
eop (end-of-page)	12 (formfeed)
left square bracket	173* (print chain left bracket for standard EBCDIC character set)
right square bracket	189* (print chain right bracket for standard EBCDIC character set)

* MAINSAIL software also treats cent sign and vertical bar (character codes 74 and 79) as left square bracket and right square bracket, respectively..

Table 21-1. EBCDIC Codes Used by MAINSAIL

22. Terminal I/O

VM/SP CMS and VM/XA SP CMS MAINSAIL support I/O from both ASCII and IBM 3270 terminals. In both cases, output is buffered until (1) an <eol> character (linefeed) is output, (2) the output buffer is full, or (3) a terminal read occurs. In each case, the buffer is output as a line to the terminal. Thus, output characters are not seen on the terminal until one of the above events occurs. The size of the output buffer depends on the terminal being used.

23. MAINEDIT, MAINSAIL Editor

The MAINSAIL editor is available on VM/SP CMS and VM/XA SP CMS. Refer to the "MAINEDIT User's Manual" for a complete description of the editor.

Since CMS is half-duplex, the user interaction is different from that on full-duplex systems. The cursor is placed either at the top or at the bottom of the screen whenever the editor is waiting for input. The actual cursor position in the text is marked in a terminal-dependent fashion; inverse video and bright mode are commonly used.

Commands are entered on the input line and are seen by the editor only when <eol> has been typed. Any number of commands can be entered on the input line. When <eol> is typed, all commands on the input line are processed and the screen is updated.

In general, entering a response to a prompt requires a terminating <eol>. In this case, it is necessary to type an additional <eol> (the first <eol> is not seen by the editor since it just terminates input).

Executing recursive macros and searches can be aborted by pressing the ATTN key once and then typing <abort> key followed by <eol>. In order to use this feature, the terminal mode must be set to VM; i.e., pressing the ATTN key once should send an interruption to the virtual machine and place the virtual machine in VM READ status. The command "CP TERMINAL MODE VM" sets the terminal mode to VM; VM is the default terminal mode.

In order that the screen be updated properly, the terminal LINESIZE setting must be OFF. When the editor is invoked, it automatically detects and remembers the current terminal LINESIZE setting and then sets it to OFF. Upon exit, the terminal LINESIZE setting is restored to its original value. If the editor is abnormally terminated, the terminal LINESIZE is not restored. In this case, use the "CP TERMINAL LINESIZE" command to restore the desired LINESIZE setting.

24. Suggested VM/SP CMS and VM/XA SP CMS Terminal Characteristics

Some of the default characters used by the CMS "TERMINAL" command are also used by MAINSAIL. For example, the default CMS terminal ESCAPE character is '"', which is used by MAINSAIL to delimit strings. With this default, it is necessary to type '""' in order that '"' be entered into a file. Problems such as these can be eliminated with the CMS commands shown in Figure 24-1.

```
cp term chardel <BS>          (backspace key)
cp term linedel <^U>          (control-U)
cp term escape off
cp term linend ~
```

Figure 24-1. Suggested CMS Terminal Characteristics

25. IBM System/370 and System/370 Extended Architecture Processor-Dependent Information

This chapter describes information specific to the IBM System/370 IBM System/370 Extended Architecture processors. The two processors are nearly identical except that the Extended Architure has a larger address space. Except where otherwise noted, features described for the System/370 apply to both processors.

25.1. Procedure Size

The size of a procedure is limited to 20K bytes on the System/370.

25.2. System/370 Data Types

Refer to Table 25.2-1. A storage unit on the System/370 is one byte (8 bits).

25.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

Data type	Representation
boolean	1 half word (2 bytes)
integer	Standard System/370 integer format (1 half word, 2 bytes)
long integer	Standard System/370 long integer format (1 word, 4 bytes)
real	Standard System/370 single precision floating point format (1 word, 4 bytes)
long real	Standard System/370 double precision floating point format (1 double word, 8 bytes)
bits	1 half word (2 bytes)
long bits	1 word (4 bytes)
string	2 words (8 bytes): first word (low address) is length, second word (high address) is charadr of first character
address	Standard System/370 address (1 word, 24 bits right aligned, with the high-order byte clear)
charadr	Same as address
pointer	Same as address

Table 25.2-1. IBM System/370 Data Types

26. Miscellaneous Information

26.1. Configuration String Location

Because of the way that the garbage collector currently works, strings must reside above the address 32K. If a string does not reside above 32K, then the garbage collector will get "confused" with unpredictable results. If you load your MAINSAIL boot anywhere below 32K, you must make sure that the strings in the CONF record are above 32K.

26.2. Disk Full Message

If MAINSAIL runs out of disk space, it prints the message "Disk full. Find some space, then continue" and then prints the "Error Response:" prompt. At this point, the user may delete one or more files using the utility module DELFIL. A directory listing may be obtained by invoking the utility module CMSDIR. This module is VM/SP CMS-specific and is, therefore, available only when running under VM/SP CMS, not VM/XA SP CMS.

If one or more files have been deleted, type <eol> in response to this prompt. MAINSAIL automatically reissues the write macro. If enough space has been made available, MAINSAIL continues execution with no further "disk full" errors.

Example 26.2-1 shows the user interaction when the "disk full" error message is given.

26.3. Running MAINSAIL from a CMS EXEC File

MAINSAIL can be run from a CMS EXEC file. In this case, input to MAINSAIL is stacked via the EXEC Control Statements &STACK and &BEGSTACK-&END.

The &BEGSTACK control statement is the preferred method of stacking non-empty input lines because the lines between the &BEGSTACK and &END statements are not processed by the EXEC interpreter and are therefore passed unmodified to MAINSAIL. Use the &STACK control statement to stack a null input line.

CMS EXEC files that invoke MAINSAIL and stack input should be CMS V-format files. If the EXEC file is a CMS F-format file, then all stacked lines are right-padded with blanks and such input is not always desirable. MAINSAIL cannot in general strip all trailing blanks from input

```
    (MAINSAIL is executing module A)

Error for file foo.dat


ERROR: FSWRITE: 13 - Disk is full.  Find some space then
    continue.
Execute CMSDIR for a directory listing and DELFIL to
    delete files.
Error Response: .e cmsdir<eol>
Type <eol> to see the directory listing.
Type <eol> to continue after the directory listing has
    been displayed.
.<eol>

    (Directory Listing is Displayed)

.<eol>
Error Response: .e delfil<eol>
Next file to be deleted (just eol to stop): .bar.dat<eol>
Next file to be deleted (just eol to stop): .<eol>
Error Response: .<eol>

    (MAINSAIL continues executing module A)
```

Example 26.2-1. Disk Full User Interaction Example

lines because it can make no assumptions about how the input is going to be interpreted by a user program; i.e., the user program may require one or more trailing blanks in response to a prompt.

UNIX MAINSAIL®
User's Guide

24 March 1989

27. Introduction

This document describes the MAINSAIL implementation for UNIX, the AT&T portable operating system, and other UNIX-compatible operating systems. This document describes only UNIX-specific MAINSAIL features. It assumes that the reader is familiar with the "MAINSAIL Language Manual" and other machine-independent MAINSAIL documentation.

27.1. Version

This version of the "UNIX MAINSAIL User's Guide" is current as of Version 12.10 of MAINSAIL. It incorporates the "UNIXVAX MAINSAIL Version 5.10 Release Note" of September, 1982; the "Unix Version 7.4 Release Note" of May, 1983; the "UNIX MAINSAIL Release Note, Version 8" of January, 1984; the "UNIX MAINSAIL Release Note, Version 9" of February, 1985; the "UNIX MAINSAIL Release Note, Version 10" of March, 1986; and the "UNIX MAINSAIL Release Note, Version 11" of July, 1987.

The different UNIX platforms are referred to as "flavors" of UNIX. In flavor-independent examples of interaction with the UNIX shell, the percent sign ("%") is shown as the shell prompt. In flavor-specific examples, the shell prompt used by the particular UNIX flavor is shown. Common UNIX shell prompts include the dollar sign ("$"), the percent sign ("%"), and the pound sign ("#").

28. General Operation

28.1. Installation Assumptions

It is assumed that the MAINSAIL files have been installed on a directory called the "MAINSAIL directory". The examples in this document assume that this directory is named "/usr/mainsail/12.10". This naming convention is only a suggestion; the MAINSAIL directory may have any name. Substitute the name of the MAINSAIL directory on your system for the string "/usr/mainsail/12.10" wherever it appears in the examples.

It is assumed that once the MAINSAIL bootstrap has been created, it is placed in the standard directory for executable files (usually "/usr/bin") so that users do not need to type the full path name of the MAINSAIL bootstrap each time they run MAINSAIL. If this assumption is not satisfied, the discussion below must be modified accordingly; i.e., the full path name of the MAINSAIL bootstrap must be typed each time MAINSAIL is invoked.

28.2. Invoking MAINSAIL

To run a MAINSAIL program type "mainsa<eol>" to the shell. MAINSAIL begins execution and types a herald identifying itself and the version of MAINSAIL being used. It then types "*" as a prompt and waits for input. The "*" prompt and possible responses to it are described in the MAINEX section of the "MAINSAIL Utilities User's Guide".

28.3. Object Module File Names

The object module file name for a module compiled for UNIX is constructed by converting the module name to lower case and appending to it the appropriate flavor-dependent prefix, as shown in Table A.1-1; as described in Appendix A, this file name is usually mapped to a different one through a searchpath. Use of the MAINSAIL compiler is described in the "MAINSAIL Compiler User's Guide".

29. CONF, MAINSAIL Configurator

The MAINSAIL configuration module, CONF, is described in the "MAINSAIL Utilities User's Guide". In addition to the target-independent commands described therein, all UNIX versions of CONF provide the additional commands shown in Table 29-1.

CONF Command	Meaning
UNIXBITS	Set UNIX-specific attributes.

Table 29-1. UNIX-Specific CONF Commands

29.1. "UNIXBITS"

This command is used to identify the UNIX flavor to the MAINSAIL runtime system. The standard configuration file automatically defines the value of this parameter correctly for the current version of MAINSAIL on the host system. Unless otherwise specified, this value _must_ be used. If an incorrect "UNIXBITS" value is given, MAINSAIL may not run at all.

If you maintain your own configuration files, you must update the "UNIXBITS" values in them when you move to a new version of MAINSAIL or another flavor of UNIX.

29.2. Flavor-Specific CONF Commands

29.2.1. "SIGPC"

The CONF command "SIGPC" is required on some UNIX flavors. Its parameter is the offset to the error PC in the UNIX stack frame when a UNIX signal is intercepted (if the error PC is not available in portable C code). The "SIGPC" parameter must be correctly specified; otherwise, signals cannot be correctly intercepted. The correct value for each system is specified in the default configuration parameters file for each system. Always use this value when making a bootstrap for any flavor of UNIX MAINSAIL for which a "SIGPC" value appears in the default configuration file.

29.3. OS Memory Pool

The OS memory pool, as specified by the CONF command "OSMEMORYPOOLSIZE", is implemented for UNIX. The specified amount of memory is first allocated with "malloc", then deallocated, to provide a pool of memory to be used by C library and FLI procedures that allocate memory using malloc. This avoids unused static pages in MAINSAIL's memory map.

29.4. General Use

The default CONF parameters are normally kept in a standard configuration file on the MAINSAIL directory. Table B.1-1 in Appendix B gives the name of the standard configuration file for each available UNIX flavor. UNIX systemwide changes should always be made to this file. The CONF "SAVE" and "RESTORE" commands can be used to make personal configuration files.

The output of CONF under UNIX is an assembly language file. This file must be assembled and linked to make a UNIX executable file. Section B.2 in Appendix B shows how to run CONF to make an executable MAINSAIL bootstrap from the output of CONF for each available UNIX flavor.

UNIX CONF appends ".s" to the output file name if it does not already end in ".s".

30. MAINSAIL and the UNIX File System

30.1. File Deletion

On UNIX, when an existing file is opened for create access, the old version of the file is immediately truncated to length zero whether or not it is currently open. Any program that has the file currently open will therefore start reading data from the new (zero-length) version of the file without receiving any warning that the data it was reading have disappeared.

30.2. MAINEDIT and Links

When MAINEDIT deletes an existing file and replaces it with a new version, it does not break links; i.e., every file to which the updated file was a link is also updated.

30.3. Protection Mode of Files

The disk module propagates the protection mode of a file when replacing an existing file. If the file does not exist, then MAINSAIL attempts to use the protection of the parent directory. If this information is not ascertainable, then the default protection modes shown in Table 30.3-1 are used.

File type	Owner	Group	Other
textFile	rw-	r--	r--
dataFile	rwx	r-x	r-x

Table 30.3-1. Default Protection Mode

If the current umask value (as set by the shell command "umask") is non-zero, the protection specified by MAINSAIL may not be the same as the protection actually given to the file. See the UNIX documentation on "umask" for details.

30.4. Disk Full

If a write to a file fails because space on the device is exhausted, the resulting error message is not fatal. The user may free up space on the device (e.g., with the MAINSAIL utility module DELFIL) and continue; the write operation is reattempted.

31. UNIX STREAMS

Child processes created with SOCPRO are in the same process group as the parent. This should prevent children from staying around after the parent dies unexpectedly.

If the child's primary I/O is a TTY, the child is placed in its own private process group, so that if a CTRL-C is passed to the child, the parent does NOT get interrupted. This is reasonable for using a PTY as it would be used in a shell.

If the child's primary I/O device is NOT a TTY, e.g., is a pipe or socket as it would be for SOCPRO, the child is left in the parent's process group so that when the parent dies, the child receives the same signals that the parent received (e.g., SIGQUIT).

32. System Information Procedures

32.1. $currentDirectory

$currentDirectory returns the full path name of the current directory ("."). Symbolic links are expanded; i.e., even if a symbolic link was originally specified in the system call that connected to the current directory, the full directory name is returned. On some flavors, it is required that every directory in the current path be readable.

32.2. $homeDirectory

$homeDirectory returns the value of the "HOME" environment variable.

32.3. $directory (for UNIX Disk Files)

$directory returns the names of the files in the specified directory. If $fullPathName is set, the directory name and a slash character are prefixed to each file name. $reportAllVersions is ignored. The files "." and ".." are not included in the directory listing.

32.4. Command Line and $programName

$programName is set to the first element of the argument vector argv. The command line is formed from the remaining elements, separated by spaces.

32.5. $fileInfo (for UNIX Disk Files)

Since UNIX does not provide the creation date and time of a file, $fileInfo sets the fields $createDate and $createTime of the class $fileInfoCls to 0L.

32.6. $userID

$userID calls cuserid or getpwuid, if available; otherwise, it returns the user name found in "/etc/passwd" for the current user ID, as given by the UNIX call "getuid".

32.7. $cpuID

At present, $cpuID returns the null string on every available flavor of UNIX.

On UNIX, if $cpuID would return the null string, it checks to see if $useAlternateCpuID ('H200) is set in the configuration bits. If so, then it returns whatever is produced by the FLI procedure $alternateCpuID. This procedure must be provided by the user in an FLI module called $aCpuID, declared as follows:

```
MODULE $aCpuID (

INTEGER
PROCEDURE    $alternateCpuID
                          (CHARADR buf; INTEGER bufSize);

);
```

$alternateCpuID returns a null-terminated string no longer than bufSize at buf. $aCpuId should be compiled with the appropriate Foreign Call Compiler and linked with a MAINSAIL bootstrap in which the 'H200 bit was set in the "CONFIGURATIONBITS" command.

32.8. Exit Codes

The return code on UNIX is the value passed to the system call "exit". $successExitCode is '0L and $failureExitCode is a long bits value in which every bit is set.

32.9. $environment

The UNIX environment pointer array "environ" is made available to MAINSAIL programs on UNIX as the string array $environment, declared as:

```
                STRING ARRAY(1 TO *) $environment;
```

Each string in the array has the form "name=value". More information may be found in part 5 of the UNIX manuals under "environ".

33. Foreign Language Interface

This section contains UNIX-specific information for the MAINSAIL Foreign Language Interface (FLI). Refer to the "MAINSAIL Compiler User's Guide" for a general description of the FLI.

33.1. FLI Compilers

The Foreign Call Compiler (FCC) is used to call foreign procedures from MAINSAIL. The MAINSAIL Entry Compiler (MEC) is used to call MAINSAIL procedures from a foreign language. Under UNIX, the FCC interfaces to C and is available on all available flavors of UNIX. The MEC also interfaces to C and is available for the flavors listed in Appendix A. If the foreign language to be called does not conform to the C calling conventions, then the standard FLI compilers cannot be used. Contact XIDAK if you wish an interface between such a language and UNIX MAINSAIL.

The compiler subcommands required to specify the FLI's to and from C are "FLI TC" and "FLI FC", except as documented in Appendix A.

The default output file name for the MAINSAIL FLI compilers is "<module name, converted to lower case>.s".

33.2. Data Types

Table 33.2-1 shows how to map C data types to MAINSAIL data types on UNIX operating systems. The byte sizes shown are usual for C implementations on byte-addressable machines, although it is possible that users may encounter a dialect of C with data types of sizes other than those shown in Table 33.2-1; if so, the following discussion must be modified accordingly.

Uses parameters are passed by value. Modifies and produces parameters are passed by reference. The C "*" operator may be used when declaring modifies and produces parameters.

Uses parameters of types string and pointer can be passed from MAINSAIL to C. Parameters of types pointer and string cannot be passed from C to MAINSAIL. Modifies and produces parameters of types string and pointer are not allowed.

C type	MAINSAIL type	Representation Passed
short	BOOLEAN, INTEGER, BITS	2 bytes
long	LONG INTEGER, LONG BITS,	4 bytes
pointer	POINTER, ADDRESS	4 bytes
float	REAL	4 bytes
double	LONG REAL	8 bytes
char array	STRING, CHARADR	address of first character*
array	ARRAY	address of first element

* Refer to discussion concerning strings in this section

Table 33.2-1. MAINSAIL Data Types and Qualifiers

MAINSAIL strings can be directly passed to C, i.e., the parameter is declared using the MAINSAIL string data type. C expects strings to be terminated with a null byte; "cWrite(s,0)" can be used to append a null byte to the MAINSAIL string s.

When a pointer is passed to C, the C structure and the MAINSAIL class must be declared such that the data in the structure are accessed correctly by both languages. Some flavors of UNIX have data alignment requirements that affect the location of members in C structures; see Appendix A for more information.

Array parameters are passed to C by passing the address of the first element of the array. The Foreign Call Compiler generates code to convert each MAINSAIL array parameter to the address of its first element. MAINSAIL stores arrays in row-major form. If the foreign language does not follow this convention, care must be taken with the indices.

A MAINSAIL nullArray is passed to C as a Zero address. C must check that the address is non-Zero before accessing any array elements if there is a possibility that a MAINSAIL nullArray has been passed.

The MAINSAIL representation of a boolean should be translated into C as a short, with 0 representing MAINSAIL FALSE and 1 representing MAINSAIL TRUE.

33.3. Caveat

When dealing with collectable data types (strings, pointers, and arrays), care must be taken not to violate the data boundaries. Overwriting the wrong parts of these structures may cause MAINSAIL to crash in ways that are difficult to trace. If, for example, a garbage collection link is destroyed, MAINSAIL may run until a collection is triggered, at which point the results are unpredictable.

Appendix A lists other restrictions and caveats for specific UNIX flavors.

33.4. MAINSAIL Foreign Call Compiler Example

Suppose that the MAINSAIL module CALLC calls the C procedure "proc1". Figures 33.4-1, 33.4-2, and 33.4-3 show the module CALLC, the C procedure proc1, and the MAINSAIL FLI module TOC, respectively. Example 33.4-4 shows how to compile and run CALLC.

```
Module CALLC (in file "callc.msl"):

BEGIN "callC"

MODULE toC (
    PROCEDURE proc1 (
        PRODUCES INTEGER        i;
        INTEGER                 j,k;
        PRODUCES CHARADR        c;
        POINTER(c1)             p;
        STRING                  s;
        BOOLEAN ARRAY(1 TO 2)   ary);
    );

CLASS c1 (LONG INTEGER li1,li2);
```

Figure 33.4-1. MAINSAIL Module CALLC That Calls C Procedure proc1 (continued)

```
INITIAL PROCEDURE;
BEGIN
INTEGER                 i;
STRING                  s;
POINTER(cl)             p;
CHARADR                 c,cc;
BOOLEAN ARRAY(1 TO 2)   ary;
# C procedure proc1 does the following:
#    (1) adds j and k and returns the result in i
#    (2) creates a string and returns the address of its
#        first character in c
#    (3) swaps the values of the fields of CLASS cl
#    (4) prints the value of s
#    (5) sets ary[1] to be FALSE
p := new(cl); p.li1 := 1L; p.li2 := 2L;
s := "Hello there!" & cvcs(0);
new(ary); ary[1] := ary[2] := TRUE;
proc1(i,1,2,c,p,s,ary);
ttyWrite("i = ",i,eol);
i := 0; cc := c; WHILE cRead(cc) DO i .+ 1;
ttyWrite(
    "String returned by proc1 = ",newString(c,i),eol &
    "p.li1 = ",p.li1,"; p.li2 = ",p.li2,eol &
    "ary[1] = ary[2]: ",
        IF ary[1] = ary[2] THEN "TRUE" EL "FALSE",eol);
END;

END "callC"
```

Figure 33.4-1. MAINSAIL Module CALLC That Calls C Procedure proc1 (end)

```
C procedure proc1 (in file "cproc.c"):

proc1 (i,j,k,c,p,s,ary)
short *i;
short j,k;
char **c;
struct {int li1,li2;} *p;
char s[];
short ary[];
{
int temp;
*i = j + k;
*c = "Bye for now..."; /* New value for string */
temp = p->li1; p->li1 = p->li2; p->li2 = temp;
printf("String is %s\n",s);
ary[1] = 0;
}
```

Figure 33.4-2. C Procedure proc1

```
MAINSAIL Module TOC (in file "toc.msl"):

BEGIN "toC"

MODULE toC (
    PROCEDURE proc1 (
        PRODUCES INTEGER        i;
        INTEGER                 j,k;
        PRODUCES CHARADR        c;
        POINTER(cl)             p;
        STRING                  s;
        BOOLEAN ARRAY(1 TO 2)   ary);
    );

PROCEDURE proc1 (
    PRODUCES INTEGER        i;
    INTEGER                 j,k;
    PRODUCES CHARADR        c;
    POINTER(cl)             p;
    STRING                  s;
    BOOLEAN ARRAY(1 TO 2)   ary);

END "toC"
```

Figure 33.4-3. MAINSAIL Foreign Language Interface Module TOC

```
(1) Compile CALLC with the MAINSAIL UNIX compiler.
    Compile TOC with the FCC to C (by specifying the
    compiler subcommand "fli tc" on most UNIX systems;
    see Appendix A for exceptions).

% mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>
```

Example 33.4-4. MAINSAIL to C Example (continued)

```
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): callc.msl<eol>
Opening intmod for $SYS...

callc.msl 1 ...

Objmod for CALLC on callc-xxx.obj
Intmod for CALLC not stored

compile (? for help): toc.msl,<eol>
>fli tc<eol>
><eol>
Opening intmod for $SYS...

toc.msl 1 ...

Output for TOC on toc.s
Intmod for TOC not stored

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

(control returns to the UNIX shell)

(2) Make a new MAINSAIL bootstrap that declares TOC to be
    a foreign module.  The foreign module names must be
    uppercase on all flavors of UNIX except ULTRIX-32,
    on which the foreign module names must be lowercase.
    UNIX assemblers are case-sensitive.
```

Example 33.4-4. MAINSAIL to C Example (continued)

```
% mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/xxx.cnf
CONF: bootfilename fcc.s<eol>
CONF: foreignmodules<eol>
FOREIGNMODULES is
...
Should be:
=<eol>
TOC<eol>
<eol>
CONF: <eol>
Bootstrap written in file fcc.s
*<eol>

(control returns to the UNIX shell)

(3) Compile the C code with the C compiler.

% cc -c cproc.c<eol>

(4) Assemble and link the new MAINSAIL bootstrap.  The
    commands shown are typical of some UNIX flavors,
    but do not apply to all UNIX flavors.  Consult
    Appendix B for details about how to
    make MAINSAIL bootstraps.

% cc -o fcc /usr/mainsail/12.10/m.o\<eol>
    fcc.s toc.s cproc.o<eol>

(5) Run the new executable MAINSAIL bootstrap and call
    the foreign procedure.
```

Example 33.4-4. MAINSAIL to C Example (continued)

```
% fcc<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*callc<eol>
```

Example 33.4-4. MAINSAIL to C Example (end)

33.5. MAINSAIL Entry Compiler Example

At present, the ability to invoke MAINSAIL from a C main program (referred to as foreign code starts execution) is not supported on all flavors of UNIX. Appendix A lists the UNIX flavors which support this feature.

Suppose that the C procedure callms is to call the MAINSAIL procedure proc1. Figures 33.5-1, 33.5-2, 33.5-3, and 33.5-4 show the C procedure callms, the MAINSAIL module MSMOD that contains the procedure proc1, the MAINSAIL FLI module TOC, and the MAINSAIL module CALLC that calls the C procedure callms, respectively. Example 33.5-5 shows how to compile and run callms.

```
C program (in file "callms.c"):

callms ()
{
/*  MAINSAIL procl does the following:
    (1) Adds its first two arguments and returns
        the result
    (2) Sets bo to be TRUE
*/
short bo;
int result,lil,li2;
bo = 0; lil = 1; li2 = 2;
result = procl(lil,li2,&bo);
printf("Result is %d\n",result);
if (bo == 0) {printf("FAILURE: bo should be nonZero\n");}
}
```

Figure 33.5-1. C Procedure That Calls MAINSAIL Procedure procl

```
MAINSAIL Module MSMOD (in file "msmod.msl"):

BEGIN "msMod"

MODULE msMod (
    LONG INTEGER PROCEDURE proc1 (
        LONG INTEGER         li1,li2;
        PRODUCES BOOLEAN     bo);
    );

LONG INTEGER PROCEDURE proc1 (
    LONG INTEGER         li1,li2;
    PRODUCES BOOLEAN     bo);
BEGIN
bo := TRUE;
RETURN(li1 + li2);
END;

END "msMod"
```

Figure 33.5-2. MAINSAIL Module MSMOD Called by C Procedure callms

```
MAINSAIL Module TOC (in file "toc.msl"):

BEGIN "toC"

MODULE toC (PROCEDURE callMs);

PROCEDURE callMs;;

END "toC"
```

Figure 33.5-3. MAINSAIL Foreign Language Interface Module TOC

```
MAINSAIL Module CALLC (in file "callc.msl"):

BEGIN "callC"

MODULE toC (PROCEDURE callMs);

INITIAL PROCEDURE;
callMs;

END "callC"
```

Figure 33.5-4. MAINSAIL Module CALLC That Calls C Procedure callms

```
(1) Compile MSMOD and CALLC with the UNIX MAINSAIL
    compiler.  Compile MSMOD with the MEC from C (by
    specifying the compiler subcommand "fli fc").
    Compile TOC with the FCC to C (by specifying the
    compiler subcommand "fli tc").  The subcommands
    "fli fc" and "fli tc" may be different on some UNIX
    flavors; see Appendix A.

% mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>

MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.

compile (? for help): msmod.msl<eol>
Opening intmod for $SYS...

msmod.msl 1 ...
```

Example 33.5-5. C to MAINSAIL Example (continued)

```
compile (? for help): callc.msl<eol>
Opening intmod for $SYS...

callc.msl ...

compile (? for help): msmod.msl,<eol>
>fli fc<eol>
><eol>
Opening intmod for $SYS...

msmod.msl 1 ...

compile (? for help): toc.msl,<eol>
>fli tc<eol>
><eol>
Opening intmod for $SYS...

toc.msl ...

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

 (control returns to the UNIX shell)

 (2) Make a new MAINSAIL bootstrap that declares TOC to be
     a foreign module.  The foreign module names must be
     uppercase on all flavors of UNIX except ULTRIX-32,
     on which the foreign module names must be lowercase.
     UNIX assemblers are case-sensitive.
```

Example 33.5-5. C to MAINSAIL Example (continued)

```
% mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/xxx.cnf
CONF: bootfilename mec.s<eol>
CONF: foreignmodules<eol>
FOREIGNMODULES is
...
Should be:
=<eol>
TOC<eol>
<eol>
CONF: <eol>
Bootstrap written in file mec.s
*<eol>

(control returns to the UNIX shell)

(3) Compile the C code with the C compiler.

% cc -c callms.c<eol>

(4) Assemble and link the new MAINSAIL bootstrap.  The
    commands shown are typical of some UNIX flavors,
    but do not apply to all UNIX flavors.  Consult
    Appendix B for details about how to
    make MAINSAIL bootstraps.

% cc -o mec /usr/mainsail/12.10/m.o\<eol>
    mec.s msmod.s toc.s callms.o<eol>

(5) Run the new executable MAINSAIL bootstrap and call
    the foreign procedure.
```

Example 33.5-5. C to MAINSAIL Example (continued)

```
% mec<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*callc<eol>
Result is 3
*<eol>
```

Example 33.5-5. C to MAINSAIL Example (end)

34. CLIPPER Processor-Dependent Information

This chapter describes CLIPPER-specific information.

34.1. Procedure Size

There is no well-defined limit for the size of a procedure on the CLIPPER. However, the compiler almost always successfully compiles procedures under 64K bytes in length that contain no Case Statements for which code exceeds 32K bytes in length. The compiler may handle procedures larger than 64K bytes, depending on the code. Procedures longer than approximately 64K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

34.2. CLIPPER Data Types

Refer to Table 34.2-1. A storage unit on the CLIPPER is one byte (8 bits).

Data type	Representation
boolean	1 word (4 bytes)
integer	1 word (4 bytes)
long integer	1 word (4 bytes)
real	CLIPPER single precision (4 bytes)
long real	CLIPPER double precision (8 bytes)
bits	1 word (4 bytes)
long bits	1 word (4 bytes)
string	2 words (8 bytes): first word (low address) is length, second word (high address) is charadr of first character
address	1 word (4 bytes)
charadr	1 word (4 bytes)
pointer	1 word (4 bytes)

Table 34.2-1. CLIPPER Data Types

34.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

35. Intel 80386 Processor-Dependent Information

This chapter describes Intel 80386-specific information.

35.1. Procedure Size

There is no well-defined limit for the size of a procedure on the Intel 80386. However, the compiler almost always successfully compiles procedures under 64K bytes in length that contain no Case Statements for which code exceeds 32K bytes in length. The compiler may handle procedures larger than 64K bytes, depending on the code. Procedures longer than approximately 64K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

35.2. Intel 80386 Data Types

Refer to Table 35.2-1. A storage unit on the Intel 80386 is one byte (8 bits).

Data type	Representation
boolean	1 word (2 bytes)
integer	1 word (2 bytes)
long integer	1 doubleword (4 bytes)
real	Intel 80386 single precision (4 bytes)
long real	Intel 80386 double precision (8 bytes)
bits	1 word (2 bytes)
long bits	1 doubleword (4 bytes)
string	2 doublewords (8 bytes): first doubleword (low address) is length, second doubleword (high address) is charadr of first character
address	1 doubleword (4 bytes)
charadr	1 doubleword (4 bytes)
pointer	1 doubleword (4 bytes)

Table 35.2-1. Intel 80386 Data Types

35.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

36. IBM System/370 and System/370 Extended Architecture Processor-Dependent Information

This chapter describes information specific to the IBM System/370 IBM System/370 Extended Architecture processors. The two processors are nearly identical except that the Extended Architure has a larger address space. Except where otherwise noted, features described for the System/370 apply to both processors.

36.1. Procedure Size

The size of a procedure is limited to 20K bytes on the System/370.

36.2. System/370 Data Types

Refer to Table 36.2-1. A storage unit on the System/370 is one byte (8 bits).

36.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

Data type	Representation
boolean	1 half word (2 bytes)
integer	Standard System/370 integer format (1 half word, 2 bytes)
long integer	Standard System/370 long integer format (1 word, 4 bytes)
real	Standard System/370 single precision floating point format (1 word, 4 bytes)
long real	Standard System/370 double precision floating point format (1 double word, 8 bytes)
bits	1 half word (2 bytes)
long bits	1 word (4 bytes)
string	2 words (8 bytes): first word (low address) is length, second word (high address) is charadr of first character
address	Standard System/370 address (1 word, 24 bits right aligned, with the high-order byte clear)
charadr	Same as address
pointer	Same as address

Table 36.2-1. IBM System/370 Data Types

37. M68000 and MC68020 Processor-Dependent Information

This chapter describes M68000 Family-specific information.

37.1. M68000 vs. MC68020 Code Generation

XIDAK's M68000 code generator produces code that runs on the entire line of processors supporting Motorola's M68000 Family architecture, including the MC68000, MC68010, and MC68020. The M68000 code generator does not take advantage of any of the special instructions available on the latter two processors. The MC68020-specific code generators produce code that runs more efficiently on the MC68020 and the accompanying floating point processors included by various manufacturers with the MC68020 (the supported floating point processors at present are the Motorola MC68881 and the Weitek FPA). Code produced for the MC68020 does not run on earlier processors in the M68000 Family.

Except where otherwise specified, features described for the M68000 apply to the MC68020 as well.

37.2. Procedure Size

There is no well-defined limit for the size of a procedure on the M68000. However, the compiler almost always successfully compiles procedures under 32K bytes in length, and may handle larger procedures, depending on the code. Procedures longer than approximately 32K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

37.3. M68000 Data Types

Refer to Table 37.3-1. A storage unit on the M68000 is one byte (8 bits).

Data type	Representation
boolean	1 word (2 bytes)
integer	Standard M68000 integer format (1 word, 2 bytes)
long integer	Standard M68000 long integer format (1 longword, 4 bytes)
real	Depends on operating system, usually 1 longword (4 bytes)
long real	Depends on operating system, usually 2 longwords (8 bytes)
bits	1 word (2 bytes)
long bits	1 longword (4 bytes)
string	2 longwords (8 bytes): first longword (low address) is length, second longword (high address) is charadr of first character
address	Standard M68000 address: 1 longword (4 bytes)
charadr	Same as address
pointer	Same as address

Table 37.3-1. M68000 Data Types

37.4. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

37.5. Program Counter at Processor Exception

The location of an M68000 processor exception as reported by the early M68000 CPU's may be as much as four bytes beyond the code which actually produced the error. This may lead

MAINDEBUG to position incorrectly on certain M68000 processor exceptions. The MC68020 does not have this problem.

38. PRISM Processor-Dependent Information

This chapter describes PRISM-specific information.

38.1. Procedure Size

There is no well-defined limit for the size of a procedure on the PRISM. However, the compiler almost always successfully compiles procedures under 64K bytes in length. The compiler may handle procedures larger than 64K bytes, depending on the code. Procedures longer than approximately 64K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

38.2. PRISM Data Types

Refer to Table 38.2-1. A storage unit on the PRISM is one byte (8 bits).

Data type	Representation
boolean	1 long word (4 bytes)
integer	1 long word (4 bytes)
long integer	1 long word (4 bytes)
real	PRISM single precision (4 bytes)
long real	PRISM double precision (8 bytes)
bits	1 long word (4 bytes)
long bits	1 long word (4 bytes)
string	2 long words (8 bytes): first long word (low address) is length, second long word (high address) is charadr of first character
address	1 long word (4 bytes)
charadr	1 long word (4 bytes)
pointer	1 long word (4 bytes)

Table 38.2-1. PRISM Data Types

38.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

39. SPARC Processor-Dependent Information

This chapter describes SPARC-specific information.

39.1. Procedure Size

There is no well-defined limit for the size of a procedure on the SPARC. However, the compiler always successfully compiles procedures under 32K bytes in length, and may handle larger procedures, depending on the code. Procedures longer than approximately 32K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

39.2. SPARC Data Types

Refer to Table 39.2-1. A storage unit on the SPARC is one byte (8 bits).

Data type	Representation
boolean	1 word (4 bytes)
integer	1 word (4 bytes)
long integer	1 word (4 bytes)
real	SPARC Single Precision Floating Point Format (1 word, 4 bytes)
long real	SPARC Double Precision Floating Point Format (1 double word, 8 bytes)
bits	1 word (4 bytes)
long bits	1 word (4 bytes)
string	2 words (8 bytes): first word (low address) is length, second word (high address) is charadr of first character
address	1 word (4 bytes)
charadr	1 word (4 bytes)
pointer	1 word (4 bytes)

Table 39.2-1. SPARC Data Types

39.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in forms such as "IF <boolean variable> THEN ...", <boolean variable> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

40. VAX-11 Processor-Dependent Information

This chapter contains information about MAINSAIL that is specific to the VAX-11 implementations.

40.1. Procedure Size

There is no well-defined limit for the size of a procedure on the VAX-11. However, the compiler may not compile procedures longer than 32K bytes, depending on the code. Procedures longer than approximately 32K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

40.2. VAX-11 Data Types

Refer to Table 40.2-1. A storage unit on the VAX-11 is one byte (8 bits).

Data Type	Representation
boolean	1 word (2 bytes)
integer	1 word (2 bytes)
long integer	1 longword (4 bytes)
real	single precision F_floating (4 bytes)
long real	double precision D_floating (8 bytes)
bits	1 word (2 bytes)
long bits	1 longword (4 bytes)
string	1 quadword (8 bytes) low-address longword is length; high-address longword is charadr of first character
address	1 longword (4 bytes)
charadr	1 longword (4 bytes)
pointer	1 longword (4 bytes)

Table 40.2-1. VAX-11 Data Types

40.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in constructs such as "IF <boolean value> THEN ...", <boolean value> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

Appendix A. Flavor-Dependent Features of UNIX MAINSAIL

Because different implementations of UNIX provide different facilities, MAINSAIL's view of its host processor and operating system is slightly different on each type of UNIX system. However, these differences do not ordinarily compromise the portability of MAINSAIL source programs that are not dependent on features specific to a particular UNIX variety.

A.1. Object Module File Name Extensions for Available UNIX Flavors

Table A.1-1 lists the objmod and intmod logical file name prefixes for each available UNIX flavor. The logical file names are normally mapped to actual file names with the searchpaths:

```
SEARCHPATH *-obj:* *2-*1.obj
SEARCHPATH *-int:* *2-*1.int
```

These searchpaths are specified in the default bootstraps and configuration files created when MAINSAIL is installed, and may be replaced if desired.

Platform Abbrev.	Flavor Name	Objmod Prefix	Intmod Prefix
aix	IBM's AIX on IBM System/370	uxa-obj:	uxa-int:
alnt	Alliant's CONCENTRIX on Motorola M68000	um6-obj:	um6-int:
hp20	HP's HP-UX on Motorola MC68020/MC68881	um2-obj:	um2-int:
hp38	SCO's XENIX on HP Vectra with Intel 80386	ui3-obj:	ui3-int:
hpux	HP's HP-UX on Motorola M68000	um6-obj:	um6-int:
ip32c	Intergraph's System V UNIX on Interpro 32C	ucl-obj:	ucl-int:
ipsc2	Intel's iPSC/2 System V UNIX on Intel 80386	ui3-obj:	ui3-int:
ix20	Apollo's DOMAIN/IX on Motorola MC68020/MC68881	um2-obj:	um2-int:
ixfpa	Apollo's DOMAIN/IX on Motorola MC68020/Weitek FPA	ua2-obj:	ua2-int:
ixpri	Apollo's DOMAIN/IX on Apollo PRISM	upr-obj:	upr-int:
sun2	Sun Microsystems' SunOS on Motorola M68000	um6-obj:	um6-int:
sun3	Sun Microsystems' SunOS on Motorola MC68020/MC68881	um2-obj:	um2-int:
sun38	Sun Microsystems' SunOS on Intel 80386	ui3-obj:	ui3-int:
sun4	Sun Microsystems' SunOS on SPARC	usp-obj:	usp-int:
ultrx	DEC's ULTRIX-32 on VAX-11	uva-obj:	uva-int:

Table A.1-1. UNIX Object Module File Name Extensions

A.2. FLI Considerations for Available UNIX Flavors

Plat. Abbr.	FCC	MEC	Foreign Starts	Label Prefix
aix	TC	FC	yes	no
alnt	TA	FA	yes	yes
hp20	TH	FH	yes	yes
hp38	TC	FC	no	no
hpux	TH	FH	yes	yes
ip32c	TC	FC	yes	yes
ipsc2	TC	FC	yes	no
ix20	TX	FX	yes	no
ixfpa	TX	FX	yes	no
ixpri	TC	FC	yes	no
sun2	TC	FC	yes	yes
sun3	TC	FC	yes	yes
sun38	TC	FC	yes	no
sun4	TC	FC	yes	yes
ultrx	TC	FC	yes	yes

Table A.2-1. Flavor-Dependent FLI Characteristics

Table A.2-1 shows flavor-dependent FLI characteristics for all supported flavors of UNIX. The meanings of the columns are:

- "Plat. Abbr.": platform (flavor) abbreviation.

- "FCC", "MEC": compiler "FLI" subcommand arguments. The assemblers and C compilers on many UNIX flavors have similar conventions. On the systems obeying the most common conventions, the FLI Foreign Call Compiler compiler subcommand is "FLI TC", and the MAINSAIL Entry Call Compiler compiler subcommand is "FLI FC". On other flavors, the assembler and/or compiler requires different conventions, so FLI abbreviations other than "TC" and "FC" must be used.

- "Foreign starts": "yes" if the $foreignCodeStartsExecution configuration bit is supported (so that C may get control before MAINSAIL).

- "Label prefix": "yes" if the default foreign label corresponding to a MAINSAIL procedure is preceded by the "_" (underscore) character. On all systems, the MAINSAIL procedure name is converted to lowercase, whether the underscore is prepended or not. The default label may be overridden with an "ENCODE" directive, as described in the "MAINSAIL Compiler User's Guide".

A.2.1. C Alignment Considerations

On some systems, C has different alignment and data type considerations from MAINSAIL's. If you wish to see an example of your C compiler's layout of a particular data structure, disassemble a C program that accesses the components of the structure.

In particular, be careful of the following:

- On some systems, C aligns long word data items to fullword boundaries. On such systems, a C structure may require an integral number of fullwords, even if the actual data do not occupy all of the last fullword. This must be taken into consideration when defining a MAINSAIL class for a C structure. For example, if the data in a C structure occupy 10 bytes, C uses and returns 12 bytes of information. In this case, the MAINSAIL class that describes this structure must be defined with an additional padding field so that it occupies at least 12 bytes. In addition, the data in the C structure do not necessarily occupy contiguous memory locations. For example, if the C structure contains a short followed by a two longs, then the short occupies the first two bytes of the structure and the next two bytes are unused. The MAINSAIL class declaration must account for such alignment within the C structure.

- On some systems, C aligns double floating point data to the size of a double. On such systems, a C structure may be padded to integral multiple of the size of a double if the structure contains double data, and the field alignment and padding considerations must be taken into account. In addition, the alignment requirement presents a problem when passing long real arrays to C procedures. When MAINSAIL creates a long real array, it does not ensure double alignment of the elements, but the C procedure assumes double alignment. For this reason, long real arrays cannot be directly passed to the C procedure. The MAINSAIL program must copy the array elements to an area such that the long real data are double aligned and then to pass the address of this area to the C procedure.

- On some systems, MAINSAIL integers are 4 bytes, the size of C longs, instead of 2 bytes, the size of C shorts. To make this difference transparent to MAINSAIL procedures that pass integer arguments to C procedures, the Foreign Call Compiler generates code to convert MAINSAIL integers to C shorts by throwing away the high-order two bytes of the MAINSAIL integer. This mechanism is used for all uses, modifies, and produces integer parameters. Modifies and produces C short parameters are sign-extended before they are passed back to the MAINSAIL caller. A C short, therefore, MUST be mapped to the MAINSAIL integer data type, not a bits, if it is treated as a signed integer.

MAINSAIL bits data are handled in the same manner as MAINSAIL integer data except that modifies and produces bits parameters are zero-extended instead of sign-extended before being passed back to the MAINSAIL caller.

Because of the difference in data type sizes, care must be used when passing MAINSAIL integer and bits arrays to a C procedure. In this case, MAINSAIL accesses 4 bytes of information for each array element, whereas C accesses 2 bytes. The MAINSAIL program must correctly store and retrieve information in integer and bits arrays used by C procedures.

The best way to avoid portability problems with C structs is to call a C procedure that returns information about the structs before constructing or passing a C struct to a C procedure. For example, the C procedure in Example A.2.1-1 informs its caller of the size of the struct s and the offsets of the s fields f1 and f2. The MAINSAIL procedure in Example A.2.1-2 uses this information to construct a C struct in scratch memory with the f1 and f2 fields filled in, then passes it to anotherCProcedure, which uses the struct. sInfo and anotherCProcedure are called through the C FLI (the FLI module is not shown).

```
LONG INTEGER sSize,f1Dsp,f2Dsp;
ADDRESS a;

...

sInfo(sSize,f1Dsp,f2Dsp);
a := newScratch(sSize);
store(a,...,f1Dsp); store(a,...,f2Dsp);
anotherCProcedure(a,...);
...
```

Example A.2.1-2. Using C Struct Information

A.3. Program Exceptions

No UNIX MAINSAIL implementation currently detects stack overflow.

```
struct s {
    short f1;
    long f2;
};

...

sInfo(sSize,f1Dsp,f2Dsp)
int *sSize,*f1Dsp,*f2Dsp;
{
    struct s ss;

    *sSize = sizeof(struct s);
    *f1Dsp = (int) &ss.f1 - (int) &ss;
    *f2Dsp = (int) &ss.f2 - (int) &ss;
}

...

anotherCProcedure(ss,...)
struct s *ss;
...
```

Example A.2.1-1. Passing C Struct Information Back to MAINSAIL

Signals (other than arithmetic exceptions) that are ignored when MAINSAIL is invoked are ignored by MAINSAIL; arithmetic signals are always intercepted. If the signals shown in Table A.3-1 are not ignored, MAINSAIL catches them and issues a fatal error message if they occur. Additional signals are intercepted as described in Section A.5.

SIGBUS	SIGEMT	SIGFPE
SIGILL	SIGIOT	SIGPIPE
SIGSEGV	SIGSYS	SIGTRAP

Table A.3-1. Standard UNIX Signals Caught by MAINSAIL

A.4. MAINEDIT

A.4.1. BIGSUN and "mainsab"

On Sun BSD 4.2 UNIX and SUN-4 UNIX, the bootstrap "mainsab" is configured to run the BIGSUN display module (user-created bootstraps must be linked with a special file, "mb.o" instead of "m.o", if they are to run the BIGSUN display module). Consult the "MAINEDIT User's Guide" for a detailed description of BIGSUN.

A.4.2. InterPro 32C Display Module

The display module for the terminal attached to the InterPro 32C is named "VT102M". <ecm> (Enter Command Mode) for this display module is <esc><esc> (the <esc> key pressed twice).

A.5. Terminal Handling

UNIX terminal handling is divided into two main types: BSD and System V. The abbreviations for the platforms that fall into each of these categories are shown in Table A.5-1.

BSD	System V
alnt	aix
ix20	hp20
ixfpa	hp38
ixpri	hpux
sun2	ip32c
sun3	ipsc2
sun38	
sun4	
ultrx	

Table A.5-1. BSD and System V UNIX Flavors

A.5.1. BSD Systems

On BSD systems, display modules use CBREAK instead of RAW mode. This has the following implications:

1. Flow control characters (CTRL-S and CTRL-Q) are intercepted and processed by UNIX.

2. SIGINT (usually CTRL-C) is intercepted by MAINSAIL, which asks for confirmation from "/dev/tty". Responding with "y<eol>" terminates the MAINSAIL process; "n<eol>" continues. The case of the response is not significant.

3. SIGTSTP (usually CTRL-Z or CTRL-Y) is intercepted by MAINSAIL, which resets the terminal to cooked mode, then re-raises the signal. This usually causes the MAINSAIL process to stop. If the process is resumed, it returns to the terminal mode in effect before the signal was received, and resumes execution.

4. SIGQUIT (usually CTRL-\) is intercepted by MAINSAIL, which displays the prompt "Yes? (? for help):". Among the options offered are:

```
?       to see a list of options
T       to see incremental CPU times
          and continue
D       to dump core and exit
E       to exit without a dump
F       to see open file descriptors
K       to kill program with SEGV to
          see which procedure is
          executing (if lucky)
B       to attempt to break at next
          debuggable procedure
<eol>   to continue the program
```

5. The "K" response to the prompt given when SIGQUIT is caught may be useful if MAINSAIL is not in a critical section and can handle the error without dying; it can then print a call stack or enter the debugger. This can be used to track down an infinite loop. Using this response saves you the trouble of going to another terminal, finding out the job number, and issuing a "kill" command to send the SEGV signal.

6. CBREAK mode clears the parity bit on terminal input characters, so that if a terminal has a META or EDIT key (which sets the parity bit), it is ignored.

A.5.2. System V Systems

On System V systems, UNIX intercepts and processes flow control characters. SIGINT behaves as on BSD systems. There is no SIGTSTP on System V UNIX. The mode used does not interfere with parity bits, so that META or EDIT keys may be used on such systems.

A.5.3. ioctl from Programs

MAINSAIL keeps track of what it believes to be the current terminal modes. It reads the current values when MAINSAIL is initialized, and keeps track of changes by MAINSAIL system software. User programs that call ioctl may confuse MAINSAIL's terminal handling if they do not restore terminal modes before the next time MAINSAIL alters the terminal mode.

The MAINSAIL STREAMS package provides facilities for terminal control that allow programs to achieve the effect of ioctl portably without confusing the MAINSAIL runtime system. See the "MAINSAIL STREAMS User's Guide" for details.

Appendix B. Flavor-Dependent Configuration on UNIX

B.1. Standard Configuration Files for Available UNIX Flavors

The standard configuration file name for each flavor of UNIX is formed by appending ".cnf" to the lowercase platform abbreviation. Table B.1-1 lists the name of the standard configuration file for each available flavor of UNIX.

Plat. Abbr.	Flavor Name	Configuration File Name
aix	IBM's AIX on IBM System/370	aix.cnf
alnt	Alliant's CONCENTRIX on Motorola M68000	alnt.cnf
hp20	HP's HP-UX on Motorola MC68020/MC68881	hp20.cnf
hp38	SCO's XENIX on HP Vectra with Intel 80386	hp38.cnf
hpux	HP's HP-UX on Motorola M68000	hpux.cnf
ip32c	Intergraph's System V UNIX on Interpro 32C	ip32c.cnf
ipsc2	Intel's iPSC/2 System V UNIX on Intel 80386	ipsc2.cnf
ix20	Apollo's DOMAIN/IX on Motorola MC68020/MC68881	ix20.cnf
ixfpa	Apollo's DOMAIN/IX on Motorola MC68020/Weitek FPA	ixfpa.cnf
ixpri	Apollo's DOMAIN/IX on Apollo PRISM	ixpri.cnf
sun2	Sun Microsystems' SunOS on Motorola M68000	sun2.cnf
sun3	Sun Microsystems' SunOS on Motorola MC68020/MC68881	sun3.cnf
sun38	Sun Microsystems' SunOS on Intel 80386	sun38.cnf
sun4	Sun Microsystems' SunOS on SPARC	sun4.cnf
ultrx	DEC's ULTRIX-32 on VAX-11	ultrx.cnf

Table B.1-1. Standard UNIX Configuration File Names

B.2. Producing a Bootstrap on Each Flavor of UNIX

This section shows a sample session for each available flavor of UNIX in which a bootstrap with default parameters is created. The output from CONF in each example is assumed to be named "mainsa.s"; the final bootstrap file is named "mainsa".

If the default configuration file includes foreign modules, these foreign modules must be included in the foreign module list for every MAINSAIL bootstrap. For example, if the default foreign module list contains:

```
UNISYS
BSDITF
```

then to add the foreign modules FOO and BAR to a MAINSAIL bootstrap, both UNISYS and BSDITF must be specified along with FOO and BAR. The "=" abbreviation allowed in mulitiline CONF commands (see the "MAINSAIL Utilities User's Guide") may be used to do this:

```
CONF: foreignmodules<eol>
FOREIGNMODULES is
UNISYS
BSDITF
Should be:
=<eol>
FOO<eol>
BAR<eol>
<eol>
```

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/aix.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-1. Making a Bootstrap for IBM's AIX on IBM System/370

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
/usr/mainsail/12.10/alnt.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -nxp -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-2. Making a Bootstrap for Alliant's CONCENTRIX on Motorola M68000

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/hp20.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o\<eol>
    -l bsdipc<eol>
```

Example B.2-3. Making a Bootstrap for HP's HP-UX on Motorola MC68020/MC68881

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/hp38.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-4. Making a Bootstrap for SCO's XENIX on HP Vectra with Intel 80386

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/hpux.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o\<eol>
    -l bsdipc<eol>
```

Example B.2-5. Making a Bootstrap for HP's HP-UX on Motorola M68000

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/ip32c.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% as -o mainsa.o mainsa.s<eol>
% cc -o mainsa mainsa.o /usr/mainsail/12.10/m.o<eol>
```

Example B.2-6. Making a Bootstrap for Intergraph's System V UNIX on Interpro 32C

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/ipsc2.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
$ as -o mainsa.o mainsa.s<eol>

   There are two different "cc" commands, one for the
   host bootstrap and one for the nodes.  For the host:

$ cc -g -o mainsa mainsa.o /usr/mainsail/12.10/hm.o\<eol>
    -host<eol>

   For the nodes:

$ cc -g -o mainsa mainsa.o /usr/mainsail/12.10/nm.o\<eol>
    -node<eol>
```

Example B.2-7. Making a Bootstrap for Intel's iPSC/2 System V UNIX on Intel 80386

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/ix20.cnf
CONF: <eol>
Bootstrap written in file mainsa.bin
*<eol>
% ld -o mainsa mainsa.bin /usr/mainsail/12.10/m.o<eol>
```

Example B.2-8. Making a Bootstrap for Apollo's DOMAIN/IX on Motorola MC68020/MC68881

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/ixfpa.cnf
CONF: <eol>
Bootstrap written in file mainsa.bin
*<eol>
% ld -o mainsa mainsa.bin /usr/mainsail/12.10/m.o<eol>
```

Example B.2-9. Making a Bootstrap for Apollo's DOMAIN/IX on Motorola MC68020/Weitek FPA

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/ixpri.cnf
CONF: <eol>
Bootstrap written in file mainsa.bin
*<eol>
% ld -o mainsa mainsa.bin /usr/mainsail/12.10/m.o<eol>
```

Example B.2-10. Making a Bootstrap for Apollo's DOMAIN/IX on Apollo PRISM

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/sun2.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-11. Making a Bootstrap for Sun Microsystems' SunOS on Motorola M68000

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/sun3.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-12. Making a Bootstrap for Sun Microsystems' SunOS on Motorola MC68020/MC68881

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/sun38.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-13. Making a Bootstrap for Sun Microsystems' SunOS on Intel 80386

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/sun4.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>

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

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

   For 4.0 and after:

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

   All versions of UNIX take the same "cc" command:

% cc -o mainsa mainsa.o /usr/mainsail/12.10/m.o<eol>
```

Example B.2-14. Making a Bootstrap for Sun Microsystems' SunOS on SPARC

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file
    /usr/mainsail/12.10/ultrx.cnf
CONF: <eol>
Bootstrap written in file mainsa.s
*<eol>
% cc -o mainsa mainsa.s /usr/mainsail/12.10/m.o<eol>
```

Example B.2-15. Making a Bootstrap for DEC's ULTRIX-32 on VAX-11

™

VAX/VMS MAINSAIL®

User's Guide

24 March 1989

xidak™

41. Introduction

This document describes the MAINSAIL implementation for VAX/VMS, the Digital Equipment Corporation operating system for the VAX-11. It describes only VAX/VMS-specific MAINSAIL features. It assumes that the reader is familiar with the "MAINSAIL Language Manual" and other machine-independent documentation.

41.1. Version

This version of the "VAX/VMS MAINSAIL User's Guide" is current as of Version 12.10 of MAINSAIL. It incorporates the "VMS MAINSAIL Version 5.11 Release Note" of October, 1982; the "VMS Version 7.4 Release Note" of May, 1983; the "VAX/VMS MAINSAIL Release Note, Version 8" of January, 1984; the "VAX/VMS MAINSAIL Release Note, Version 9" of February, 1985; the "VAX/VMS MAINSAIL Release Note, Version 10" of March, 1986; and the "VAX/VMS MAINSAIL Release Note, Version 11" of July, 1987.

42. General Operation

42.1. Installation Assumptions

This document assumes that a directory has been created that contains all of the MAINSAIL system files for the current version of MAINSAIL, and that its name has been assigned to the logical name "ms:". This assumption is automatically satisfied if MAINSAIL is installed according to the directions contained in the release note that accompanies the MAINSAIL distribution tape. Should MAINSAIL be installed in some other way, the instructions in this document must be modified accordingly.

It is assumed in the examples in this document that the directory containing the MAINSAIL system files is "[MAINSAIL.VMS.1210]".

42.2. Invoking MAINSAIL

To run MAINSAIL, type "r ms:mainsa<eol>" to the VAX/VMS command executive. MAINSAIL begins execution and types a herald identifying itself and the version of MAINSAIL being used. It then types "*" as a prompt and waits for input. The "*" prompt and possible responses to it are described in the MAINEX section of the "MAINSAIL Utilities User's Guide".

42.3. Default Intmod and Objmod Searchpaths

The default intmod and objmod searchpaths on VAX/VMS are:

```
SEARCHPATH *-int:* *2*1.int
SEARCHPATH *-obj:* *2*1.obj
```

43. MAINSAIL Configurator, CONF

The MAINSAIL configuration module, CONF, is described in the "MAINSAIL Utilities User's Guide".

Default CONF parameters are usually kept in the file "MS:VMS.CNF". VAX/VMS systemwide changes should always be made to this file.

The output of CONF for VAX/VMS is a VAX-11 assembly language file. This file must be assembled and linked to make a new executable bootstrap.

Example 43-1 illustrates the use of CONF and shows the VAX/VMS commands to assemble and link the resulting bootstrap. Default values are restored from the file "MS:VMS.CNF" and the bootstrap is written to the file "MAINSA.MAR".

```
$ r ms:mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ms:vms.cnf
CONF: <eol>
Bootstrap written in file mainsa.mar
*<eol>
$ macro mainsa<eol>
$ link mainsa<eol>
$ r mainsa<eol>

    (MAINSAIL executes)
```

Example 43-1. Making a New Bootstrap

43.1. VAX/VMS Stack Size

The initial coroutine "MAINSAIL" on VAX/VMS uses the system stack, which is quite large; the configuration "STACKSIZE" parameter is ignored when the initial coroutine is allocted. It is used by default when subsequent coroutines are allocated, however.

44. Uniform System Caller

Occasionally a programmer finds it necessary to call on the operating system to perform a special function for which there is no equivalent construct in MAINSAIL. This is done under VAX/VMS by means of the VAX/VMS-dependent procedure "$sysCall".

The procedure "$sysCall" performs a VAX/VMS system service call. This procedure is, of course, available only on machines running under VAX/VMS. MAINSAIL code that uses procedure "$sysCall" must not be considered portable. We recommend that you use this procedure only when necessary, and isolate the VAX/VMS-dependent code in a single place if possible, to facilitate moving the program to a machine running under another operating system.

44.1. Using $sysCall

$sysCall can be used to call any VAX/VMS system service. The number of parameters expected by the different system services varies from 0 to 16. There is a VAX/VMS-dependent MAINSAIL procedure, $sysCall<n>, for each group of system services that expects n parameters, where n ranges from 0 to 16. $sysCall is a generic procedure representing the procedures $sysCall0, $sysCall1, ..., $sysCall16. Figure 44.1-1 shows the declaration of procedure $sysCall<n>. sysCall is the address of the system service's entry vector, expressed as a long bits. parm1 through parm<n> are the parameters to the system service (if the system service expects any parameters), also expressed as long bits.

If the system service returns control to the user program, the result of $sysCall is the status code returned by the system service. A status code with the '1L bit set indicates successful completion; see the "VAX/VMS System Services Reference Manual" for a description of other status codes.

```
LONG BITS PROCEDURE $sysCall<n> (
    LONG BITS sysCall,parm1,...,parm<n>);
```

Figure 44.1-1. $sysCall<n> Declaration

44.1.1. $sysCall Parameters

All parameters expected by the system service must be specified in the call to $sysCall, even those marked as optional in the "VAX/VMS System Services Reference Manual". The parameters must be specified in the order in which they are shown in the system services manual.

It is the user's responsibility to convert the parameters properly to long bits. Passing pointers and strings requires care that a MAINSAIL garbage collection does not occur between the time the "cva" or "cvc" is performed and the call to $sysCall is actually done.

Some system service parameters are passed by address. MAINSAIL has no construct for obtaining the address of a local variable. If a system service expects one of its parameters to be an address and the parameter's address is not otherwise available to you, follow the steps shown in Figure 44.1.1-1.

```
1. Before the call to $sysCall, obtain a small amount of
   memory for use as a temporary work area by calling the
   MAINSAIL procedure "newScratch".  Refer to the
   "MAINSAIL Language Manual" for a description of
   newScratch.

2. If the system service uses as an input parameter the
   value(s) stored at the address, initialize the newly
   allocated work area to the value(s) to be passed to
   the system service.

3. Call $sysCall, passing the address of the work area
   (converted to LONG BITS) as the address parameter.

4. If the system service uses the work area to return
   result(s), copy the result(s) to another variable.

5. Dispose of the work area by calling the MAINSAIL
   procedure "scratchDispose".  Refer to the "MAINSAIL
   Language Manual" for a description of scratchDispose.
```

Figure 44.1.1-1. Passing an Address to a System Service

Some system services require the address of a VAX-11 string descriptor, which can be obtained by following the steps shown in Figure 44.1.1-2.

```
1. Include in your program a class declaration similar to
   the one shown in Example 44.1.1-3.

2. Use newScratch to allocate space for the descriptor and
   for the area the descriptor will point to, as shown in
   Example 44.1.1-4.  The identifier
   "numberOfStorageUnits" denotes an integer the value of
   which is the size in bytes of the area to be pointed
   to by the descriptor.
```

Figure 44.1.1-2. Passing VAX-11 String Descriptors to System Services

```
CLASS ascidCls (
    INTEGER len;
    BITS classAndType;
    ADDRESS addr);
```

Example 44.1.1-3. Class for VAX-11 String Descriptors

```
ADDRESS(ascidCls) vaxDscr;

vaxDscr := newScratch(size(ascidCls));
vaxDscr.len := numberOfStorageUnits;
vaxDscr.addr := newScratch(numberOfStorageUnits);
```

Example 44.1.1-4. Allocating Space for VAX-11 String Descriptors

44.1.2. System Service Entry Vector Addresses

You can find the address of a system service's entry vector by following the steps shown in Figure 44.1.2-1.

```
1. Write a short VAX-11 assembly language program
   referring to the symbol denoting the system service.

2. Assemble and link the program with the "DEBUG" option.

3. Run the program.  The VAX/VMS debugger will be invoked.
   Ask the debugger to display the value of the symbol
   denoting the system service's entry vector address.
   This is the symbol listed in the "VAX/VMS System
   Services Reference Manual" in the description of the
   particular system service, under the heading
   "High-Level Language Format".  The debugger will
   display a hexadecimal address.

4. In your MAINSAIL program, define a similar symbol to be
   the system service's entry vector address, and use the
   symbol in the call to $sysCall.
```

Figure 44.1.2-1. Finding a System Service Entry Vector Address

44.2. $sysCall Example

Suppose that a MAINSAIL module needs to suspend its own execution for an arbitrary number of seconds. It could do so by calling the $SETIMR and $WAITFR system services, the parameters of which are described in Figures 44.2-1 and 44.2-2 (it could also do so portably by calling the MAINSAIL system procedure $timeout).

```
efn     Number of event flag for which to wait.
```

Figure 44.2-2. $WAITFR Parameters

efn	Number of event flag to set when time interval expires.
dayTim	Address of quadword containing time interval expressed as a negative number in tenths of microseconds.
astAdr	Address of AST service routine to be called when time interval expires; 0 for none.
reqIdt	Number denoting request identification; 0 for none.

Figure 44.2-1. $SETIMR Parameters

Example 44.2-3 illustrates a short assembly language program that can be used to find out the addresses of the $SETIMR and $WAITFR system services' entry vectors.

```
.ENTRY  start,^M<>
$SETIMR_S efn=1,dayTim=start
$WAITFR_S efn=1
.END    start
```

Example 44.2-3. MACRO Program Using $SETIMR and $WAITFR

Example 44.2-4 shows the commands to assemble and link the program, and to run the VAX/VMS debugger to find out the value of SYS$SETIMR, the address of $SETIMR's entry vector, and SYS$WAITFR, the address of $WAITFR's entry vector. The program is contained in the file "SYSCAL.MAR".

The debugger showed that $SETIMR's entry vector address is 'H80000220L and $WAITFR's is 'H80000278L. Example 44.2-5 contains a MAINSAIL procedure to call the $SETIMR and $WAITFR system services using $sysCall.

```
$ macro/enable=debug syscal<eol>
$ link/debug syscal<eol>
$ r syscal<eol>
                    VAX-11 DEBUG Version 3.0
%DEBUG-I-INITIAL, language MACRO, module set to '.MAIN.'
DBG>ev sys$setimr<eol>
80000220
DBG>ev sys$waitfr<eol>
80000278
DBG>exit<eol>
$
```

Example 44.2-4. Finding the System Services' Addresses

```
PROCEDURE wait (INTEGER secondsToWait);
# Suspend the execution of the program for secondsToWait
# seconds.
BEGIN
DEFINE
    sysSeTimr           =   'H80000220L,
    sysWaitFr           =   'H80000278L,
    maxTimeInSecs       =   214,
    longWordSize        =   4,
    dayTimBufSize       =   8,
    deltaTicksPerSec    =   -10000000L,
    # arbitrarily use event flag number 2:
    efn                 =   2;
INTEGER secsToWaitThisTime;
ADDRESS dayTim;
```

Example 44.2-5. Calling $SETIMR and $WAITFR from MAINSAIL (continued)

```
# Allocate space for the quadword to contain the time
# interval to wait, and store a -1 in the high order
# longword.
dayTim := newScratch(dayTimBufSize);
store(dayTim,-1L,longWordSize);
# A time interval of more than 214 seconds requires more
# than a longword to represent, so break long intervals
# into 214-second chunks.
WHILE secondsToWait > 0 DOB
    # Calculate the length of time to wait this iteration
    # through the loop, and convert the time from seconds
    # to the delta time in tenths of microseconds, storing
    # the result in the quadword's low order longword.
    secsToWaitThisTime := secondsToWait MIN maxTimeInSecs;
    store(dayTim,deltaTicksPerSec
        * cvli(secsToWaitThisTime));
    # Call the $SETIMR and $WAITFR system services.
    $sysCall(sysSeTimr,cvlb(efn),cvlb(dayTim),'0L,'0L);
    $sysCall(sysWaitFr,cvlb(efn));
    # Calculate the remaining length of time to wait.
    secondsToWait .- secsToWaitThisTime;
    END;
# Dispose of the quadword.
scratchDispose(dayTim);
END;
```

Example 44.2-5. Calling $SETIMR and $WAITFR from MAINSAIL (end)

45. File System

MAINSAIL correctly processes VAX/VMS variable (RMS) files and files with fixed-length records and embedded line feeds delimiting lines (byte stream files). The standard MAINSAIL view of a text file is a stream of bytes with lines separated by eol (eol = ASCII linefeed = 10 decimal).

45.1. Sequential Text Input File Formats

The input records of a record-oriented text file (RMS format) are automatically translated by the MAINSAIL runtime system into the MAINSAIL byte stream format, which is then made available to the user's program. Thus, the user program need not concern itself with the VAX/VMS concept of records except to ensure that the proper translation is applied. Table 45.1-1 describes the translation algorithm for each device prefix.

Device Prefix	Translation Algorithm
BS	Byte stream file. No translation takes place.
VAR	Variable-length records (RMS format). eol is appended.

Table 45.1-1. Input Translation Rules for Text Files

If no device prefix is specified, the format of the VAX/VMS file determines which device prefix is used, as shown in Table 45.1-2.

VAX/VMS Format	Device Prefix Used
Unstructured	BS
Variable (RMS)	VAR

Table 45.1-2. Default Device Prefixes for Sequential Text Input

45.2. Sequential Text Output File Formats

The behavior of the available device prefixes for a text file opened for sequential output is shown in Table 45.2-1. Here, "n" indicates an optional record size.

Device Prefix	Output Format
BS	Byte stream file.
BS(n)	Byte stream file, record size = n.
VAR(n)	Variable-length records of maximum size n, maximum supported n = 512 (VAX/VMS RMS format). A new output record is started when either n bytes are output or an eol character is output. In the latter case, the eol is discarded. Records are not padded with blanks.

Table 45.2-1. Device Prefixes for Sequential Text Output

If a device prefix is specified, the file is created according to the device specification. If a device prefix is not specified, then the device prefix "VAR(512)>" is used; i.e., the default format for sequential text output files is VAX/VMS RMS format with maximum record length 512 characters.

45.3. Sequential Data File Formats

Any data file can be opened for sequential input. Sequential output data files must be byte stream format.

45.4. Random File Formats

Any file can be opened for random input. Random text output files must be byte stream format; random data output files must be byte stream format.

45.5. Converting Between File Formats

The MAINSAIL utility module COPIER can be used to convert between file formats by explicitly specifying the device prefix for the output file. Refer to the "MAINSAIL Utilities User's Guide" for a complete description of this utility.

Example 45.5-1 shows how to use COPIER to convert an RMS file to a byte stream file. COPIER copies the RMS format file "rmsFile" to the byte stream file "bsFile". The device prefix "BS>" must be specified; otherwise, MAINSAIL creates a variable format file (the default for sequential text output files).

```
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*copier<eol>
Text File Copier
Input file (just <eol> to stop): rmsFile<eol>
Output file: BS>bsFile<eol>
Input file (just <eol> to stop): <eol>
*
```

Example 45.5-1. COPIER Example

46. System Information Procedures

46.1. $currentDirectory

$currentDirectory returns the name of the current directory, in square brackets, as obtained from the VAX/VMS system service $PARSE.

46.2. $homeDirectory

$homeDirectory returns the string found by the system call $TRNLNM in the table "LNM$JOB" under the logical name "SYS$LOGIN".

46.3. Command Line

The command line is returned by the VAX/VMS library procedure LIB$GET_FOREIGN. In order to specify a command line to DCL, you must use DCL to create a foreign command for your MAINSAIL bootstrap, then invoke the bootstrap with the foreign command. For example, for a bootstrap file "foo.exe", you must do:

```
$ foo :== $foo.exe<eol>
```

The command "foo" can now be invoked with arguments, e.g.:

```
$ foo arg1 arg2 arg3<eol>
```

LIB$GET_FOREIGN converts all arguments to uppercase, so MAINSAIL would see the command line arguments as "ARG1 ARG2 ARG3".

46.4. $programName

The program name is unavailable on VAX/VMS, so $programName is always the null string.

46.5. $directory (for VAX/VMS Disk Files)

$directory makes use of the VAX/VMS system services $PARSE and $SEARCH. If $reportAllVersions is set, all versions of files are returned; otherwise, only the most recent is returned. If $fullPathNames is set, the directory specification is included in the file name.

46.6. $fileInfo (for VAX/VMS Disk Files)

$fileInfo fills in all fields of $fileInfoCls.

46.7. $userID

$userId returns the string that is the user name of the current user, trailing blanks removed.

46.8. $cpuID

On an 11/780, 11/785, or 8600, $cpuID returns the CPU serial number as a decimal string. The serial number is obtained from the low-order 11 bits of the SID. On any other model, $cpuID returns the null string.

On VAX/VMS, if $cpuID would return the null string, it checks to see if $useAlternateCpuID ('H200) is set in the configuration bits. If so, then it returns whatever is produced by the FLI procedure $alternateCpuID. This procedure must be provided by the user in an FLI module called $aCpuID, declared as follows:

```
MODULE $aCpuID (

INTEGER
PROCEDURE   $alternateCpuID
                              (CHARADR buf; INTEGER bufSize);

);
```

$alternateCpuID returns a null-terminated string no longer than bufSize at buf. $aCpuId should be compiled with the appropriate Foreign Call Compiler and linked with a MAINSAIL bootstrap in which the 'H200 bit was set in the "CONFIGURATIONBITS" command.

46.9. Exit Codes

When MAINSAIL exits, the value of the exit code is pased to SYS$EXIT. $successExitCode is '1L and $failureExitCode is '0L.

47. Shared Module Libraries

VAX/VMS provides a method for several users to share a single copy of a file called a global section. When a global section is in memory, new users of the section will access this copy rather than reading a new copy of their own. If heavily used libraries (such as the file "SYSTEM.LIB") are made global sections, memory use and I/O overhead are reduced.

If a global section for a library has been installed, MAINSAIL automatically sets up access to the global section when the library is opened.

47.1. Global Section Installation

Global sections are installed with the MAINSAIL module GBLSEC. Available GBLSEC commands are listed in Table 47.1-1. Only enough of a command to make it uniquely identifiable need be typed.

Command	Meaning
CHECKGLOBALSECTION s	See if global section with name s exists.
CREATEGLOBALSECTION f	Create global section for file f.
DELETEGLOBALSECTION s	Delete global section named s. Note that this has no effect on the file associated with it.
REPLACEGLOBALSECTION f	Replace global section for file f. If a new version of a library is created, the global section must be updated with this command.

Table 47.1-1. GBLSEC Commands

A global section has associated with it both a file and a name. This name is a one- to fifteen-character case-sensitive string. When a global section is created with the "CREATEGLOBALSECTION" command, the name is derived from the root of the specified file name. For example, the global section name derived from "ms:system.lib" is "system", and that derived from "MS:SYSTEM.LIB" is "SYSTEM". Note that since the global section name is case-sensitive, "system" and "SYSTEM" are distinct global sections that reference the same file. When VAX/VMS derives a global section name from a file name, it converts it to lower case. For this reason, all file names should be entered in lower case.

47.2. Example of Creating a Global Section

```
MAINSAIL version x (? for help)
*gblsec<eol>
Next Command (? for help): check system<eol>
Section system does not already exists
<note: check case of sectionName>

Next Command (? for help): create ms:system.lib<eol>
Section system does not already exists
<note: check case of sectionName>

command succesfully completed.

Next Command (? for help): check system<eol>
Section system already exists
<note: check case of sectionName>

Next Command (? for help): check SYSTEM<eol>
Section SYSTEM does not already exists
<note: check case of sectionName>

Next Command (? for help): exit<eol>
*
```

Example 47.2-1. Creating a Global Section

47.3. Caveat

All global sections must be reinstalled if the system crashes.

Care must be taken when running more than one version of MAINSAIL. If a global section has been made for "system.lib", that global section is used regardless of the bootstrap that is run. Thus, the wrong runtime system may be used inadvertently. If "system.lib" is a global section and several versions of MAINSAIL are in use at your installation, the global section must be renamed. To do this, follow the steps shown in Figure 47.3-1 for each version of MAINSAIL that is installed.

```
1. Edit the file "VMS.CNF", and change the name of the
   system library ("ms:system.lib").  The new name should
   have the version number embedded within it; e.g., use
   "ms:sys88.lib" for version 8.8.

2. Using CONF and the new "VMS.CNF", create a new
   bootstrap.

3. Rename "ms:system.lib" to the new name.

4. Assemble and link the new bootstrap as described in the
   installation procedure.

5. Install the new system library as a global section.
```

Figure 47.3-1. Installing a Global Section

48. Foreign Language Interface

This chapter contains information about the MAINSAIL Foreign Language Interface (FLI) specific to VAX/VMS. Refer to the "MAINSAIL Compiler User's Guide" for a general description of the FLI.

Calls into and out of MAINSAIL are supported.

48.1. FLI Compiler Output File Names

The default name for the output file created by the FLI compilers is "<module name>.MAR".

48.2. VAX-11 Procedure Calling Standard

The VAX/VMS Foreign Call Compiler for the VAX-11 procedure calling standard is used to call from MAINSAIL to foreign language procedures that conform to the standard promoted by DEC. All compilers supported by DEC for the VAX-11 produce procedure interfaces that follow this standard.

The procedure calling standard governs:

- the way a procedure is called,
- the way it accesses its parameters,
- the way it returns to its caller,
- the way it returns a result (if it has one).

The standard does not govern whether parameters are passed by value or by address, nor does it specify the amount of space occupied by parameters passed by value, although it does require that each parameter passed by value occupy a multiple of 4 bytes. The choice of whether parameters are passed by address or value and the amount of space occupied by value parameters depends on the particular compiler being used. The VAX-11 calling standard FLI is invoked with the MAINSAIL compiler's "FLI TV" ("To Vax") subcommand.

48.2.1. Passing Parameters

In calls from MAINSAIL to a foreign language procedure, modifies and produces parameters are passed by address. Addresses occupy 4 bytes each. Uses parameters are passed by value, and are padded if necessary to occupy a multiple of 4 bytes. The amount of space occupied by a value (without padding) depends on the value's type. Section 49.2 describes the way values of each data type are represented on the VAX-11.

Produces parameters are initialized to Zero before the call to the foreign language procedure is made.

Values of any MAINSAIL data type may be passed as uses parameters to a foreign language procedure. However, pointers, strings, and arrays may not be passed as produces parameters, and neither pointers nor arrays may be passed as modifies parameters. Strings may be passed as modifies parameters so that they will be passed by address, but they should not be modified by the foreign language procedure.

48.2.2. String Parameters

VAX-11 string descriptors promoted by DEC occupy 8 bytes each, and are represented as shown in Figure 48.2.2-1.

```
+-------+-------+---------------+
| class | type  |    length     |
+-------+-------+---------------+
|  address of first character   |
+-------------------------------+
```

Figure 48.2.2-1. VAX-11 String Descriptor Used by DEC

The "class" field occupies one byte and indicates the kind of descriptor; zero means "unspecified class".

The "type" field occupies one byte and indicates the data type of the string's components; zero means "unspecified data type".

MAINSAIL string descriptors are identical in format to VAX-11 string descriptors with class and type fields both equal to zero.

If a string is passed as a uses parameter to a foreign language procedure, the string's MAINSAIL string descriptor is passed as the parameter. If the string is passed as a modifies parameter, the address of the string descriptor is passed.

Many foreign language procedures that take a string parameter expect to be passed the address of a VAX-11 string descriptor, even though they do not modify the descriptor and thus need only the descriptor's value. For this reason, the VAX/VMS Foreign Calls Compiler allows strings to be passed as modifies parameters to a foreign language procedure. The foreign language procedure should not modify the string descriptor. If it does, then MAINSAIL may crash or produce invalid results.

48.2.3. Array Parameters

When an array is passed as a uses parameter to a foreign language procedure, the address of the array's first element is passed. Thus, the foreign language procedure may change the array's elements, but it has no access to the array descriptor. If the array's upper bounds are not known within the foreign language procedure, they should be passed explicitly as additional parameters. A nullArray is passed as 0 (i.e., nullAddress).

48.2.4. Caveat

When dealing with values subject to garbage collection (i.e., strings, records, and arrays), care must be taken not to violate the data boundaries. Writing beyond the boundaries of these structures may cause MAINSAIL to crash. If, for example, a garbage collection link is destroyed, MAINSAIL may run until a collection is triggered, at which point the results are unpredictable.

48.3. C Procedure Calling Standard

The VMS to C FLI is identical to the VAX standard calling sequence with the following exceptions:

1. Only the charadr portion of a string is passed to C. The caller must guarantee that the string is null terminated.

2. Uses real parameters are converted to long real. This is because C passes only long data types on the stack.

3. The default label for a C procedure is the MAINSAIL procedure name converted to lower case and prefixed by "_" (underscore).

4. The C FLI is invoked using the MAINSAIL compiler's "FLI TC" subcommand.

48.4. Identifiers Containing "$ or "_"

The identifier generated for a foreign call is the same as the procedure name, unless the procedure name contains dollar signs or underscores. Dollar signs are suppressed by the FLI compiler and underscores are not allowed in MAINSAIL identifiers. The "ENCODE" directive to the compiler permits an arbitrary string to be used as the label for the foreign call. Example 48.4-1 shows how to call the routine "sys$setimr".

```
MODULE forCal (
        PROCEDURE sysSetTimr;
        );

ENCODE sysSetTimr "sys$setimr";
```

Example 48.4-1. Calling a Routine with "$" in Its Name

In the file "FORCAL.MAR", the label "sys$setimr" is used for the call to the procedure sysSetTimr.

VAX/VMS system services may also be accessed by means of the VAX/VMS-dependent procedure $sysCall, described in Chapter 44.

48.5. Foreign Language Interface Example

Suppose that the MAINSAIL module FOOSUB is to call the FORTRAN subroutine FTNADD. The text for FTNADD is in the file "FTNSUB". Example 48.5-1 shows FTNADD, Example 48.5-2 shows the MAINSAIL FLI module that defines this foreign procedure, and Example 48.5-3 shows the MAINSAIL module FOOSUB.

```
      SUBROUTINE FTNADD (I,J,K)
C     ADD J AND K, RETURNING THE RESULT IN I.
      I = J + K
      RETURN
      END
```

Example 48.5-1. Sample FORTRAN Subroutine

```
BEGIN "fliSub"

# All calls from MAINSAIL to FTNADD pass through
# this FLI module.

# FORTRAN expects to be passed the address of each
# parameter, so the FLI declaration of ftnAdd must
# declare each parameter as either MODIFIES or
# PRODUCES (PRODUCES only if the parameter's input
# value is not used by the foreign procedure).  A
# FORTRAN integer occupies 4 bytes, the equivalent
# of a MAINSAIL long integer on the VAX-11.

MODULE fliSub
    (PROCEDURE ftnAdd (PRODUCES LONG INTEGER i;
                       MODIFIES LONG INTEGER j,k));

# The body for every interface procedure in a
# module must appear somewhere in that module:

PROCEDURE ftnAdd (PRODUCES LONG INTEGER i;
                  MODIFIES LONG INTEGER j,k);;

END "fliSub"
```

Example 48.5-2. Sample FLI Module

```
BEGIN "fooSub"

MODULE fliSub
    (PROCEDURE ftnAdd (PRODUCES LONG INTEGER i;
                       MODIFIES LONG INTEGER j,k));

INITIAL PROCEDURE;
BEGIN
LONG INTEGER i,j,k;
j := 1L; k := 2L;
ftnAdd(i,j,k);
ttyWrite("I = ",i,eol);
END;

END "fooSub"
```

Example 48.5-3. MAINSAIL Module Calling FORTRAN Module

Example 48.5-4 show the steps that must be taken in order to run FOOSUB. FOOSUB is compiled with the MAINSAIL VAX/VMS compiler. The module FLISUB is compiled with the subcommand "FLI TV", and the resulting assembly language file, "FLISUB.MAR", is assembled with the VAX/VMS assembler. The FORTRAN subroutine FTNADD is compiled with the VAX/VMS FORTRAN compiler. A new bootstrap assembly language file, "FLIMAINSA.MAR", is made by running the MAINSAIL utility module CONF, and the new bootstrap is assembled. The new bootstrap is linked with the FLI code and the FORTRAN object module to create an executable file, "FLIMAINSA.EXE". MAINSAIL is run by invoking "FLIMAINSA.EXE", and FOOSUB is run.

```
$ r ms:mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
compile (? for help): foosub<eol>
foosub 1 ...

Output for FOOSUB on foosubvms.obj

compile (? for help): flisub,<eol>
>fli tv<eol>
><eol>
flisub 1 ...

Output for FLISUB on FLISUB.MAR
```

Example 48.5-4. Using the Foreign Language Interface (continued)

```
compile (? for help): <eol>
*<eol>
$ macro flisub<eol>
$ fortra ftnsub<eol>
$ r ms:mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ms:vms.cnf
CONF: foreignmodule flisub<eol>
CONF: bootfilename flimainsa.mar<eol>
CONF: <eol>
Bootstrap written in file flimainsa.mar
*<eol>
$ macro flimainsa<eol>
$ link flimainsa,flisub,ftnsub<eol>
$ r flimainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*foosub<eol>
I = 3
*
```

Example 48.5-4. Using the Foreign Language Interface (end)

Example 48.5-5 and Table 48.5-6 show some code written in C and the FLI module used to interface to it.

```
struct foo {
    int x1;
    float f1;
    };

int csample (ar,ub,s,scale,r)
int ar[];
int ub;
char *s;
float scale;
struct foo *r;
{
int i,tot;

/* print the string we passed in */
printf("The string is %s\n",s);
/* sum the elements of the array.  Note that arrays are
   all 0 origin in C */
tot = 0;
for (i = 0; i < ub; i++) tot += ((r->x1 + ar[i]) / r->f1);
/* scale by our scaling factor */
tot *= scale;
return(tot);
}
```

Example 48.5-5. Sample C Subroutine

```
BEGIN "cexamp"

# ints in C are LONG INTEGERs in MAINSAIL.  Since we are
# using the C FLI, the string will automatically be
# converted a CHARADR.  REALs, although converted to LONG
# REAL on the stack, are still passed in as REALs

CLASS fooCls (
    LONG INTEGER x1;
    REAL f1);


MODULE cexamp (
    LONG INTEGER
    PROCEDURE csample (LONG INTEGER ARRAY (0 TO *) ar;
                       LONG INTEGER ub;
                       STRING s;
                       REAL scale;
                       POINTER(fooCls) r);
    );

# VAX C will internally call the procedure "_csample",
# so the following ENCODE directive is needed.

ENCODE csample "_csample";

LONG INTEGER
PROCEDURE csample (LONG INTEGER ARRAY (0 TO *) ar;
                   LONG INTEGER ub;
                   STRING s;
                   REAL scale;
                   POINTER(fooCls) r);
;

END "cexamp"
```

Table 48.5-6. FLI Module Used to Interface to Code in Example 48.5-5

Example 48.5-7 shows MAINSAIL code that uses this procedure.

```
BEGIN "cSub"

CLASS fooCls (
    LONG INTEGER x1;
    REAL f1);

MODULE cexamp (
    LONG INTEGER
    PROCEDURE csample (LONG INTEGER ARRAY (0 TO *) ar;
                       LONG INTEGER ub;
                       STRING s;
                       REAL scale;
                       POINTER(fooCls) r);
    );

INITIAL PROCEDURE;
BEGIN
LONG INTEGER ARRAY (0 TO *) tots;
LONG INTEGER val;
STRING s;
POINTER(fooCls) p;

new(tots,1,500);
p := new(fooCls);
p.x1 := 10;
p.f1 := 2.0;

# the procedure initMyArray initializes the elements of
# tots, and may change the upper boundry.

initMyArray(tots);

# set the string we will pass in.  Note that we must
# null-terminate it.

s := "Array tots" & cvcs(0);

# now call the C routine
```

Example 48.5-7. MAINSAIL Module Calling C Example (continued)

```
val := csample(tots,cvli(tots.ub1),s,.5,p);
write(logFile,"Value of csample is ",val,eol);
END;

END "cSub"
```

Example 48.5-7. MAINSAIL Module Calling C Example (end)

Example 48.5-8 shows the steps required to compile and link the C example. This example assumes that the C code is in the file "cproc.c", the FLI module is "cexamp.msl", and the module that invokes the C procedure is "csub.msl". The compiler subcommand to invoke the C FLI is "FLI TC". The command:

```
define lnk$library sys$library:vaxcrtl.olb
```

is required by C.

```
$ r ms:mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
compile (? for help): csub.msl<eol>
csub.msl 1 ...

Output for CSUB on csubvms.obj

compile (? for help): cexamp.msl,<eol>
>fli tc<eol>
><eol>
cexamp.msl 1 ...
```

Example 48.5-8. Using the C Foreign Language Interface (continued)

```
Output for CEXAMP on CEXAMP.MAR

compile (? for help): <eol>
*<eol>
$ macro csub<eol>
$ cc cproc<eol>
$ r ms:mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ms:vms.cnf
CONF: foreignmodule cexamp<eol>
CONF: bootfilename flimainsa.mar<eol>
CONF: <eol>
Bootstrap written in file flimainsa.mar
*<eol>
$ define lnk$library sys$library:vaxcrtl.olb<eol>
$ macro flimainsa<eol>
$ link flimainsa,cexamp,cproc<eol>
$ r flimainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*csub<eol>
The string is Array tots
The value of csample is 24
*
```

Example 48.5-8. Using the C Foreign Language Interface (end)

48.6. MAINSAIL Entry Compiler

MAINSAIL procedures can be invoked from foreign code by using the MAINSAIL Entry Compiler (MEC). As with Foreign Call Compiler, entry to MAINSAIL can be made either from the VAX-11 standard calling convention, or from C. The VAX standard MEC is invoked using the MAINSAIL compiler's "FLI FV" (From VAX) subcommand. The C MEC is invoked with the "FLI FC" subcommand.

If a foreign-language program initiates execution instead of MAINSAIL, the configuration bit $foreignCodeStartsExecution must be set in the MAINSAIL bootstrap. Consult the description of the FLI in "MAINSAIL Compiler User's Guide" and the description of CONF in the "MAINSAIL Utilities User's Guide" for details. $foreignCodeStartsExecution is supported for both the "FV" and "FC" MAINSAIL entry compilers. Currently, the bootstrap produced by CONF still has a transfer address even when $foreignCodeStartsExecution is set, which causes the linker to complain about multiple transfer addresses. If the module in which control is to originate is specified first in the "link" command list, it receives control first, as desired.

48.7. MAINSAIL Entry Compiler Example

This example uses the C entry compiler. Example 48.7-1 and Table 48.7-2 show a sample MAINSAIL module that contains a procedure to be invoked from C and the C code that calls it.

```
BEGIN "msproc"
# this is file msproc.msl

MODULE msproc (
        LONG INTEGER
        PROCEDURE mproc (REAL a, b, c);
        );

ENCODE mproc "_mproc";

LONG INTEGER PROCEDURE mproc (REAL a, b, c);
RETURN(cvli((a + b) / c));

END "msproc"
```

Example 48.7-1. MAINSAIL Module to Be Called by C

```
cproc () {
/* this is file cproc.c */
float a,b,c;
int i;
a = 15.0;
b = 13.0;
c = 2.0;

i = mproc(a,b,c);
printf("The value of mproc is %d\n",i);

}
```

Table 48.7-2. C Code to Call a MAINSAIL Module

A call is needed to start the C program. This code and the FLI module required to invoke the C call are shown in Examples 48.7-4 and 48.7-3.

```
BEGIN "ccode"

# this is file ccode.msl

MODULE ccode (
    PROCEDURE cproc;
    );

PROCEDURE cproc;;

END "ccode"
```

Example 48.7-3. FLI Module to Invoke C Call

```
BEGIN "ccall"

# this is file ccall.msl

MODULE ccode (
    PROCEDURE cproc;
    );

INITIAL PROCEDURE;
cproc;

END "ccall"
```

Example 48.7-4. MAINSAIL Code to Start C Execution

To use the MEC, take the following steps:

1. The entry module, MSPROC, is compiled with the MAINSAIL compiler to produce the MAINSAIL module.

2. The entry module is compiled with the "FLI FC" subcommand to the MAINSAIL compiler, to produce the file "MSPROC.MAR", which contains the C entry code.

3. The module "CCODE" is compiled with the "FLI TC" subcommand, to produce the C FLI to invoke the main C program.

4. The module "CCALL" is compiled with no subcommands to produce the main MAINSAIL module.

5. CONF is run to produce a boot, "mecmainsa.mar", which knows about the CCODE FLI.

6. The files "mecmainsa.mar", "msproc.mar", and "ccode.mar" are assembled with MACRO.

7. The file "cproc.c" is compiled with the C compiler.

8. "mecmainsa", "msproc", "cproc", and "ccode" are linked.

9. Run "mecmainsa" and invoke the module "ccall". C executes and calls back into MAINSAIL.

This entire example is shown in Example 48.7-5. The line:

```
define lnk$library sys$library:vaxcrtl.olb
```

is required by C.

```
$ r ms:mainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*compil<eol>
MAINSAIL (R) Compiler
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
compile (? for help): msproc.msl,<eol>
msproc.msl 1 ...

Output for MSPROC on MSPROCVMS.OBJ

compile (? for help): msproc.msl,<eol>
>fli fc<eol>
><eol>
msproc.msl 1 ...

Output for MSPROC on MSPROC.MAR

compile (? for help): ccode.msl,<eol>
>fli tc<eol>
><eol>

ccode.msl 1 ...

Output for CCODE on CCODE.MAR

compile (? for help): ccall.msl,<eol>
>nofli<eol>
><eol>

ccall.msl 1

Output for CCALL on CCALLVMS.OBJ
```

Example 48.7-5. Using the MAINSAIL Entry Compiler (continued)

```
compile (? for help): <eol>
*conf<eol>
MAINSAIL (R) Bootstrap Configurator
Restoring configuration values from file ms:vms.cnf
CONF: foreignmodule ccode<eol>
CONF: bootfilename mecmainsa.mar<eol>
CONF: <eol>
Bootstrap written in file mecmainsa.mar
*<eol>
$ macro msproc<eol>
$ macro ccode<eol>
$ macro mecmainsa<eol>
$ cc cproc<eol>
$ define lnk$library sys$library:vaxcrtl.olb<eol>
$ link mecmainsa,cproc,msproc,ccode<eol>
$ r mecmainsa<eol>
MAINSAIL (R) Version 12.10 (? for help)
Copyright (c) 1984, 1985, 1986, 1987, 1988, and 1989 by
 XIDAK, Inc., Menlo Park, California, USA.
*ccall<eol>
The value of mproc is 14
*
```

Example 48.7-5. Using the MAINSAIL Entry Compiler (end)

49. VAX-11 Processor-Dependent Information

This chapter contains information about MAINSAIL that is specific to the VAX-11 implementations.

49.1. Procedure Size

There is no well-defined limit for the size of a procedure on the VAX-11. However, the compiler may not compile procedures longer than 32K bytes, depending on the code. Procedures longer than approximately 32K bytes are not guaranteed to work, and such procedures may compile correctly on one machine and not on another.

49.2. VAX-11 Data Types

Refer to Table 49.2-1. A storage unit on the VAX-11 is one byte (8 bits).

Data Type	Representation
boolean	1 word (2 bytes)
integer	1 word (2 bytes)
long integer	1 longword (4 bytes)
real	single precision F_floating (4 bytes)
long real	double precision D_floating (8 bytes)
bits	1 word (2 bytes)
long bits	1 longword (4 bytes)
string	1 quadword (8 bytes) low-address longword is length; high-address longword is charadr of first character
address	1 longword (4 bytes)
charadr	1 longword (4 bytes)
pointer	1 longword (4 bytes)

Table 49.2-1. VAX-11 Data Types

49.3. Miscellaneous Information

The standard representation for boolean FALSE is all bits clear, and the standard for boolean TRUE is low-order bit set, all other bits clear. However, in constructs such as "IF <boolean value> THEN ...", <boolean value> is considered to be TRUE if any bits are set.

String variables have both the length and charadr component equal to Zero for the string Zero (no characters).

50. Miscellaneous

50.1. CTRL-C

MAINSAIL traps CTRL-C. It prompts with "Yes (? for help):". Possible answers are "q" to quit MAINSAIL, "b" to enter the debugger at the next debuggable procedure, if possible, and anything else to continue. MAINSAIL does not trap CTRL-Y.

50.2. Event Flags

EFN 0 is used for terminal I/O.

50.3. Exceptions

The MAINSAIL error handler is designed to intercept all VAX/VMS error exceptions. These exceptions, and the VAX/VMS symbolic name for each, are listed in Table 50.3-1. If, during the execution of the error routine, another error exception occurs, the message "Error while processing an error" is given, and execution immediately terminates. More details on exceptions can be found in the "VAX/VMS System Services Reference Manual" supplied by DEC.

SS$_ACCVIO	Access Violation
SS$_ARTRES	Reserved Arithmetic Trap
SS$_BREAK	Breakpoint Instruction Encountered
SS$_CMODSUPR	Change mode to Supervisor Encountered
SS$_CMODUSER	Change mode to User Encountered
SS$_DECOVF	Decimal overflow
SS$_FLTDIV	Floating/Decimal divide by zero
SS$_FLTOVF	Floating Overflow
SS$_FLTUND	Floating Underflow
SS$_INTDIV	Integer divide by zero
SS$_INTOVF	Integer Overflow
SS$_OPCCUS	Opcode reserved to Customer
SS$_OPCDEC	Opcode reserved to Digital
SS$_PAGRDERR	Read error occured during during an attempt to read a faulted disk page
SS$_RADRMOD	Attempt to use a reserved addressing mode
SS$_ROPRAND	Attempt to use a reserved operand

Table 50.3-1. Exception Conditions Caught by MAINSAIL

XIDAK, Inc., 530 Oak Grove Avenue, M/S 101, Menlo Park, CA 94025, (415) 324-8745