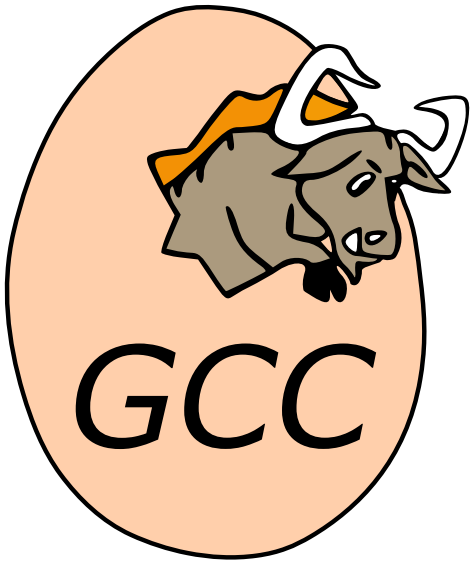

The GNU Modula-2 Compiler

Release 13.0.1 (experimental 20220101)

Gaius Mulley

Nov 07, 2022

CONTENTS

CHAPTER

ONE

WHAT IS GNU MODULA-2

GNU Modula-2 is a front end for the GNU Compiler Collection (GCC). The GNU Modula-2 compiler is compliant with the PIM2, PIM3, PIM4 and ISO dialects. Also implemented are a complete set of free ISO libraries and PIM libraries.

The four Modula-2 dialects supported are defined in the following references:

PIM2: 'Programming in Modula-2', 2nd Edition, Springer Verlag, 1982, 1983 by Niklaus Wirth (PIM2).

PIM3: 'Programming in Modula-2', 3rd Corrected Edition, Springer Verlag, 1985 (PIM3).

PIM4: 'Programming in Modula-2', 4th Edition, Springer Verlag, 1988 (PIM4).

ISO: the ISO Modula-2 language as defined in 'ISO/IEC Information technology - programming languages - part 1: Modula-2 Language, ISO/IEC 10514-1 (1996)'

CHAPTER

TWO

WHY USE GNU MODULA-2

There are a number of advantages of using GNU Modula-2 rather than translate an existing project into another language.

The first advantage is of maintainability of the original sources and the ability to debug the original project source code using a combination of gm2 and gdb.

The second advantage is that gcc runs on many processors and platforms. gm2 builds and runs on powerpc64le, amd64, i386, aarch64 to name but a few processors.

The compiler provides semantic analysis and runtime checking (full ISO Modula-2 checking is implemented) and there is a plugin which can, under certain conditions, detect runtime errors at compile time.

gm2 can produce swig interface headers to allow access from Python and other scripting languages. The compiler supports PIM2, PIM3, PIM4 and ISO dialects of Modula-2, work is underway to implement M2R10. Many of the GCC builtins are available and access to assembly programming is achieved using the same syntax as that used by GCC.

CHAPTER

THREE

RELEASE MAP

GNU Modula-2 is now part of GCC and therefore will adopt the GCC release schedule with a git branches matching the various GCC release numbers. It is intended that GNU Modula-2 implement more of the GCC builtins (vararg access) and GCC features.

There is an intention to implement the M2R10 dialect of Modula-2 and any of the language changes. If you wish to see something different please email gm2@nongnu.org with your ideas.

CHAPTER

FOUR

NEWS

gm2 release numbers
: are now superceded by gcc branch numbers as gm2 is tracking the gcc release model.

accuracy of error messages
: has been improved and offending subexpressions are now highlighted.

new type checker
: has been implemented which will check all data types, including recursive procedure parameter types.

libraries
: the gm2 driver program has been enhanced to allow third party libraries to be installed alongside gm2 libraries. For example if the user specifies library `foo` using `-flibs=foo` the driver will check the standard GCC install directory for a subdirectory `foo`.

libpth
: has been removed from gm2 and the coroutines are now implemented using the gcc portable threading library.

libulm
: the Ulm libraries have been removed from the gm2 tree as they were not GPL3.

Talk given at The GNU Tools Cauldron 2018
: here is a talk on GNU Modula-2 given at The GNU Tools Cauldron, Manchester on 8th September 2018. The title is, 'GNU Modula-2 update, catching semantic errors post code optimisation and improved debugging' [slides and video].

gm2 1.8.2
: was released on Aug 30th 2018. gm2-1.8.2 grafts onto gcc-8.2.0 and contains integer overflow detection for addition, subtraction, negation and multiplication. It also detects and traps when a floating point nan occurs. This is the first release with the new semantic checking plugin which checks whether any exception will occur post optimization (see -fsoft-check-all). The compiler also works well with the automake tools.

gm2 1.2.0
: was released on May 11th 2017. gm2-1.2.0 grafts onto gcc-5.2.0 and supports much better line number accuracy in debugging output. Source to code relationship can be further improved by the new option -fm2-g. -fm2-whole-program also provides whole program optimization.

Talk given at The GNU Tools Cauldron 2016
: here is a talk on GNU Modula-2 given at The GNU Tools Cauldron, Hebden Bridge on 9th September 2016. The title is, 'GNU Modula-2 status, whole program optimisation and language interoperability' [slides and video].

gm2 1.1.6
: was released on February 22nd 2016. gm2-1.1.6 grafts onto gcc-4.7.4.

gm2 1.1.5
: was released on September 3 2015, passes all regression tests and has many bug fixes applied. Arrays and Records can be assigned to and from WORD, LOC, BYTE providing sizes permit. Also a small number of fixes to the library module MemStream.mod. Fixed a number of bugs shown by valgrind.

gm2 1.1.3
: was released on April 15 2015. gm2-1.1.3 passes all regression tests on Debian Wheezy (x86_64) and (i686). Also passes all regression tests under Debian Jessie (x86_64). It also builds on armv7l Ubuntu Trusty Tahr.

gm2 1.1.1
: was released on January 26 2015. gm2-1.1.1 passes all regression tests on Debian Wheezy (x86_64) and (i686). Also passes all regression tests under Debian Jessie (x86_64).

gm2 1.1.0
: was released on January 02 2015. gm2-1.1.0 passes all regression tests on Debian Wheezy (x86_64) and (i686).

gm2 1.0.9
: Beta was released on September 23 2014, all regressions passed on x86_64 Debian Wheezy.

gm2 1.0.4
: was released on September 30 2011. This is a bug fix release.

gm2 1.0
: was released on December 11 2010.

CHAPTER

FIVE

HOW TO GET SOURCE CODE USING GIT

GNU Modula-2 is in the process of migrating into the GCC git tree. The development branch is available via git:

```
$ git clone git://gcc.gnu.org/git/gcc.git gcc-git-devel-modula2
$ cd gcc-git-devel-modula2
$ git checkout devel/modula-2
$ cd ..
```

GNU Modula-2 is in the process of migrating into the GCC git tree. See development section in this GM2 documentation for git details and See gcc:downloading-the-source in the GCC documentation.

CHAPTER

SIX

GNU MODULA-2 FEATURES

- the compiler currently complies with Programming in Modula-2 Edition 2, 3, 4 and ISO Modula-2. Users can switch on specific language features by using: `-fpim`, `-fpim2`, `-fpim3`, `-fpim4` or `-fiso`.
- the option `-fswig` will automatically create a swig interface file which corresponds to the definition module of the file being compiled.
- exception handling is compatible with C++ and swig. Modula-2 code can be used with C or C++ code.
- Python can call GNU Modula-2 modules via swig.
- shared libraries can be built.
- fixed sized types are now available from `SYSTEM`.
- support for dynamic `ARRAY` s has been added into `gdb`.
- variables can be declared at addresses.
- much better dwarf-2 debugging support and when used with `gdb` the programmer can display `RECORD` s, `ARRAY` s, `SET` s, subranges and constant char literals in Modula-2 syntax.
- supports sets of any ordinal size (memory permitting).
- easy interface to C, and varargs can be passed to C routines.
- many Logitech libraries have been implemented and can be accessed via: `-flibs=m2log,m2pim,m2iso`.
- coroutines have been implemented in the PIM style and these are accessible from SYSTEM. A number of supporting libraries (executive and file descriptor mapping to interrupt vector libraries are available through the `-flibs=m2iso,m2pim` switch).
- can be built as a cross compiler (for embedded microprocessors such as the AVR and the ARM).

CHAPTER

SEVEN

DOCUMENTATION

The GNU Modula-2 documentation is available on line at the gm2 homepage or in the pdf, info, html file format.

CHAPTER

EIGHT

REGRESSION TESTS FOR GM2 IN THE REPOSITORY

The regression testsuite can be run from the gcc build directory:

```
$ cd build-gcc
$ make check -j 24
```

which runs the complete testsuite for all compilers using 24 parallel invocations of the compiler. Individual language testsuites can be run by specifying the language, for example the Modula-2 testsuite can be run using:

```
$ cd build-gcc
$ make check-m2 -j 24
```

Finally the results of the testsuite can be emailed to the gcc-testresults list using the script:

```
$ gccsrcdir/contrib/test_summary
```

CHAPTER

NINE

LIMITATIONS

Logitech compatibility library is incomplete. The principle modules for this platform exist however for a comprehensive list of completed modules please check the documentation gm2.html.

CHAPTER

TEN

OBJECTIVES

- The intention of GNU Modula-2 is to provide a production Modula-2 front end to GCC.
- It should support all Niklaus Wirth PIM Dialects [234] and also ISO Modula-2 including a reimplementation of all the ISO modules.
- There should be an easy interface to C.
- Exploit the features of GCC.
- Listen to the requests of the users.

CHAPTER

ELEVEN

FAQ

11.1 Why use the C++ exception mechanism in GCC, rather than a bespoke Modula-2 mechanism?

The C++ mechanism is tried and tested, it also provides GNU Modula-2 with the ability to link with C++ modules and via swig it can raise Python exceptions.

CHAPTER

TWELVE

COMMUNITY

You can subscribe to the GNU Modula-2 mailing by sending an email to: gm2-subscribe@nongnu.org or by http://lists.nongnu.org/mailman/listinfo/gm2. The mailing list contents can be viewed http://lists.gnu.org/archive/html/gm2.

CHAPTER

THIRTEEN

OTHER LANGUAGES FOR GCC

These exist and can be found on the frontends web page on the gcc web site.

CHAPTER

FOURTEEN

PAPERS AND TALKS

A paper was presented at the GCC 2006 conference on the state of GNU Modula-2.

A lightening talk has been given at the GCC 2010 conference titled “Exploiting front end knowledge to effortlessly create Python modules” and the slides are available.

CHAPTER

FIFTEEN

USING GNU MODULA-2

15.1 Example compile and link

@c man begin SYNOPSIS gm2 gm2 [@option{-c}|@option{-S}] [@option{-g}] [@option{-pg}]

[@option{-O}@var{level}] [@option{-W}@var{warn}…] [@option{-I}@var{dir}…] [@option{-L}@var{dir}…] [@option{-f}@var{option}…] [@option{-m}@var{machine-option}…] [@option{-o} @var{outfile}] [@@@var{file}] @var{infile}…

Only the most useful options are listed here; see below for the remainder. @c man end @c man begin SEEALSO gpl(7), gfdl(7), fsf-funding(7), gcc(1) and the Info entries for @file{gm2} and @file{gcc}. @c man end The **gm2** command is the GNU compiler for the Modula-2 language and supports many of the same options as **gcc**. See Option Summary. This manual only documents the options specific to **gm2**.

This section describes how to compile and link a simple hello world program. It provides a few examples of using the different options mentioned in see Compiler options. Assuming that you have a file called `hello.mod` in your current directory which contains:

```
MODULE hello ;

FROM StrIO IMPORT WriteString, WriteLn ;

BEGIN
   WriteString('hello world') ; WriteLn
END hello.
```

You can compile and link it by: `gm2 -g hello.mod`. The result will be an `a.out` file created in your directory.

You can split this command into two steps if you prefer. The compile step can be achieved by: `gm2 -g -c -fscaffold-main hello.mod` and the link via: `gm2 -g hello.o`.

To see all the compile actions taken by `gm2` users can also add the `-v` flag at the command line, for example:

```
gm2 -v -g -I. hello.mod
```

This displays the subprocesses initiated by `gm2` which can be useful when trouble shooting.

15.2 Compiler options

This section describes the compiler options specific to GNU Modula-2 for generic flags details See GCC Command Options.

For any given input file, the file name suffix determines what kind of compilation is done. The following kinds of input file names are supported:

***file*.mod**
: Modula-2 implementation or program source files. See the `-fmod=` option if you wish to compile a project which uses a different source file extension.

***file*.def**
: Modula-2 definition module source files. Definition modules are not compiled separately, in GNU Modula-2 definition modules are parsed as required when program or implementation modules are compiled. See the `-fdef=` option if you wish to compile a project which uses a different source file extension.

  You can specify more than one input file on the **gm2** command line,

-g
: create debugging information so that debuggers such as `gdb` can inspect and control executables.

-I
: used to specify the search path for definition and implementation modules. An example is: `gm2 -g -c -I.:../../libs foo.mod`. If this option is not specified then the default path is added which consists of the current directory followed by the appropriate language dialect library directories.

-fdebug-builtins
: call a real function, rather than the builtin equivalent. This can be useful for debugging parameter values to a builtin function as it allows users to single step code into a real function.

-fdump-system-exports
: display all inbuilt system items. This is an internal command line option.

-fswig
: generate a swig interface file.

-fshared
: generate a shared library from the module.

-fruntime-modules=
: specify, using a comma separated list, the runtime modules and their order. These modules will initialized first before any other modules in the application dependency. By default the runtime modules list is set to `Storage,SYSTEM,M2RTS,RTExceptions,IOLink`. Note that these modules will only be linked into your executable if they are required. So adding a long list of dependant modules will not effect the size of the executable it merely states the initialization order should they be required.

-fnil
: generate code to detect accessing data through a `NIL` value pointer.

-fno-nil
do not generate code to detect accessing data through a `NIL` value pointer.

-fwholediv
generate code to detect whole number division by zero or modulus by zero.

-fno-wholediv
do not generate code to detect whole number division by zero or modulus by zero.

-findex
generate code to check whether array index values are out of bounds.

-fno-index
do not generate code to check whether array index values are out of bounds.

-frange
generate code to check the assignment range, return value range set range and constructor range.

-fno-range
do not generate code to check the assignment range, return value range set range and constructor range.

-freturn
generate code to check that functions always exit with a `RETURN` and do not fall out at the end.

-fcase
turns on compile time checking to check whether a `CASE` statement requires an `ELSE` clause when on was not specified.

-fsoft-check-all
turns on all runtime checks. This is the same as invoking GNU Modula-2 using the command options `-fnil -frange -findex`

`-fwholediv -fcase -freturn`.

-fauto-init
turns on auto initialization of pointers to NIL. Whenever a block is created all pointers declarated within this scope will have their addresses assigned to NIL.

-fno-exceptions
turns off all generation of exception handling code and no references are made to the runtime exception libraries.

-v
display all calls to subsidiary programs, such as the C preprocessor, the GNU Modula-2 linker and compiler.

-fm2-statistics
generates quadruple information: number of quadruples generated, number of quadruples remaining after optimisation and number of source lines compiled.

-fm2-whole-program
compile all implementation modules and program module at once. Notice that you need

to take care if you are compiling different dialect modules (particularly with the negative operands to modulus). But this option, when coupled together with `-O3`, can deliver huge performance improvements.

`-fm2-g`
improve the debugging experience for new programmers at the expense of generating `nop` instructions if necessary to ensure single stepping precision over all code related keywords. An example of this is in termination of a list of nested `IF` statements where multiple `END` keywords are mapped onto a sequence of `nop` instructions.

`-fm2-lower-case`
render keywords in error messages using lower case.

`fno-pthread`
do not automatically link against the pthread library. This option is likely useful if gm2 is configured as a cross compiler targetting embedded systems. By default GNU Modula-2 uses the GCC pthread libraries to implement coroutines (see the SYSTEM implementation module).

`-fuse-list=`*`filename`*
if `-fscaffold-static` is enabled then use the file `filename` for the initialization order of modules. Whereas if `-fscaffold-dynamic` is enabled then use this file to force linking of all module ctors. This option cannot be used if `-fgen-module-list=` is enabled.

`-fgen-module-list=`*`filename`*
attempt to find all modules when linking and generate a module list. If the `filename` is `-` then the contents are not written and only used to force the linking of all module ctors. This option cannot be used if `-fuse-list=` is enabled.

`-fscaffold-static`
the option ensures that `gm2` will generate a static scaffold within the program module. The static scaffold is useful for debugging and single stepping the initialization blocks of implementation modules.

`-fscaffold-dynamic`
the option ensures that `gm2` will generate a dynamic scaffold infastructure when compiling implementation and program modules. By default this option is on. Use `-fno-scaffold-dynamic` to turn it off or select `-fno-scaffold-dynamic`.

`-fcpp`
preprocess the source with `cpp -lang-asm -traditional-cpp` For further details about these options See Invocation. If `-fcpp` is supplied then all definition modules and implementation modules which are parsed will be preprocessed by `cpp`.

`-fiso`
turn on ISO standard features. Currently this enables the ISO `SYSTEM` module and alters the default library search path so that the ISO libraries are searched before the PIM libraries. It also effects the behaviour of `DIV` and `MOD` operators. See GNU Modula-2 supported dialects.

`-fpim`
turn on PIM standard features. Currently this enables the PIM `SYSTEM` module and determines which identifiers are pervasive (declared in the base module). If no other `-fpim[234]` switch

is used then division and modulus operators behave as defined in PIM4. See GNU Modula-2 supported dialects.

-fpim2
turn on PIM-2 standard features. Currently this removes `SIZE` from being a pervasive identifier (declared in the base module). It places `SIZE` in the `SYSTEM` module. It also effects the behaviour of `DIV` and `MOD` operators. See GNU Modula-2 supported dialects.

-fpim3
turn on PIM-3 standard features. Currently this only effects the behaviour of `DIV` and `MOD` operators. See GNU Modula-2 supported dialects.

-fpim4
turn on PIM-4 standard features. Currently this only effects the behaviour of `DIV` and `MOD` operators. See GNU Modula-2 supported dialects.

-fpositive-mod-floor-div
forces the `DIV` and `MOD` operators to behave as defined by PIM4. All modulus results are positive and the results from the division are rounded to the floor. See GNU Modula-2 supported dialects.

-flibs=
modifies the default library search path. The libraries supplied are: m2pim, m2iso, m2min, m2log and m2cor. These map onto the Programming in Modula-2 base libraries, ISO standard libraries, minimal library support, Logitech compatible library and Programming in Modula-2 with coroutines. Multiple libraries can be specified and are comma separated with precidence going to the first in the list. It is not necessary to use -flibs=m2pim or -flibs=m2iso if you also specify -fpim, -fpim2, -fpim3, -fpim4 or -fiso. Unless you are using -flibs=m2min you should include m2pim as the they provide the base modules which all other dialects utilize. The option `-fno-libs=-` disables the `gm2` driver from modifying the search and library paths.

-fextended-opaque
allows opaque types to be implemented as any type. This is a GNU Modula-2 extension and it requires that the implementation module defining the opaque type is available so that it can be resolved when compiling the module which imports the opaque type.

-fsources
displays the path to the source of each module. This option can be used at compile time to check the correct definition module is being used.

-fdef=
recognise the specified suffix as a definition module filename. The default implmentation and module filename suffix is `.def`. If this option is used GNU Modula-2 will still fall back to this default if a requested definition module is not found.

-fmod=
recognise the specified suffix as implementation and module filenames. The default implmentation and module filename suffix is `.mod`. If this option is used GNU Modula-2 will still fall back to this default if it needs to read an implmentation module and the specified suffixed filename does not exist.

-fxcode
issues all errors and warnings in the `Xcode` format.

-funbounded-by-reference
enable optimization of unbounded parameters by attempting to pass non VAR unbounded parameters by reference. This optimization avoids the implicit copy inside the callee procedure. GNU Modula-2 will only allow unbounded parameters to be passed by reference if, inside the callee procedure, they are not written to, no address is calculated on the array and it is not passed as a VAR parameter. Note that it is possible to write code to break this optimization, therefore this option should be used carefully. For example it would be possible to take the address of an array, pass the address and the array to a procedure, read from the array in the procedure and write to the location using the address parameter.

Due to the dangerous nature of this option it is not enabled when the -O option is specified.

-Wverbose-unbounded
inform the user which non VAR unbounded parameters will be passed by reference. This only produces output if the option -funbounded-by-reference is also supplied on the command line.

-Wstyle
checks for poor programming style. This option is aimed at new users of Modula-2 in that it checks for situations which might cause confusion and thus mistakes. It checks whether variables of the same name are declared in different scopes and whether variables look like keywords. Experienced users might find this option too aggressive.

-Wpedantic
forces the compiler to reject nested WITH statements referencing the same record type. Does not allow multiple imports of the same item from a module. It also checks that: procedure variables are written to before being read; variables are not only written to but read from; variables are declared and used. If the compiler encounters a variable being read before written it will terminate with a message. It will check that FOR loop indices are not used outside the end of this loop without being reset.

-Wpedantic-param-names
procedure parameter names are checked in the definition module against their implementation module counterpart. This is not necessary in ISO or PIM versions of Modula-2.

-Wpedantic-cast
warns if the ISO system function is used and if the size of the variable is different from that of the type. This is legal in ISO Modula-2, however it can be dangerous. Some users may prefer to use VAL instead in these situations and use CAST exclusively for changes in type on objects which have the same size.

-Wunused-variable
warns if a variable has been declared and it not used.

-Wunused-parameter
warns if a parameter has been declared and it not used.

-Wall
turn on all Modula-2 warnings.

15.3 GNU Modula-2 related environment variables

This section descibes the environment variables used by GNU Modula-2 and how they can be used to switch between releases of the compiler. Other environment variables can be set to modify the default library path. Initially we will consider environment variables most likely used by the end user. These two environment variables are `GM2IPATH` and `GM2OPATH`.

For example suppose a compile and link on the command line looks like this:

```
$ gm2 -g -c -I. -I../project -I../project/unix foo.mod
$ gm2 -fonlylink -g -I. -I../project -I../project/unix \
  -fobject-path=../project/obj -Iobject-path=../project/unix/obj \
  -I. foo.mod
```

they can be simplified by utilising two environment variables to do exactly the same compile and link.

```
$ export GM2IPATH=../project:../project/unix
$ export GM2OPATH=../project/obj:../project/unix/obj
$ gm2 -g -I. foo.mod
```

It is important to note that the two environment variables `GM2IPATH` and `GM2OPATH` have a lower priority than any `-I` or `-fobject-path=` command line option. The search order for compiling and linking is: command line switches followed by environment variable paths followed by default runtime libraries or Modula-2 dialect libraries. If in doubt include the `-v` option to see the search path used between the compiler subcomponents.

15.4 Elementary data types

This section describes the elementary data types supported by GNU Modula-2. It also describes the relationship between these data types and the equivalent C data types.

The following data types are supported: `INTEGER`, `LONGINT`, `SHORTINT`, `CARDINAL`, `LONGCARD`, `SHORTCARD`, `BOOLEAN`, `REAL`, `LONGREAL`, `SHORTREAL`, `COMPLEX`, `LONGCOMPLEX`, `SHORTCOMPLEX` and `CHAR`.

An equivalence table is given below:

```
GNU Modula-2              GNU C
====================================
INTEGER                   int
LONGINT                   long long int
SHORTINT                  short int
CARDINAL                  unsigned int
LONGCARD                  long long unsigned int
SHORTCARD                 short unsigned int
BOOLEAN                   int
REAL                      double
LONGREAL                  long double
SHORTREAL                 float
```

(continues on next page)

(continued from previous page)

```
CHAR                    char
SHORTCOMPLEX            complex float
COMPLEX                 complex double
LONGCOMPLEX             complex long double
```

Note that GNU Modula-2 also supports fixed sized data types which are exported from the SYSTEM module. See The PIM system module. See The ISO system module.

15.5 Permanently accessible base procedures.

This section describes the procedures and functions which are always visible.

15.5.1 Standard procedures and functions common to PIM and ISO

The following procedures are implemented and conform with Programming in Modula-2 and ISO Modula-2: NEW, DISPOSE, INC, DEC, INCL, EXCL and HALT. The standard functions are: ABS, CAP, CHR, FLOAT, HIGH, LFLOAT, LTRUNC, MIN, MAX, ODD, SFLOAT, STRUNC TRUNC and VAL. All these functions and procedures (except HALT, NEW, DISPOSE and, under non constant conditions, LENGTH) generate in-line code for efficiency.

```
(*
   ABS - returns the positive value of i.
*)

ABS
PROCEDURE ABS (i: <any signed type>) : <any signed type> ;
```

```
(*
   CAP - returns the capital of character ch providing
         ch lies within the range 'a'..'z'. Otherwise ch
         is returned unaltered.
*)

CAP
PROCEDURE CAP (ch: CHAR) : CHAR ;
```

```
(*
   CHR - converts a value of a <whole number type> into a CHAR.
         CHR(x) is shorthand for VAL(CHAR, x).
*)

CHR
PROCEDURE CHR (x: <whole number type>) : CHAR ;
```

```
(*
   DISPOSE - the procedure DISPOSE is replaced by:
```

(continues on next page)

(continued from previous page)

```
            DEALLOCATE(p, TSIZE(p^)) ;
            The user is expected to import the procedure DEALLOCATE
            (normally found in the module, Storage.)

            In:  a variable p: of any pointer type which has been
                 initialized by a call to NEW.
            Out: the area of memory
                 holding p^ is returned to the system.
                 Note that the underlying procedure DEALLOCATE
                 procedure in module Storage will assign p to NIL.
*)

DISPOSE
PROCEDURE DISPOSE (VAR p:<any pointer type>) ;
```

```
(*
   DEC - can either take one or two parameters.  If supplied
         with one parameter then on the completion of the call to
         DEC, v will have its predecessor value.  If two
         parameters are supplied then the value v will have its
         n'th predecessor.  For these reasons the value of n
         must be >=0.
*)

DEC
PROCEDURE DEC (VAR v: <any base type>; [n: <any base type> = 1]) ;
```

```
(*
   EXCL - excludes bit element e from a set type s.
*)

EXCL
PROCEDURE EXCL (VAR s: <any set type>; e: <element of set type s>) ;
```

```
(*
   FLOAT - will return a REAL number whose value is the same as o.
*)

FLOAT
PROCEDURE FLOAT (o: <any whole number type>) : REAL ;
```

```
(*
   FLOATS - will return a SHORTREAL number whose value is the same as o.
*)

FLOATS
PROCEDURE FLOATS (o: <any whole number type>) : REAL ;
```

```
(*
   FLOATL - will return a LONGREAL number whose value is the same as o.
```

(continues on next page)

(continued from previous page)

```
*)

FLOATL
PROCEDURE FLOATL (o: <any whole number type>) : REAL ;
```

```
(*
   HALT - will call the HALT procedure inside the module M2RTS.
          Users can replace M2RTS.
*)

HALT
PROCEDURE HALT ;
```

```
(*
   HIGH - returns the last accessible index of an parameter declared as
          ARRAY OF CHAR. Thus

          PROCEDURE foo (a: ARRAY OF CHAR) ;
          VAR
             c: CARDINAL ;
          BEGIN
             c := HIGH(a)
          END foo ;

          BEGIN
             foo('hello')
          END

          will cause the local variable c to contain the value 4
*)

HIGH
PROCEDURE HIGH (a: ARRAY OF CHAR) : CARDINAL ;
```

```
(*
   INC - can either take one or two parameters.  If supplied
         with one parameter then on the completion of the call to
         INC, v will have its successor value.  If two
         parameters are supplied then the value v will have its
         n'th successor.  For these reasons the value of n
         must be >=0.
*)

INC
PROCEDURE INC (VAR v: <any base type>; [n: <any base type> = 1]) ;
```

```
(*
   INCL - includes bit element e to a set type s.
*)
```

(continues on next page)

(continued from previous page)

```
INCL
PROCEDURE INCL (VAR s: <any set type>; e: <element of set type s>) ;
```

```
(*
   LFLOAT - will return a LONGREAL number whose value is the same as o.
*)

LFLOAT
PROCEDURE LFLOAT (o: <any whole number type>) : LONGREAL ;
```

```
(*
   LTRUNC - will return a LONG<type> number whose value is the
            same as o.  PIM2, PIM3 and ISO Modula-2 will return
            a LONGCARD whereas PIM4 returns LONGINT.
*)

LTRUNC
PROCEDURE LTRUNC (o: <any floating point type>) : LONG<type> ;
```

```
(*
   MIN - returns the lowest legal value of an ordinal type.
*)

MIN
PROCEDURE MIN (t: <ordinal type>) : <ordinal type> ;
```

```
(*
   MAX - returns the largest legal value of an ordinal type.
*)

MAX
PROCEDURE MAX (t: <ordinal type>) : <ordinal type> ;
```

```
(*
   NEW - the procedure NEW is replaced by:
         ALLOCATE(p, TSIZE(p^)) ;
         The user is expected to import the procedure ALLOCATE
         (normally found in the module, Storage.)

         In:  a variable p: of any pointer type.
         Out: variable p is set to some allocated memory
              which is large enough to hold all the contents of p^.
*)

NEW
PROCEDURE NEW (VAR p:<any pointer type>) ;
```

```
(*
   ODD - returns TRUE if the value is not divisible by 2.
```

(continues on next page)

(continued from previous page)

```
*)

ODD
PROCEDURE ODD (x: <whole number type>) : BOOLEAN ;
```

```
(*
   SFLOAT - will return a SHORTREAL number whose value is the same
            as o.
*)

SFLOAT
PROCEDURE SFLOAT (o: <any whole number type>) : SHORTREAL ;
```

```
(*
   STRUNC - will return a SHORT<type> number whose value is the same
            as o.  PIM2, PIM3 and ISO Modula-2 will return a
            SHORTCARD whereas PIM4 returns SHORTINT.
*)

STRUNC
PROCEDURE STRUNC (o: <any floating point type>) : SHORT<type> ;
```

```
(*
   TRUNC - will return a <type> number whose value is the same as o.
           PIM2, PIM3 and ISO Modula-2 will return a CARDINAL
           whereas PIM4 returns INTEGER.
*)

TRUNC
PROCEDURE TRUNC (o: <any floating point type>) : <type> ;
```

```
(*
   TRUNCS - will return a <type> number whose value is the same
            as o.  PIM2, PIM3 and ISO Modula-2 will return a
            SHORTCARD whereas PIM4 returns SHORTINT.
*)

TRUNCS
PROCEDURE TRUNCS (o: <any floating point type>) : <type> ;
```

```
(*
   TRUNCL - will return a <type> number whose value is the same
            as o.  PIM2, PIM3 and ISO Modula-2 will return a
            LONGCARD whereas PIM4 returns LONGINT.
*)

TRUNCL
PROCEDURE TRUNCL (o: <any floating point type>) : <type> ;
```

```
(*
   VAL - converts data i of <any simple data type 2> to
         <any simple data type 1> and returns this value.
         No range checking is performed during this conversion.
*)

VAL
PROCEDURE VAL (<any simple data type 1>,
               i: <any simple data type 2>) : <any simple data type 1> ;
```

15.5.2 ISO specific standard procedures and functions

The standard function `LENGTH` is specific to ISO Modula-2 and is defined as:

```
(*
   IM - returns the imaginary component of a complex type.
        The return value will the same type as the imaginary field
        within the complex type.
*)

IM
PROCEDURE IM (c: <any complex type>) : <floating point type> ;
```

```
(*
   INT - returns an INTEGER value which has the same value as v.
         This function is equivalent to: VAL(INTEGER, v).
*)

INT
PROCEDURE INT (v: <any ordinal type>) : INTEGER ;
```

```
(*
   LENGTH - returns the length of string a.
*)

LENGTH
PROCEDURE LENGTH (a: ARRAY OF CHAR) : CARDINAL ;
```

This function is evaluated at compile time, providing that string `a` is a constant. If `a` cannot be evaluated then a call is made to `M2RTS.Length`.

```
(*
   ODD - returns a BOOLEAN indicating whether the whole number
         value, v, is odd.
*)

ODD
PROCEDURE ODD (v: <any whole number type>) : BOOLEAN ;
```

```
(*
   RE - returns the real component of a complex type.
        The return value will the same type as the real field
        within the complex type.
*)

RE
PROCEDURE RE (c: <any complex type>) : <floating point type> ;
```

15.6 GNU Modula-2 supported dialects

This section describes the dialects understood by GNU Modula-2. It also describes the differences between the dialects and any command line switches which determine dialect behaviour.

The GNU Modula-2 compiler is compliant with four dialects of Modula-2. The language as defined in 'Programming in Modula-2' 2nd Edition, Springer Verlag, 1982, 1983 by Niklaus Wirth (PIM2), 'Programming in Modula-2', 3rd Corrected Edition, Springer Verlag, 1985 (PIM3) and 'Programming in Modula-2', 4th Edition, Springer Verlag, 1988 (PIM4) http://freepages.modula2.org/report4/modula-2.html and the ISO Modula-2 language as defined in ISO/IEC Information technology - programming languages - part 1: Modula-2 Language, ISO/IEC 10514-1 (1996) (ISO).

The command line switches `-fpim2`, `-fpim3`, `-fpim4` and `-fiso` can be used to force mutually exclusive features. However by default the compiler will not aggressively fail if a non mutually exclusive feature is used from another dialect. For example it is possible to specify `-fpim2` and still utilise `DEFINITION MODULES` which have no export list.

Some dialect differences will force a compile time error, for example in PIM2 the user must `IMPORT SIZE` from the module `SYSTEM`, whereas in PIM3 and PIM4 `SIZE` is a pervasive function. Thus compiling PIM4 source code with the `-fpim2` switch will cause a compile time error. This can be fixed quickly with an additional `IMPORT` or alternatively by compiling with the `-fpim4` switch.

However there are some very important differences between the dialects which are mutually exclusive and therefore it is vital that users choose the dialects with care when these language features are used.

15.6.1 Integer division, remainder and modulus

The most dangerous set of mutually exclusive features found in the four dialects supported by GNU Modula-2 are the `INTEGER` division, remainder and modulus arithmetic operators. It is important to note that the same source code can be compiled to give different runtime results depending upon these switches! The reference manual for the various dialects of Modula-2 are quite clear about this behaviour and sadly there are three distinct definitions.

The table below illustrates the problem when a negative operand is used.

```
                 Pim2/3          Pim4               ISO
               -----------    -----------    ----------------------
lval    rval   DIV     MOD    DIV     MOD    DIV    MOD    /    REM
```

(continues on next page)

(continued from previous page)

```
 31      10      3       1      3       1      3      1     3     1
-31      10     -3      -1     -4       9     -4      9    -3    -1
 31     -10     -3       1     -3       1     Exception    -3     1
-31     -10      3      -1      4       9     Exception     3    -1
```

See also P24 of PIM2, P27 of PIM3, P29 of PIM4 and P201 of the ISO Standard. At present all dialect division, remainder and modulus are implemented as above, apart from the exception calling in the ISO dialect. Instead of exception handling the results are the same as the PIM4 dialect. This is a temporary implementation situation.

15.7 Exception implementation

This section describes how exceptions are implemented in GNU Modula-2 and how command line switches affect their behaviour. The option `-fsoft-check-all` enables all software checking of nil dereferences, division by zero etc. Additional code is produced to check these conditions and exception handlers are invoked if the conditions prevail.

Without `-fsoft-check-all` these exceptions will be caught by hardware (assuming the hardware support exists) and a signal handler is invoked. The signal handler will in turn `THROW` an exception which will be caught by the appropriate Modula-2 handler. However the action of throwing an exception from within a signal handler is implementation defined (according to the C++ documentation). For example on the x86_64 architecture this works whereas on the i686 architecture it does not. Therefore to ensure portability it is recommended to use `-fsoft-check-all`.

`-fsoft-check-all` can be effectively combined with `-O2` to semantically analyse source code for possible runtime errors at compile time.

15.8 How to detect runtime problems at compile time

Consider the following program:

```
MODULE assignvalue ;  (*!m2iso+gm2*)

PROCEDURE bad () : INTEGER ;
VAR
   i: INTEGER ;
BEGIN
   i := -1 ;
   RETURN i
END bad ;

VAR
   foo: CARDINAL ;
BEGIN
   (* the m2rte plugin will detect this as an error, post
      optimization.  *)
```

(continues on next page)

(continued from previous page)

```
   foo := bad ()
END assignvalue.
```

here we see that the programmer has overlooked that the return value from `bad` will cause an overflow to `foo`. If we compile the code with the following options:

```
$ gm2 -g -fsoft-check-all -O2 -c assignvalue.mod
assignvalue.mod:16:0:inevitable that this error will occur at runtime,
assignment will result in an overflow
```

The gm2 semantic plugin is automatically run and will generate a warning message for every exception call which is known as reachable. It is highly advised to run the optimizer (`-O2` or `-O3`) with `-fsoft-check-all` so that the compiler is able to run the optimizer and perform variable and flow analysis before the semantic plugin is invoked.

15.9 GNU Modula-2 language extensions

This section introduces the GNU Modula-2 language extensions. The GNU Modula-2 compiler allows abstract data types to be any type, not just restricted to a pointer type providing the `-fextended-opaque` option is supplied See Compiler options.

Declarations can be made in any order, whether they are types, constants, procedures, nested modules or variables.

GNU Modula-2 also allows programmers to interface to `C` and assembly language.

GNU Modula-2 provides support for the special tokens `__LINE__`, `__FILE__`, `__FUNCTION__` and `__DATE__`. Support for these tokens will occur even if the `-fcpp` option is not supplied. A table of these identifiers and their data type and values is given below:

```
Scope       GNU Modula-2 token      Data type and example value

anywhere    __LINE__                Constant Literal compatible
                                    with CARDINAL, INTEGER and WORD.
                                    Example 1234

anywhere    __FILE__                Constant string compatible
                                    with parameter ARRAY OF CHAR or
                                    an ARRAY whose SIZE is >= string
                                    length. Example
                                    "hello.mod"

procedure   __FUNCTION__            Constant string compatible
                                    with parameter ARRAY OF CHAR or
                                    an ARRAY whose SIZE is >= string
                                    length. Example
                                    "calc"

module      __FUNCTION__            Example
```

(continues on next page)

(continued from previous page)

```
                                "module hello initialization"

anywhere    __DATE__            Constant string compatible
                                with parameter ARRAY OF CHAR or
                                an ARRAY whose SIZE is >= string
                                length. Example
                                "Thu Apr 29 10:07:16 BST 2004"

anywhere    __COLUMN__          Gives a contant literal number
                                determining the left hand column
                                where the first _ appears in
                                __COLUMN__.  The left most column
                                is 1.
```

The preprocessor `cpp` can be invoked via the `-fcpp` command line option. This in turn invokes `cpp` with the following arguments `-traditional -lang-asm`. These options preserve comments and all quotations. `gm2` treats a `#` character in the first column as a preprocessor directive.

For example here is a module which calls `FatalError` via the macro `ERROR`.

```
MODULE cpp ;

FROM SYSTEM IMPORT ADR, SIZE ;
FROM libc IMPORT exit, printf, malloc ;

PROCEDURE FatalError (a, file: ARRAY OF CHAR;
                      line: CARDINAL;
                      func: ARRAY OF CHAR) ;
BEGIN
   printf("%s:%d:fatal error, %s, in %s\n",
          ADR(file), line, ADR(a), ADR(func)) ;
   exit(1)
END FatalError ;

#define ERROR(X)  FatalError(X, __FILE__, __LINE__, __FUNCTION__)

VAR
   pc: POINTER TO CARDINAL;
BEGIN
   pc := malloc(SIZE(CARDINAL)) ;
   IF pc=NIL
   THEN
      ERROR('out of memory')
   END
END cpp.
```

Another use for the C preprocessor in Modula-2 might be to turn on debugging code. For example the library module `FormatStrings.mod` uses procedures from `DynamicStrings.mod` and to track down memory leaks it was useful to track the source file and line where each string was created. Here is a section of `FormatStrings.mod` which shows how the debugging code was enabled and disabled by adding `-fcpp` to the command line.

```
FROM DynamicStrings IMPORT String, InitString, InitStringChar, Mark,
                           ConCat, Slice, Index, char,
                           Assign, Length, Mult, Dup, ConCatChar,
                           PushAllocation, PopAllocationExemption,
                           InitStringDB, InitStringCharStarDB,
                           InitStringCharDB, MultDB, DupDB, SliceDB ;

(*
#define InitString(X) InitStringDB(X, __FILE__, __LINE__)
#define InitStringCharStar(X) InitStringCharStarDB(X, __FILE__, \
                                                   __LINE__)
#define InitStringChar(X) InitStringCharDB(X, __FILE__, __LINE__)
#define Mult(X,Y) MultDB(X, Y, __FILE__, __LINE__)
#define Dup(X) DupDB(X, __FILE__, __LINE__)
#define Slice(X,Y,Z) SliceDB(X, Y, Z, __FILE__, __LINE__)
*)

PROCEDURE doDSdbEnter ;
BEGIN
   PushAllocation
END doDSdbEnter ;

PROCEDURE doDSdbExit (s: String) ;
BEGIN
   s := PopAllocationExemption(TRUE, s)
END doDSdbExit ;

PROCEDURE DSdbEnter ;
BEGIN
END DSdbEnter ;

PROCEDURE DSdbExit (s: String) ;
BEGIN
END DSdbExit ;

(*
#define DBsbEnter doDBsbEnter
#define DBsbExit  doDBsbExit
*)

PROCEDURE Sprintf1 (s: String; w: ARRAY OF BYTE) : String ;
BEGIN
   DSdbEnter ;
   s := FormatString(HandleEscape(s), w) ;
   DSdbExit(s) ;
   RETURN( s )
END Sprintf1 ;
```

It is worth noting that the overhead of this code once `-fcpp` is not present and -O2 is used will be zero since the local empty procedures `DSdbEnter` and `DSdbExit` will be thrown away by the optimization passes of the GCC backend.

15.9.1 Optional procedure parameter

GNU Modula-2 allows the last parameter to a procedure or function parameter to be optional. For example in the ISO library `COROUTINES.def` the procedure `NEWCOROUTINE` is defined as having an optional fifth argument (`initProtection`) which, if absent, is automatically replaced by `NIL`.

```
NEWCOROUTINE
PROCEDURE NEWCOROUTINE (procBody: PROC; workspace: SYSTEM.ADDRESS;
                        size: CARDINAL; VAR cr: COROUTINE;
                        [initProtection: PROTECTION = NIL]);

  (* Creates a new coroutine whose body is given by procBody,
     and returns the identity of the coroutine in cr.
     workspace is a pointer to the work space allocated to
     the coroutine; size specifies the size of this workspace
     in terms of SYSTEM.LOC.

     The optional fifth argument may contain a single parameter
     which specifies the initial protection level of the coroutine.
  *)
```

The implementation module `COROUTINES.mod` implements this procedure using the following syntax:

```
PROCEDURE NEWCOROUTINE (procBody: PROC; workspace: SYSTEM.ADDRESS;
                        size: CARDINAL; VAR cr: COROUTINE;
                        [initProtection: PROTECTION]);
BEGIN

END NEWCOROUTINE ;
```

Note that it is illegal for this declaration to contain an initialiser value for `initProtection`. However it is necessary to surround this parameter with the brackets `[` and `]`. This serves to remind the programmer that the last parameter was declared as optional in the definition module.

Local procedures can be declared to have an optional final parameter in which case the initializer is mandatory in the implementation or program module.

GNU Modula-2 also provides additional fixed sized data types which are all exported from the `SYSTEM` module. See The PIM system module. See The ISO system module.

15.10 Type compatibility

This section discuss the issues surrounding assignment, expression and parameter compatibility, their effect of the additional fixed sized datatypes and also their effect of runtime checking. The data types supported by the compiler are:

```
GNU Modula-2              scope      switches
=========================================
INTEGER                   pervasive
LONGINT                   pervasive
```

(continues on next page)

(continued from previous page)

```
SHORTINT                pervasive
CARDINAL                pervasive
LONGCARD                pervasive
SHORTCARD               pervasive
BOOLEAN                 pervasive
BITSET                  pervasive
REAL                    pervasive
LONGREAL                pervasive
SHORTREAL               pervasive
CHAR                    pervasive
SHORTCOMPLEX            pervasive
COMPLEX                 pervasive
LONGCOMPLEX             pervasive

LOC                     SYSTEM     -fiso
BYTE                    SYSTEM
WORD                    SYSTEM
ADDRESS                 SYSTEM

The following extensions are supported for
most architectures (please check SYSTEM.def).
=============================================
INTEGER8                SYSTEM
INTEGER16               SYSTEM
INTEGER32               SYSTEM
INTEGER64               SYSTEM
CARDINAL8               SYSTEM
CARDINAL16              SYSTEM
CARDINAL32              SYSTEM
CARDINAL64              SYSTEM
BITSET8                 SYSTEM
BITSET16                SYSTEM
BITSET32                SYSTEM
WORD16                  SYSTEM
WORD32                  SYSTEM
WORD64                  SYSTEM
REAL32                  SYSTEM
REAL64                  SYSTEM
REAL96                  SYSTEM
REAL128                 SYSTEM
COMPLEX32               SYSTEM
COMPLEX64               SYSTEM
COMPLEX96               SYSTEM
COMPLEX128              SYSTEM
```

The Modula-2 language categorises compatibility between entities of possibly differing types into three subcomponents: expressions, assignments, and parameters. Parameter compatibility is further divided into two sections for pass by reference and pass by value compatibility.

For more detail on the Modula-2 type compatibility see the Modula-2 ISO standard BS ISO/IEC 10514-1:1996 page 121-125. For detail on the PIM type compatibility see Programming in Modula-2 Edition 4 page 29, (Elementary Data Types).

15.10.1 Expression compatibility

Modula-2 restricts the types of expressions to the same type. Expression compatibility is a symmetric relation.

For example two sub expressions of `INTEGER` and `CARDINAL` are not expression compatible (http://freepages.modula2.org/report4/modula-2.html and ISO Modula-2).

In GNU Modula-2 this rule is also extended across all fixed sized data types (imported from SYSTEM).

15.10.2 Assignment compatibility

This section discusses the assignment issues surrounding assignment compatibility of elementary types (`INTEGER`, `CARDINAL`, `REAL` and `CHAR` for example). The information here is found in more detail in the Modula-2 ISO standard BS ISO/IEC 10514-1:1996 page 122.

Assignment compatibility exists between the same sized elementary types.

Same type family of different sizes are also compatible as long as the `MAX(` type `)` and `MIN(` type `)` is known. So for example this includes the `INTEGER` family, `CARDINAL` family and the `REAL` family.

The reason for this is that when the assignment is performed the compiler will check to see that the expression (on the right of the `:=`) lies within the range of the designator type (on the left hand side of the `:=`). Thus these ordinal types can be assignment compatible. However it does mean that `WORD32` is not compatible with `WORD16` as `WORD32` does not have a minimum or maximum value and therefore cannot be checked. The compiler does not know which of the two bytes from `WORD32` should be copied into `WORD16` and which two should be ignored. Currently the types `BITSET8`, `BITSET16` and `BITSET32` are assignment incompatible. However this restriction maybe lifted when further runtime checking is achieved.

Modula-2 does allow `INTEGER` to be assignment compatible with `WORD` as they are the same size. Likewise GNU Modula-2 allows `INTEGER16` to be compatible with `WORD16` and the same for the other fixed sized types and their sized equivalent in either `WORD` n, `BYTE` or `LOC` types. However it prohibits assignment between `WORD` and `WORD32` even though on many systems these sizes will be the same. The reasoning behind this rule is that the extended fixed sized types are meant to be used by applications requiring fixed sized data types and it is more portable to forbid the bluring of the boundaries between fixed sized and machine dependant sized types.

Intemediate code runtime checking is always generated by the front end. However this intemediate code is only translated into actual code if the appropriate command line switches are specified. This allows the compiler to perform limited range checking at compile time. In the future it will allow the extensive GCC optimisations to propagate constant values through to the range checks which if they are found to exceed the type range will result in a compile time error message.

15.10.3 Parameter compatibility

Parameter compatibility is divided into two areas, pass by value and pass by reference (`VAR`). In the case of pass by value the rules are exactly the same as assignment. However in the second case, pass by reference, the actual parameter and formal parameter must be the same size and family. Furthermore `INTEGER` and `CARDINAL` s are not treated as compatible in the pass by reference case.

The types `BYTE`, `LOC`, `WORD` and `WORD` n derivitives are assignment and parameter compatible with any data type of the same size.

15.11 Unbounded by reference

This section documents a GNU Modula-2 compiler switch which implements a language optimisation surrounding the implementation of unbounded arrays. In GNU Modula-2 the unbounded array is implemented by utilising an internal structure `struct {dataType *address, unsigned int high}`. So given the Modula-2 procedure declaration:

```
PROCEDURE foo (VAR a: ARRAY OF dataType) ;
BEGIN
   IF a[2]= (* etc *)
END foo ;
```

it is translated into GCC `tree` s, which can be represented in their C form thus:

```
void foo (struct {dataType *address, unsigned int high} a)
{
   if (a.address[2] == /* etc */
}
```

Whereas if the procedure `foo` was declared as:

```
PROCEDURE foo (a: ARRAY OF dataType) ;
BEGIN
   IF a[2]= (* etc *)
END foo ;
```

then it is implemented by being translated into the following GCC `tree` s, which can be represented in their C form thus:

```
void foo (struct {dataType *address, unsigned int high} a)
{
   dataType *copyContents = (dataType *)alloca (a.high+1);
   memcpy(copyContents, a.address, a.high+1);
   a.address = copyContents;

   if (a.address[2] == /* etc */
}
```

This implementation works, but it makes a copy of each non VAR unbounded array when a procedure is entered. If the unbounded array is not changed during procedure `foo` then this imple-

mentation will be very inefficient. In effect Modula-2 lacks the `REF` keyword of Ada. Consequently the programmer maybe tempted to sacrifice semantic clarity for greater efficiency by declaring the parameter using the `VAR` keyword in place of `REF`.

The `-funbounded-by-reference` switch instructs the compiler to check and see if the programmer is modifying the content of any unbounded array. If it is modified then a copy will be made upon entry into the procedure. Conversely if the content is only read and never modified then this non `VAR` unbounded array is a candidate for being passed by reference. It is only a candidate as it is still possible that passing this parameter by reference could alter the meaning of the source code. For example consider the following case:

```
PROCEDURE StrConCat (VAR a: ARRAY OF CHAR; b, c: ARRAY OF CHAR) ;
BEGIN
   (* code which performs string a := b + c *)
END StrConCat ;

PROCEDURE foo ;
VAR
   a: ARRAY [0..3] OF CHAR ;
BEGIN
   a := 'q' ;
   StrConCat(a, a, a)
END foo ;
```

In the code above we see that the same parameter, `a`, is being passed three times to `StrConCat`. Clearly even though parameters `b` and `c` are never modified it would be incorrect to implement them as pass by reference. Therefore the compiler checks to see if any non `VAR` parameter is type compatible with any `VAR` parameter and if so it generates runtime procedure entry checks to determine whether the contents of parameters `b` or `c` matches the contents of `a`. If a match is detected then a copy is made and the `address` in the unbounded `struct` ure is modified.

The compiler will check the address range of each candidate against the address range of any `VAR` parameter, providing they are type compatible. For example consider:

```
PROCEDURE foo (a: ARRAY OF BYTE; VAR f: REAL) ;
BEGIN
   f := 3.14 ;
   IF a[0]=BYTE(0)
   THEN
      (* etc *)
   END
END foo ;

PROCEDURE bar ;
BEGIN
   r := 2.0 ;
   foo(r, r)
END bar ;
```

Here we see that although parameter, `a`, is a candidate for the passing by reference, it would be incorrect to use this transformation. Thus the compiler detects that parameters, `a` and `f` are type compatible and will produce runtime checking code to test whether the address range of their

respective contents intersect.

This section describes the linking related options. There are three linking strategies available which are dynamic scaffold, static scaffold and user defined. The dynamic scaffold is enabled by default and each module will register itself to the runtime `M2RTS` via a constructor. The static scaffold mechanism will invoke each modules `_init` and `_finish` function in turn via a sequence of calls from within `main`. Lastly the user defined strategy can be implemented by turning off the dynamic and static options via `-fno-scaffold-dynamic` and `-fno-scaffold-static`.

In the simple test below:

```
$ gm2 hello.mod
```

the driver will add the options `-fscaffold-dynamic` and `-fgen-module-list=-` which generate a list of application modules and also creates the `main` function with calls to `M2RTS`. It can be useful to add the option `-fsources` which displays the source files as they are parsed and summarizes whether the source file is required for compilation or linking.

If you wish to split the above command line into a compile and link then you could use these steps:

```
$ gm2 -c -fscaffold-main hello.mod
$ gm2 hello.o
```

The `-fscaffold-main` informs the compiler to generate the `main` function and scaffold. You can enable the environment variable `GCC_M2LINK_RTFLAG` to trace the construction and destruction of the application. The values for `GCC_M2LINK_RTFLAG` are shown in the table below:

```
value   | meaning
================
all     | turn on all flags below
module  | trace modules as they register themselves
pre     | generate module list prior to dependency resolution
dep     | trace module dependency resolution
post    | generate module list after dependency resolution
force   | generate a module list after dependency and forced
        | ordering is complete
```

The values can be combined using a comma separated list.

One of the advantages of the dynamic scaffold is that the driver behaves in a similar way to the other front end drivers. For example consider a small project consisting of 4 definition implementation modules (`a.def`, `a.mod`, `b.def`, `b.mod`, `c.def`, `c.mod`, `d.def`, `d.mod`) and a program module `program.mod`.

To link this project we could:

```
$ gm2 -g -c a.mod
$ gm2 -g -c b.mod
$ gm2 -g -c c.mod
$ gm2 -g -c d.mod
$ gm2 -g program.mod a.o b.o c.o d.o
```

The module initialization sequence is defined by the ISO standard to follow the import graph

traversal. The initialization order is the order in which the corresponding separate modules finish the processing of their import lists.

However, if required, you can override this using `-fruntime-modules=a,b,c,d` for example which forces the initialization sequence to `a`, `b`, `c` and `d`.

15.12 Building a shared library

This section describes building a tiny shared library implemented in Modula-2 and built with `libtool`. Suppose a project consists of two definition modules and two implementation modules and a program module `a.def`, `a.mod`, `b.def`, `b.mod` and `c.mod`. The first step is to compile the modules using position independent code. This can be achieved by the following three commands:

```
libtool --tag=CC --mode=compile gm2 -g -c a.mod -o a.lo
libtool --tag=CC --mode=compile gm2 -g -c b.mod -o b.lo
libtool --tag=CC --mode=compile gm2 -g -c c.mod -o c.lo
```

The second step is to generate the shared library initialization and finalization routines. We can do this by asking gm2 to generate a list of dependant modules and then use this to generate the scaffold. We also must compile the scaffold.

```
gm2 -c -g -fmakelist c.mod
gm2 -c -g -fmakeinit -fshared c.mod
libtool --tag=CC --mode=compile g++ -g -c c_m2.cpp -o c_m2.lo
```

The third step is to link all these `.lo` files.

```
libtool --mode=link gcc -g c_m2.lo a.lo b.lo c.lo \
        -L$(prefix)/lib64 \
        -rpath `pwd` -lgm2 -lstdc++ -lm -o libabc.la
```

At this point the shared library `libabc.so` will have been created inside the directory `.libs`.

15.13 How to produce swig interface files

This section describes how Modula-2 implementation modules can be called from Python (and other scripting languages such as TCL and Perl). GNU Modula-2 can be instructed to create a swig interface when it is compiling an implementation module. Swig then uses the interface file to generate all the necessary wrapping to that the desired scripting language may access the implementation module.

Here is an example of how you might call upon the services of the Modula-2 library module `NumberIO` from Python3.

The following commands can be used to generate the Python3 module:

```
export src=directory to the sources
export prefix=directory to where the compiler is installed
gm2 -I${src} -c -g -fswig ${src}/../../../gm2-libs/NumberIO.mod
gm2 -I${src} -c -g -fmakelist ${src}/../../../gm2-libs/NumberIO.mod

gm2 -I${src} -c -g -fmakeinit -fshared \
   ${src}/../../../gm2-libs/NumberIO.mod

swig -c++ -python3 NumberIO.i

libtool --mode=compile g++ -g -c -I${src} NumberIO_m2.cpp \
  -o NumberIO_m2.lo

libtool --tag=CC --mode=compile gm2 -g -c \
  -I${src}../../../gm2-libs \
  ${src}/../../../gm2-libs/NumberIO.mod -o NumberIO.lo

libtool --tag=CC --mode=compile g++ -g -c NumberIO_wrap.cxx \
  -I/usr/include/python3 -o NumberIO_wrap.lo

libtool --mode=link gcc -g NumberIO_m2.lo NumberIO_wrap.lo \
   -L${prefix}/lib64 \
   -rpath `pwd` -lgm2 -lstdc++ -lm -o libNumberIO.la

cp .libs/libNumberIO.so _NumberIO.so
```

The first four commands, generate the swig interface file `NumberIO.i` and python wrap files `NumberIO_wrap.cxx` and `NumberIO.py`. The next three `libtool` commnads compile the C++ and Modula-2 source code into `.lo` objects. The last `libtool` command links all the `.lo` files into a `.la` file and includes all shared library dependencies.

Now it is possible to run the following Python script (called `testnum.py`):

```
import NumberIO

print ("1234 x 2 =", NumberIO.NumberIO_StrToInt("1234")*2)
```

like this:

```
$ python3 testnum.py
1234 x 2 = 2468
```

See How to produce a Python module for another example which uses the `UNQUALIFIED` keyword to reduce the module name clutter from the viewport of Python3.

15.13.1 Limitations of automatic generated of Swig files

This section discusses the limitations of automatically generating swig files. From the previous example we see that the module `NumberIO` had a swig interface file `NumberIO.i` automatically generated by the compiler. If we consider three of the procedure definitions in `NumberIO.def` we can see the success and limitations of the automatic interface generation.

```
PROCEDURE StrToHex (a: ARRAY OF CHAR; VAR x: CARDINAL) ;
PROCEDURE StrToInt (a: ARRAY OF CHAR; VAR x: INTEGER) ;
PROCEDURE ReadInt (VAR x: CARDINAL) ;
```

Below are the swig interface prototypes:

```
extern void NumberIO_StrToHex (char *_m2_address_a,
                               int _m2_high_a, unsigned int *OUTPUT);
/*  parameters: x is known to be an OUTPUT */
extern void NumberIO_StrToInt (char *_m2_address_a,
                               int _m2_high_a, int *OUTPUT);
/*  parameters: x is guessed to be an OUTPUT */
extern void NumberIO_ReadInt (int *x);
/*  parameters: x is unknown */
```

In the case of `StrToHex` it can be seen that the compiler detects that the last parameter is an output. It explicitly tells swig this by using the parameter name `OUTPUT` and in the following comment it informs the user that it knows this to be an output parameter. In the second procedure `StrToInt` it marks the final parameter as an output, but it tells the user that this is only a guess. Finally in `ReadInt` it informs the user that it does not know whether the parameter, `x`, is an output, input or an inout parameter.

The compiler decides whether to mark a parameter as either: `INPUT`, `OUTPUT` or `INOUT` if it is read before written or visa versa in the first basic block. At this point it will write output that the parameter is known. If it is not read or written in the first basic block then subsequent basic blocks are searched and the result is commented as a guess. Finally if no read or write occurs then the parameter is commented as unknown. However, clearly it is possible to fool this mechanism. Nevertheless automatic generation of implementation module into swig interface files was thought sufficiently useful despite these limitations.

In conclusion it would be wise to check all parameters in any automatically generated swig interface file. Furthermore you can force the automatic mechanism to generate correct interface files by reading or writing to the `VAR` parameter in the first basic block of a procedure.

15.14 How to produce a Python module

This section descibes how it is possible to produce a Python module from Modula-2 code. There are a number of advantages to this approach, it ensures your code reaches a wider audience, maybe it is easier to initialize your application in Python.

The example application here is a pedagogical two dimensional gravity next event simulation. The Python module needs to have a clear API which should be placed in a single definition module.

Furthermore the API should only use fundamental pervasive data types and strings. Below the API is contained in the file `twoDsim.def`:

```
DEFINITION MODULE twoDsim ;

EXPORT UNQUALIFIED gravity, box, poly3, poly5, poly6, mass,
                   fix, circle, pivot, velocity, accel, fps,
                   replayRate, simulateFor ;
(*
   gravity - turn on gravity at: g m^2
*)

PROCEDURE gravity (g: REAL) ;

(*
   box - place a box in the world at (x0,y0),(x0+i,y0+j)
*)

PROCEDURE box (x0, y0, i, j: REAL) : CARDINAL ;

(*
   poly3 - place a triangle in the world at:
           (x0,y0),(x1,y1),(x2,y2)
*)

PROCEDURE poly3 (x0, y0, x1, y1, x2, y2: REAL) : CARDINAL ;

(*
   poly5 - place a pentagon in the world at:
           (x0,y0),(x1,y1),(x2,y2),(x3,y3),(x4,y4)
*)

PROCEDURE poly5 (x0, y0, x1, y1,
                 x2, y2, x3, y3, x4, y4: REAL) : CARDINAL ;

(*
   poly6 - place a hexagon in the world at:
           (x0,y0),(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5)
*)

PROCEDURE poly6 (x0, y0, x1, y1,
                 x2, y2, x3, y3,
                 x4, y4, x5, y5: REAL) : CARDINAL ;

(*
   mass - specify the mass of an object and return the, id.
*)

PROCEDURE mass (id: CARDINAL; m: REAL) : CARDINAL ;

(*
   fix - fix the object to the world.
*)
```

(continues on next page)

(continued from previous page)

```
PROCEDURE fix (id: CARDINAL) : CARDINAL ;

(*
   circle - adds a circle to the world.  Center
            defined by: x0, y0 radius, r.
*)

PROCEDURE circle (x0, y0, r: REAL) : CARDINAL ;

(*
   velocity - give an object, id, a velocity, vx, vy.
*)

PROCEDURE velocity (id: CARDINAL; vx, vy: REAL) : CARDINAL ;

(*
   accel - give an object, id, an acceleration, ax, ay.
*)

PROCEDURE accel (id: CARDINAL; ax, ay: REAL) : CARDINAL ;

(*
   fps - set frames per second.
*)

PROCEDURE fps (f: REAL) ;

(*
   replayRate - set frames per second during replay.
*)

PROCEDURE replayRate (f: REAL) ;

(*
   simulateFor - render for, t, seconds.
*)

PROCEDURE simulateFor (t: REAL) ;

END twoDsim.
```

The keyword `UNQUALIFIED` can be used to ensure that the compiler will provide externally accessible functions `gravity`, `box`, `poly3`, `poly5`, `poly6`, `mass`, `fix`, `circle`, `pivot`, `velocity`, `accel`, `fps`, `replayRate`, `simulateFor` rather than name mangled alternatives. Hence in our Python3 application we could write:

```
#!/usr/bin/env python3

from twoDsim import *
```

(continues on next page)

(continued from previous page)

```
b = box (0.0, 0.0, 1.0, 1.0)
b = fix (b)
c1 = circle (0.7, 0.7, 0.05)
c1 = mass (c1, 0.01)
c2 = circle (0.7, 0.1, 0.05)
c2 = mass (c2, 0.01)
c2 = fix (c2)
gravity (-9.81)
fps (24.0*4.0)
replayRate (24.0)
print ("creating frames")
try:
    simulateFor (1.0)
    print ("all done")
except:
    print ("exception raised")
```

which accesses the various functions defined and implemented by the module `twoDsim`. The Modula-2 source code is compiled via:

```
$ gm2 -g -fiso -c -fswig twoDsim.mod
$ gm2 -g -fiso -c -fmakelist twoDsim.mod
$ gm2 -g -fiso -c -fmakeinit twoDsim.mod
```

The first command both compiles the source file creating `twoDsim.o` and produces a swig interface file `swig.i`. We now use `swig` and `g++` to produce and compile the interface wrappers:

```
$ libtool --mode=compile g++ -g -c twoDsim_m2.cpp -o twoDsim_m2.lo
$ swig -c++ -python3 twoDsim.i
$ libtool --mode=compile g++ -c -fPIC twoDsim_wrap.cxx \
   -I/usr/include/python3 -o twoDsim_wrap.lo
$ libtool --mode=compile gm2 -g -fPIC -fiso -c deviceGnuPic.mod
$ libtool --mode=compile gm2 -g -fPIC -fiso -c roots.mod
$ libtool --mode=compile gm2 -g -fPIC -fiso -c -fswig \
   twoDsim.mod -o twoDsim.lo
```

Finally the application is linked into a shared library:

```
$ libtool --mode=link gcc -g twoDsim_m2.lo twoDsim_wrap.lo \
  roots.lo deviceGnuPic.lo \
   -L${prefix}/lib64 \
   -rpath `pwd` -lgm2 -lstdc++ -lm -o libtwoDsim.la
cp .libs/libtwoDsim.so _twoDsim.so
```

The library name must start with `_` to comply with the Python3 module naming scheme.

15.15 Interfacing GNU Modula-2 to C

The GNU Modula-2 compiler tries to use the C calling convention wherever possible however some parameters have no C equivalent and thus a language specific method is used. For example unbounded arrays are passed as a `struct {void *address, unsigned int high}` and the contents of these arrays are copied by callee functions when they are declared as non `VAR` parameters. The `VAR` equivalent unbounded array parameters need no copy, but still use the `struct` representation.

The recommended method of interfacing GNU Modula-2 to C is by telling the definition module that the implementation is in the C language. This is achieved by using the tokens `DEFINITION MODULE FOR "C"`. Here is an example `libprintf.def`.

```
DEFINITION MODULE FOR "C" libprintf ;

EXPORT UNQUALIFIED printf ;

PROCEDURE printf (a: ARRAY OF CHAR; ...) : [ INTEGER ] ;

END libprintf.
```

the `UNQUALIFIED` keyword in the definition module informs GNU Modula-2 not to prefix the module name to exported references in the object file.

The `printf` declaration states that the first parameter semantically matches `ARRAY OF CHAR` but since the module is for the C language it will be mapped onto `char *`. The token `...` indicates a variable number of arguments (varargs) and all parameters passed here are mapped onto their C equivalents. Arrays and constant strings are passed as pointers. Lastly `[ INTEGER ]` states that the caller can ignore the function return result if desired.

The hello world program can be rewritten as:

```
MODULE hello ;

FROM libprintf IMPORT printf ;

BEGIN
   printf("hello world\n")
END hello.
```

and it can be compiled by:

```
gm2 -g -I. hello.mod -lc
```

In reality the `-lc` is redundant as libc is always included in the linking process. It is shown here to emphasize that the C library or object file containing `printf` must be present.

If a procedure function is declared using varargs then some parameter values are converted. The table below summarises the default conversions and default types used.

```
Actual Parameter       |  Default conversion  |   Type of actual
                       |                      |   value passed
================================================================
```

(continues on next page)

(continued from previous page)

```
123                     |  none               |   long long int
"hello world"           |  none               |   const char *
a: ARRAY OF CHAR        |  ADR(a)             |   char *
a: ARRAY [0..5] OF CHAR |  ADR(a)             |   char *
3.14                    |  none               |   long double
```

If you wish to pass `int` values then you should explicitly convert the constants using one of the conversion mechanisms. For example: `INTEGER(10)` or `VAL(INTEGER, 10)` or `CAST(INTEGER, 10)`.

15.16 Interface to assembly language

The interface for GNU Modula-2 to assembly language is almost identical to GNU C. The only alterations are that the keywords `asm` and `volatile` are in capitals, following the Modula-2 convention.

A simple, but highly non optimal, example is given below. Here we want to add the two `CARDINAL`s `foo` and `bar` together and return the result. The target processor is assumed to be executing the x86_64 instruction set.

```
PROCEDURE Example (foo, bar: CARDINAL) : CARDINAL ;
VAR
   myout: CARDINAL ;
BEGIN
   ASM VOLATILE ("movq %1,%%rax; addq %2,%%rax; movq %%rax,%0"
      : "=rm" (myout)            (* outputs *)
      : "rm" (foo), "rm" (bar)   (* inputs  *)
      : "rax") ;                 (* we trash *)
   RETURN( myout )
END Example ;
```

For a full description of this interface we refer the reader to the GNU C manual.

See Extended Asm - Assembler Instructions with C Expression Operands.

The same example can be written using the newer extensions of naming the operands rather than using numbered arguments.

```
PROCEDURE Example (foo, bar: CARDINAL) : CARDINAL ;
VAR
   myout: CARDINAL ;
BEGIN
   ASM VOLATILE (
    "movq %[left],%%rax; addq %[right],%%rax; movq %%rax,%[output]"
      : [output] "=rm" (myout)                  (* outputs *)
      : [left] "rm" (foo), [right] "rm" (bar)   (* inputs  *)
      : "rax") ;                                (* we trash *)
   RETURN( myout )
END Example ;
```

Both examples generate exactly the same code. It is worth noting that the specifier 'rm' indicates that the operand can be either a register or memory. Of course you must choose an instruction which can take either, but this allows the compiler to take make more efficient choices depending upon the optimization level given to the compiler.

15.17 Data type alignment

GNU Modula-2 allows you to specify alignment for types and variables. The syntax for alignment is to use the ISO pragma directives `<* bytealignment ( expression )` and `*>`. These directives can be used after type and variable declarations.

The ebnf of the alignment production is:

```
Alignment := [ ByteAlignment ] =:
ByteAlignment := '<*' AttributeExpression '*>' =:
AlignmentExpression := "(" ConstExpression ")" =:
```

The `Alignment` ebnf statement may be used during contruction of types, records, record fields, arrays, pointers and variables. Below is an example of aligning a type so that the variable `bar` is aligned on a 1024 address.

```
MODULE align ;

TYPE
   foo = INTEGER <* bytealignment(1024) *> ;

VAR
   z  : INTEGER ;
   bar: foo ;
BEGIN
END align.
```

The next example aligns a variable on a 1024 byte boundary.

```
MODULE align2 ;

VAR
   x  : CHAR ;
   z  : ARRAY [0..255] OF INTEGER <* bytealignment(1024) *> ;
BEGIN
END align2.
```

Here the example aligns a pointer on a 1024 byte boundary.

```
MODULE align4 ;

FROM SYSTEM IMPORT ADR ;
FROM libc IMPORT exit ;

VAR
```

(continues on next page)

(continued from previous page)

```
   x  : CHAR ;
   z  : POINTER TO INTEGER <* bytealignment(1024) *> ;
BEGIN
   IF ADR(z) MOD 1024=0
   THEN
      exit(0)
   ELSE
      exit(1)
   END
END align4.
```

In example `align5` record field `y` is aligned on a 1024 byte boundary.

```
MODULE align5 ;

FROM SYSTEM IMPORT ADR ;
FROM libc IMPORT exit ;

TYPE
   rec = RECORD
            x: CHAR ;
            y: CHAR <* bytealignment(1024) *> ;
         END ;
VAR
   r: rec ;
BEGIN
   IF ADR(r.y) MOD 1024=0
   THEN
      exit(0)
   ELSE
      exit(1)
   END
END align5.
```

In the example below module `align6` declares `foo` as an array of 256 `INTEGER` s. The array `foo` is aligned on a 1024 byte boundary.

```
MODULE align6 ;

FROM SYSTEM IMPORT ADR ;
FROM libc IMPORT exit ;

TYPE
   foo = ARRAY [0..255] OF INTEGER <* bytealignment(1024) *> ;

VAR
   x  : CHAR ;
   z  : foo ;
BEGIN
   IF ADR(z) MOD 1024=0
   THEN
```

(continues on next page)

(continued from previous page)

```
      exit(0)
   ELSE
      exit(1)
   END
END align6.
```

15.18 Packing data types

The pragma `<* bytealignment(0) *>` can be used to specify that the fields within a `RECORD` are to be packed. Currently this only applies to fields which are declared as subranges, ordinal types and enumerated types. Here is an example of how two subranges might be packed into a byte.

```
TYPE
   bits3c =  [0..7] ;
   bits3i = [-4..3] ;

   byte = RECORD
             <* bytealignment(0) *>
             x: bits3c ;
             <* bitsunused(2) *>
             y: bits3i ;
          END ;
```

Notice that the user has specified that in between fields `x` and `y` there are two bits unused.

Now the user wishes to create a record with byte numbers zero and one occupied and then an `INTEGER32` field which is four byte aligned. In this case byte numbers two and three will be unused. The pragma `bytealignment` can be issued at the start of the record indicating the default alignment for the whole record and this can be overridden by individual fields if necessary.

```
rec = RECORD
         <* bytealignment (1) *> ;
         a, b: byte ;
         x: INTEGER32 <* bytealignment(4) *> ;
      END ;
```

In the following example the user has specified that a record has two fields `p` and `q` but that there are three bytes unused between these fields.

```
header = RECORD
            <* bytealignment(1) *>
            p: byte ;
            <* bytesunused(3) *>
            q: byte ;
         END ;
```

The pragma `<* bytesunused(x) *>` can only be used if the current field is on a byte boundary. There is also a `SYSTEM` pseudo procedure function `TBITSIZE(T)` which returns the minimum number of bits necessary to represent type `T`.

Another example of packing record bit fields is given below:

```
MODULE align21 ;

FROM libc IMPORT exit ;

TYPE
   colour = (red, blue, green, purple, white, black) ;

   soc = PACKEDSET OF colour ;

   rec = RECORD
            <* bytealignment(0) *>
            x: soc ;
            y: [-1..1] ;
         END ;

VAR
   r: rec ;
   v: CARDINAL ;
BEGIN
   v := SIZE(r) ;
   IF SIZE(r)#1
   THEN
      exit(1)
   END ;
   r.x := soc{blue} ;
   IF r.x#soc{blue}
   THEN
      exit(2)
   END
END align21.
```

Here we see that the total size of this record is one byte and consists of a six bit set type followed by a 2 bit integer subrange.

15.19 Accessing GNU Modula-2 Built-ins

This section describes the built-in constants and functions defined in GNU Modula-2. The following compiler constants can be accessed using the `__ATTRIBUTE__ __BUILTIN__` keywords. These are not part of the Modula-2 language and they may differ depending upon the target architecture but they provide a method whereby common libraries can interface to a different underlying architecture.

The built-in constants are: `BITS_PER_UNIT`, `BITS_PER_WORD`, `BITS_PER_CHAR` and `UNITS_PER_WORD`. They are integrated into GNU Modula-2 by an extension to the `ConstFactor` rule:

```
ConstFactor := ConstQualidentOrSet | Number | ConstString |
               "(" ConstExpression ")" | "NOT" ConstFactor |
               ConstAttribute =:

ConstAttribute := "__ATTRIBUTE__" "__BUILTIN__" "(" "(" Ident ")" ")" =:
```

Here is an example taken from the ISO library `SYSTEM.def` :

```
CONST
   BITSPERLOC    = __ATTRIBUTE__ __BUILTIN__ ((BITS_PER_UNIT)) ;
   LOCSPERWORD   = __ATTRIBUTE__ __BUILTIN__ ((UNITS_PER_WORD)) ;
```

Built-in functions are transparent to the end user. All built-in functions are declared in `DEFINITION MODULE` s and are imported as and when required. Built-in functions are declared in definition modules by using the `__BUILTIN__` keyword. Here is a section of the ISO library `LongMath.def` which demonstrates this feature.

```
PROCEDURE __BUILTIN__ sqrt (x: LONGREAL): LONGREAL;
  (* Returns the square root of x *)
```

This indicates that the function `sqrt` will be implemented using the gcc built-in maths library. If gcc cannot utilise the built-in function (for example if the programmer requested the address of `sqrt`) then code is generated to call the alternative function implemented in the `IMPLEMENTATION MODULE`.

Sometimes a function exported from the `DEFINITION MODULE` will have a different name from the built-in function within gcc. In such cases the mapping between the GNU Modula-2 function name and the gcc name is expressed using the keywords `__ATTRIBUTE__ __BUILTIN__ ((Ident))`. For example the function `sqrt` in `LongMath.def` maps onto the gcc built-in function `sqrtl` and this is expressed as:

```
PROCEDURE __ATTRIBUTE__ __BUILTIN__ ((sqrtl)) sqrt
                                    (x: LONGREAL) : LONGREAL;
  (* Returns the positive square root of x *)
```

The following module `Builtins.def` enumerates the list of built-in functions which can be accessed in GNU Modula-2. It also serves to define the parameter and return value for each function:

```
DEFINITION MODULE Builtins ;

FROM SYSTEM IMPORT ADDRESS ;

(* floating point intrinsic procedure functions *)

isfinitef
PROCEDURE __BUILTIN__ isfinitef (x: SHORTREAL) : BOOLEAN ;
isfinite
PROCEDURE __BUILTIN__ isfinite (x: REAL) : BOOLEAN ;
isfinitel
PROCEDURE __BUILTIN__ isfinitel (x: LONGREAL) : BOOLEAN ;

sinf
PROCEDURE __BUILTIN__ sinf (x: SHORTREAL) : SHORTREAL ;
sin
PROCEDURE __BUILTIN__ sin (x: REAL) : REAL ;
sinl
PROCEDURE __BUILTIN__ sinl (x: LONGREAL) : LONGREAL ;
```

(continues on next page)

(continued from previous page)

```
cosf
PROCEDURE __BUILTIN__ cosf (x: SHORTREAL) : SHORTREAL ;
cos
PROCEDURE __BUILTIN__ cos (x: REAL) : REAL ;
cosl
PROCEDURE __BUILTIN__ cosl (x: LONGREAL) : LONGREAL ;

sqrtf
PROCEDURE __BUILTIN__ sqrtf (x: SHORTREAL) : SHORTREAL ;
sqrt
PROCEDURE __BUILTIN__ sqrt (x: REAL) : REAL ;
sqrtl
PROCEDURE __BUILTIN__ sqrtl (x: LONGREAL) : LONGREAL ;

atan2f
PROCEDURE __BUILTIN__ atan2f (x, y: SHORTREAL) : SHORTREAL ;
atan2
PROCEDURE __BUILTIN__ atan2 (x, y: REAL) : REAL ;
atan2l
PROCEDURE __BUILTIN__ atan2l (x, y: LONGREAL) : LONGREAL ;

fabsf
PROCEDURE __BUILTIN__ fabsf (x: SHORTREAL) : SHORTREAL ;
fabs
PROCEDURE __BUILTIN__ fabs (x: REAL) : REAL ;
fabsl
PROCEDURE __BUILTIN__ fabsl (x: LONGREAL) : LONGREAL ;

logf
PROCEDURE __BUILTIN__ logf (x: SHORTREAL) : SHORTREAL ;
log
PROCEDURE __BUILTIN__ log (x: REAL) : REAL ;
logl
PROCEDURE __BUILTIN__ logl (x: LONGREAL) : LONGREAL ;

expf
PROCEDURE __BUILTIN__ expf (x: SHORTREAL) : SHORTREAL ;
exp
PROCEDURE __BUILTIN__ exp (x: REAL) : REAL ;
expl
PROCEDURE __BUILTIN__ expl (x: LONGREAL) : LONGREAL ;

log10f
PROCEDURE __BUILTIN__ log10f (x: SHORTREAL) : SHORTREAL ;
log10
PROCEDURE __BUILTIN__ log10 (x: REAL) : REAL ;
log10l
PROCEDURE __BUILTIN__ log10l (x: LONGREAL) : LONGREAL ;

exp10f
```

(continues on next page)

(continued from previous page)

```
PROCEDURE __BUILTIN__ exp10f (x: SHORTREAL) : SHORTREAL ;
exp10
PROCEDURE __BUILTIN__ exp10 (x: REAL) : REAL ;
exp10l
PROCEDURE __BUILTIN__ exp10l (x: LONGREAL) : LONGREAL ;

ilogbf
PROCEDURE __BUILTIN__ ilogbf (x: SHORTREAL) : INTEGER ;
ilogb
PROCEDURE __BUILTIN__ ilogb (x: REAL) : INTEGER ;
ilogbl
PROCEDURE __BUILTIN__ ilogbl (x: LONGREAL) : INTEGER ;

huge_val
PROCEDURE __BUILTIN__ huge_val () : REAL ;
huge_valf
PROCEDURE __BUILTIN__ huge_valf () : SHORTREAL ;
huge_vall
PROCEDURE __BUILTIN__ huge_vall () : LONGREAL ;

significand
PROCEDURE __BUILTIN__ significand (r: REAL) : REAL ;
significandf
PROCEDURE __BUILTIN__ significandf (s: SHORTREAL) : SHORTREAL ;
significandl
PROCEDURE __BUILTIN__ significandl (l: LONGREAL) : LONGREAL ;

modf
PROCEDURE __BUILTIN__ modf (x: REAL; VAR y: REAL) : REAL ;
modff
PROCEDURE __BUILTIN__ modff (x: SHORTREAL;
                             VAR y: SHORTREAL) : SHORTREAL ;
modfl
PROCEDURE __BUILTIN__ modfl (x: LONGREAL; VAR y: LONGREAL) : LONGREAL ;

signbit
PROCEDURE __BUILTIN__ signbit (r: REAL) : INTEGER ;
signbitf
PROCEDURE __BUILTIN__ signbitf (s: SHORTREAL) : INTEGER ;
signbitl
PROCEDURE __BUILTIN__ signbitl (l: LONGREAL) : INTEGER ;

nextafter
PROCEDURE __BUILTIN__ nextafter (x, y: REAL) : REAL ;
nextafterf
PROCEDURE __BUILTIN__ nextafterf (x, y: SHORTREAL) : SHORTREAL ;
nextafterl
PROCEDURE __BUILTIN__ nextafterl (x, y: LONGREAL) : LONGREAL ;

nexttoward
PROCEDURE __BUILTIN__ nexttoward (x, y: REAL) : LONGREAL ;
```

(continues on next page)

(continued from previous page)

```
nexttowardf
PROCEDURE __BUILTIN__ nexttowardf (x, y: SHORTREAL) : LONGREAL ;
nexttowardl
PROCEDURE __BUILTIN__ nexttowardl (x, y: LONGREAL) : LONGREAL ;

scalb
PROCEDURE __BUILTIN__ scalb (x, n: REAL) : REAL ;
scalbf
PROCEDURE __BUILTIN__ scalbf (x, n: SHORTREAL) : SHORTREAL ;
scalbl
PROCEDURE __BUILTIN__ scalbl (x, n: LONGREAL) : LONGREAL ;

scalbln
PROCEDURE __BUILTIN__ scalbln (x: REAL; n: LONGINT) : REAL ;
scalblnf
PROCEDURE __BUILTIN__ scalblnf (x: SHORTREAL; n: LONGINT) : SHORTREAL ;
scalblnl
PROCEDURE __BUILTIN__ scalblnl (x: LONGREAL; n: LONGINT) : LONGREAL ;

scalbn
PROCEDURE __BUILTIN__ scalbn (x: REAL; n: INTEGER) : REAL ;
scalbnf
PROCEDURE __BUILTIN__ scalbnf (x: SHORTREAL; n: INTEGER) : SHORTREAL ;
scalbnl
PROCEDURE __BUILTIN__ scalbnl (x: LONGREAL; n: INTEGER) : LONGREAL ;

(* complex arithmetic intrincic procedure functions *)

cabsf
PROCEDURE __BUILTIN__ cabsf (z: SHORTCOMPLEX) : SHORTREAL ;
cabs
PROCEDURE __BUILTIN__ cabs (z: COMPLEX) : REAL ;
cabsl
PROCEDURE __BUILTIN__ cabsl (z: LONGCOMPLEX) : LONGREAL ;

cargf
PROCEDURE __BUILTIN__ cargf (z: SHORTCOMPLEX) : SHORTREAL ;
carg
PROCEDURE __BUILTIN__ carg (z: COMPLEX) : REAL ;
cargl
PROCEDURE __BUILTIN__ cargl (z: LONGCOMPLEX) : LONGREAL ;

conjf
PROCEDURE __BUILTIN__ conjf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
conj
PROCEDURE __BUILTIN__ conj (z: COMPLEX) : COMPLEX ;
conjl
PROCEDURE __BUILTIN__ conjl (z: LONGCOMPLEX) : LONGCOMPLEX ;

cpowerf
PROCEDURE __BUILTIN__ cpowerf (base: SHORTCOMPLEX;
```

(continues on next page)

(continued from previous page)

```
                              exp: SHORTREAL) : SHORTCOMPLEX ;
cpower
PROCEDURE __BUILTIN__ cpower (base: COMPLEX; exp: REAL) : COMPLEX ;
cpowerl
PROCEDURE __BUILTIN__ cpowerl (base: LONGCOMPLEX;
                               exp: LONGREAL) : LONGCOMPLEX ;

csqrtf
PROCEDURE __BUILTIN__ csqrtf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
csqrt
PROCEDURE __BUILTIN__ csqrt (z: COMPLEX) : COMPLEX ;
csqrtl
PROCEDURE __BUILTIN__ csqrtl (z: LONGCOMPLEX) : LONGCOMPLEX ;

cexpf
PROCEDURE __BUILTIN__ cexpf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
cexp
PROCEDURE __BUILTIN__ cexp (z: COMPLEX) : COMPLEX ;
cexpl
PROCEDURE __BUILTIN__ cexpl (z: LONGCOMPLEX) : LONGCOMPLEX ;

clnf
PROCEDURE __BUILTIN__ clnf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
cln
PROCEDURE __BUILTIN__ cln (z: COMPLEX) : COMPLEX ;
clnl
PROCEDURE __BUILTIN__ clnl (z: LONGCOMPLEX) : LONGCOMPLEX ;

csinf
PROCEDURE __BUILTIN__ csinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
csin
PROCEDURE __BUILTIN__ csin (z: COMPLEX) : COMPLEX ;
csinl
PROCEDURE __BUILTIN__ csinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

ccosf
PROCEDURE __BUILTIN__ ccosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
ccos
PROCEDURE __BUILTIN__ ccos (z: COMPLEX) : COMPLEX ;
ccosl
PROCEDURE __BUILTIN__ ccosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

ctanf
PROCEDURE __BUILTIN__ ctanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
ctan
PROCEDURE __BUILTIN__ ctan (z: COMPLEX) : COMPLEX ;
ctanl
PROCEDURE __BUILTIN__ ctanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

carcsinf
PROCEDURE __BUILTIN__ carcsinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
```

(continues on next page)

(continued from previous page)

```
carcsin
PROCEDURE __BUILTIN__ carcsin (z: COMPLEX) : COMPLEX ;
carcsinl
PROCEDURE __BUILTIN__ carcsinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

carccosf
PROCEDURE __BUILTIN__ carccosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
carccos
PROCEDURE __BUILTIN__ carccos (z: COMPLEX) : COMPLEX ;
carccosl
PROCEDURE __BUILTIN__ carccosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

carctanf
PROCEDURE __BUILTIN__ carctanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
carctan
PROCEDURE __BUILTIN__ carctan (z: COMPLEX) : COMPLEX ;
carctanl
PROCEDURE __BUILTIN__ carctanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

(* memory and string intrincic procedure functions *)

alloca
PROCEDURE __BUILTIN__ alloca (i: CARDINAL) : ADDRESS ;
memcpy
PROCEDURE __BUILTIN__ memcpy (dest, src: ADDRESS;
                              nbytes: CARDINAL) : ADDRESS ;
index
PROCEDURE __BUILTIN__ index (s: ADDRESS; c: INTEGER) : ADDRESS ;
rindex
PROCEDURE __BUILTIN__ rindex (s: ADDRESS; c: INTEGER) : ADDRESS ;
memcmp
PROCEDURE __BUILTIN__ memcmp (s1, s2: ADDRESS;
                              nbytes: CARDINAL) : INTEGER ;
memset
PROCEDURE __BUILTIN__ memset (s: ADDRESS; c: INTEGER;
                              nbytes: CARDINAL) : ADDRESS ;
memmove
PROCEDURE __BUILTIN__ memmove (s1, s2: ADDRESS;
                               nbytes: CARDINAL) : ADDRESS ;
strcat
PROCEDURE __BUILTIN__ strcat (dest, src: ADDRESS) : ADDRESS ;
strncat
PROCEDURE __BUILTIN__ strncat (dest, src: ADDRESS;
                               nbytes: CARDINAL) : ADDRESS ;
strcpy
PROCEDURE __BUILTIN__ strcpy (dest, src: ADDRESS) : ADDRESS ;
strncpy
PROCEDURE __BUILTIN__ strncpy (dest, src: ADDRESS;
                               nbytes: CARDINAL) : ADDRESS ;
strcmp
PROCEDURE __BUILTIN__ strcmp (s1, s2: ADDRESS) : INTEGER ;
```

(continues on next page)

(continued from previous page)

```
strncmp
PROCEDURE __BUILTIN__ strncmp (s1, s2: ADDRESS;
                               nbytes: CARDINAL) : INTEGER ;
strlen
PROCEDURE __BUILTIN__ strlen (s: ADDRESS) : INTEGER ;
strstr
PROCEDURE __BUILTIN__ strstr (haystack, needle: ADDRESS) : ADDRESS ;
strpbrk
PROCEDURE __BUILTIN__ strpbrk (s, accept: ADDRESS) : ADDRESS ;
strspn
PROCEDURE __BUILTIN__ strspn (s, accept: ADDRESS) : CARDINAL ;
strcspn
PROCEDURE __BUILTIN__ strcspn (s, accept: ADDRESS) : CARDINAL ;
strchr
PROCEDURE __BUILTIN__ strchr (s: ADDRESS; c: INTEGER) : ADDRESS ;
strrchr
PROCEDURE __BUILTIN__ strrchr (s: ADDRESS; c: INTEGER) : ADDRESS ;

(*
   longjmp - this GCC builtin restricts the val to always 1.
*)
(* do not use these two builtins, as gcc, only really
   anticipates that the Ada front end should use them
   and it only uses them in its runtime exception handling.
   We leave them here in the hope that someday they will
   behave more like their libc counterparts.  *)

longjmp
PROCEDURE __BUILTIN__ longjmp (env: ADDRESS; val: INTEGER) ;
setjmp
PROCEDURE __BUILTIN__ setjmp (env: ADDRESS) : INTEGER ;

(*
   frame_address - returns the address of the frame.
                   The current frame is obtained if level is 0,
                   the next level up if level is 1 etc.
*)

frame_address
PROCEDURE __BUILTIN__ frame_address (level: CARDINAL) : ADDRESS ;

(*
   return_address - returns the return address of function.
                    The current function return address is
                    obtained if level is 0,
                    the next level up if level is 1 etc.
*)

return_address
PROCEDURE __BUILTIN__ return_address (level: CARDINAL) : ADDRESS ;
```

(continues on next page)

(continued from previous page)

```
(*
   alloca_trace - this is a no-op which is used for internal debugging.
*)

alloca_trace
PROCEDURE alloca_trace (returned: ADDRESS; nBytes: CARDINAL) : ADDRESS ;

END Builtins.
```

Although this module exists and will result in the generation of in-line code if optimization flags are passed to GNU Modula-2, users are advised to utilize the same functions from more generic libraries. The built-in mechanism will be applied to these generic libraries where appropriate. Note for the mathematical routines to be in-lined you need to specify the `-ffast-math -O` options.

15.20 The PIM system module

```
DEFINITION MODULE SYSTEM ;

EXPORT QUALIFIED BITSPERBYTE, BYTESPERWORD,
                 LOC, WORD, BYTE, ADDRESS, INTEGER8,
                 INTEGER16, INTEGER32, INTEGER64, CARDINAL8,
                 CARDINAL16, CARDINAL32, CARDINAL64, WORD16,
                 WORD32, WORD64, BITSET8, BITSET16,
                 BITSET32, REAL32, REAL64, REAL128,
                 COMPLEX32, COMPLEX64, COMPLEX128, CSIZE_T,
                 CSSIZE_T,
                 ADR, TSIZE, ROTATE, SHIFT, THROW, TBITSIZE ;
                 (* SIZE is also exported if -fpim2 is used *)

CONST
BITSPERBYTE    (const)
  BITSPERBYTE   = __ATTRIBUTE__ __BUILTIN__ ((BITS_PER_UNIT)) ;
BYTESPERWORD   (const)
  BYTESPERWORD  = __ATTRIBUTE__ __BUILTIN__ ((UNITS_PER_WORD)) ;

(* all the following types are declared internally to gm2
TYPE
LOC (type)
   LOC ;
WORD (type)
   WORD ;
BYTE (type)
   BYTE ;
ADDRESS (type)
   ADDRESS ;
INTEGER8 (type)
   INTEGER8 ;
INTEGER16 (type)
   INTEGER16 ;
```

(continues on next page)

(continued from previous page)

```
INTEGER32 (type)
   INTEGER32 ;
INTEGER64 (type)
   INTEGER64 ;
CARDINAL8 (type)
   CARDINAL8 ;
CARDINAL16 (type)
   CARDINAL16 ;
CARDINAL32 (type)
   CARDINAL32 ;
CARDINAL64 (type)
   CARDINAL64 ;
WORD16 (type)
   WORD16 ;
WORD32 (type)
   WORD32 ;
WORD64 (type)
   WORD64 ;
BITSET8 (type)
   BITSET8 ;
BITSET16 (type)
   BITSET16 ;
BITSET32 (type)
   BITSET32 ;
REAL32 (type)
   REAL32 ;
REAL64 (type)
   REAL64 ;
REAL128 (type)
   REAL128 ;
COMPLEX32 (type)
   COMPLEX32 ;
COMPLEX64 (type)
   COMPLEX64 ;
COMPLEX128 (type)
   COMPLEX128 ;
CSIZE_T (type)
   CSIZE_T ;
CSSIZE_T (type)
   CSSIZE_T ;
*)

(*
   all the functions below are declared internally to gm2
   ======================================================

ADR
PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
  (* Returns the address of variable v. *)

SIZE
```

(continues on next page)

(continued from previous page)

```
PROCEDURE SIZE (v: <type>) : ZType;
  (* Returns the number of BYTES used to store a v of
     any specified <type>.  Only available if -fpim2 is used.
  *)

TSIZE
PROCEDURE TSIZE (<type>) : CARDINAL;
  (* Returns the number of BYTES used to store a value of the
     specified <type>.
  *)

ROTATE
PROCEDURE ROTATE (val: <a set type>;
                  num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by rotating up/right
     or down/right by the absolute value of num.  The direction is
     down/right if the sign of num is negative, otherwise the direction
     is up/left.
  *)

SHIFT
PROCEDURE SHIFT (val: <a set type>;
                 num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by shifting up/left
     or down/right by the absolute value of num, introducing
     zeros as necessary.  The direction is down/right if the sign of
     num is negative, otherwise the direction is up/left.
  *)

THROW
PROCEDURE THROW (i: INTEGER) ;
  (*
     THROW is a GNU extension and was not part of the PIM or ISO
     standards.  It throws an exception which will be caught by the
     EXCEPT block (assuming it exists).  This is a compiler builtin
     function which interfaces to the GCC exception handling runtime
     system.
     GCC uses the term throw, hence the naming distinction between
     the GCC builtin and the Modula-2 runtime library procedure Raise.
     The later library procedure Raise will call SYSTEM.THROW after
     performing various housekeeping activities.
  *)

TBITSIZE
PROCEDURE TBITSIZE (<type>) : CARDINAL ;
  (* Returns the minimum number of bits necessary to represent
     <type>.  This procedure function is only useful for determining
     the number of bits used for any type field within a packed RECORD.
     It is not particularly useful elsewhere since <type> might be
     optimized for speed, for example a BOOLEAN could occupy a WORD.
  *)
```

(continues on next page)

(continued from previous page)

```
*)

(* The following procedures are invoked by GNU Modula-2 to
   shift non word sized set types. They are not strictly part
   of the core PIM Modula-2, however they are used by
   GNU Modula-2 to implement the SHIFT procedure defined above,
   which are in turn used by the Logitech compatible libraries.

   Users will access these procedures by using the procedure
   SHIFT above and GNU Modula-2 will map SHIFT onto one of
   the following procedures.
*)

(*
   ShiftVal - is a runtime procedure whose job is to implement
              the SHIFT procedure of ISO SYSTEM. GNU Modula-2 will
              inline a SHIFT of a single WORD sized set and will only
              call this routine for larger sets.
*)

ShiftVal
PROCEDURE ShiftVal (VAR s, d: ARRAY OF BITSET;
                    SetSizeInBits: CARDINAL;
                    ShiftCount: INTEGER) ;

(*
   ShiftLeft - performs the shift left for a multi word set.
               This procedure might be called by the back end of
               GNU Modula-2 depending whether amount is known at
               compile time.
*)

ShiftLeft
PROCEDURE ShiftLeft (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     ShiftCount: CARDINAL) ;

(*
   ShiftRight - performs the shift left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

ShiftRight
PROCEDURE ShiftRight (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      ShiftCount: CARDINAL) ;

(*
   RotateVal - is a runtime procedure whose job is to implement
```

(continues on next page)

(continued from previous page)

```
               the ROTATE procedure of ISO SYSTEM. GNU Modula-2 will
               inline a ROTATE of a single WORD (or less)
               sized set and will only call this routine for larger
               sets.
*)

RotateVal
PROCEDURE RotateVal (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     RotateCount: INTEGER) ;

(*
   RotateLeft - performs the rotate left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

RotateLeft
PROCEDURE RotateLeft (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      RotateCount: CARDINAL) ;

(*
   RotateRight - performs the rotate right for a multi word set.
                 This procedure might be called by the back end of
                 GNU Modula-2 depending whether amount is known at
                 compile time.
*)

RotateRight
PROCEDURE RotateRight (VAR s, d: ARRAY OF BITSET;
                       SetSizeInBits: CARDINAL;
                       RotateCount: CARDINAL) ;

END SYSTEM.
```

The different dialects of Modula-2 PIM-[234] and ISO Modula-2 declare the function `SIZE` in different places. PIM-[34] and ISO Modula-2 declare `SIZE` as a pervasive function (declared in the base module). PIM-2 defined `SIZE` in the `SYSTEM` module (as shown above).

GNU Modula-2 allows users to specify the dialect of Modula-2 by using the `-fiso` and `-fpim2` command line switches.

The data types `CSIZE_T` and `CSSIZE_T` are also exported from the `SYSTEM` module. The type `CSIZE_T` is unsigned and is mapped onto the target C data type `size_t` whereas the type `CSSIZE_T` is mapped onto the signed C data type `ssize_t`.

It is anticipated that these should only be used to provide cross platform definition modules for C libraries.

There are also a variety of fixed sized `INTEGER` and `CARDINAL` types. The variety of the fixed sized

types will depend upon the target architecture.

15.21 The ISO system module

```
DEFINITION MODULE SYSTEM;

  (* Gives access to system programming facilities that are probably
     non portable. *)

  (* The constants and types define underlying properties of storage *)

EXPORT QUALIFIED BITSPERLOC, LOCSPERWORD,
                 LOC, ADDRESS, BYTE, WORD, INTEGER8,
                 INTEGER16, INTEGER32, INTEGER64, CARDINAL8,
                 CARDINAL16, CARDINAL32, CARDINAL64, WORD16,
                 WORD32, WORD64, BITSET8, BITSET16,
                 BITSET32, REAL32, REAL64, REAL128,
                 COMPLEX32, COMPLEX64, COMPLEX128, CSIZE_T,
                 CSSIZE_T,
                 ADDADR, SUBADR, DIFADR, MAKEADR, ADR, ROTATE,
                 SHIFT, CAST, TSIZE,

                 (* Internal GM2 compiler functions *)
                 ShiftVal, ShiftLeft, ShiftRight,
                 RotateVal, RotateLeft, RotateRight,
                 THROW, TBITSIZE ;

CONST
                  (* <implementation-defined constant> ; *)
BITSPERLOC     (const)
  BITSPERLOC    = __ATTRIBUTE__ __BUILTIN__ ((BITS_PER_UNIT)) ;
                  (* <implementation-defined constant> ; *)
LOCSPERWORD    (const)
  LOCSPERWORD   = __ATTRIBUTE__ __BUILTIN__ ((UNITS_PER_WORD)) ;
                  (* <implementation-defined constant> ; *)
LOCSPERBYTE  (const)
  LOCSPERBYTE = 8 DIV BITSPERLOC ;

(*
   all the objects below are declared internally to gm2
   ===================================================

TYPE
LOC (type)
   LOC ;
ADDRESS (type)
   ADDRESS ;
BYTE (type)
   BYTE ;
WORD (type)
   WORD ;
```

(continues on next page)

(continued from previous page)

```
INTEGER8 (type)
   INTEGER8 ;
INTEGER16 (type)
   INTEGER16 ;
INTEGER32 (type)
   INTEGER32 ;
INTEGER64 (type)
   INTEGER64 ;
CARDINAL8 (type)
   CARDINAL8 ;
CARDINAL16 (type)
   CARDINAL16 ;
CARDINAL32 (type)
   CARDINAL32 ;
CARDINAL64 (type)
   CARDINAL64 ;
WORD16 (type)
   WORD16 ;
WORD32 (type)
   WORD32 ;
WORD64 (type)
   WORD64 ;
BITSET8 (type)
   BITSET8 ;
BITSET16 (type)
   BITSET16 ;
BITSET32 (type)
   BITSET32 ;
REAL32 (type)
   REAL32 ;
REAL64 (type)
   REAL64 ;
REAL128 (type)
   REAL128 ;
COMPLEX32 (type)
   COMPLEX32 ;
COMPLEX64 (type)
   COMPLEX64 ;
COMPLEX128 (type)
   COMPLEX128 ;
CSIZE_T (type)
   CSIZE_T ;
CSSIZE_T (type)
   CSSIZE_T ;

TYPE
  LOC; (* A system basic type. Values are the uninterpreted
          contents of the smallest addressable unit of storage *)
ADDRESS (type)
  ADDRESS = POINTER TO LOC;
WORD (type)
```

(continues on next page)

(continued from previous page)

```
  WORD = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

  (* BYTE and LOCSPERBYTE are provided if appropriate for machine *)

TYPE
BYTE (type)
  BYTE = ARRAY [0 .. LOCSPERBYTE-1] OF LOC;

ADDADR
PROCEDURE ADDADR (addr: ADDRESS; offset: CARDINAL): ADDRESS;
  (* Returns address given by (addr + offset), or may raise
     an exception if this address is not valid.
  *)

SUBADR
PROCEDURE SUBADR (addr: ADDRESS; offset: CARDINAL): ADDRESS;
  (* Returns address given by (addr - offset), or may raise an
     exception if this address is not valid.
  *)

DIFADR
PROCEDURE DIFADR (addr1, addr2: ADDRESS): INTEGER;
  (* Returns the difference between addresses (addr1 - addr2),
     or may raise an exception if the arguments are invalid
     or address space is non-contiguous.
  *)

MAKEADR
PROCEDURE MAKEADR (high: <some type>; ...): ADDRESS;
  (* Returns an address constructed from a list of values whose
     types are implementation-defined, or may raise an
     exception if this address is not valid.

     In GNU Modula-2, MAKEADR can take any number of arguments
     which are mapped onto the type ADDRESS. The first parameter
     maps onto the high address bits and subsequent parameters map
     onto lower address bits. For example:

     a := MAKEADR(BYTE(0FEH), BYTE(0DCH), BYTE(0BAH), BYTE(098H),
                  BYTE(076H), BYTE(054H), BYTE(032H), BYTE(010H)) ;

     then the value of, a, on a 64 bit machine is: 0FEDCBA9876543210H

     The parameters do not have to be the same type, but constants
     _must_ be typed.
  *)

ADR
PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
  (* Returns the address of variable v. *)
```

(continues on next page)

(continued from previous page)

```
ROTATE
PROCEDURE ROTATE (val: <a packedset type>;
                  num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by rotating up/right
     or down/right by the absolute value of num.  The direction is
     down/right if the sign of num is negative, otherwise the direction
     is up/left.
  *)

SHIFT
PROCEDURE SHIFT (val: <a packedset type>;
                 num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by shifting up/left
     or down/right by the absolute value of num, introducing
     zeros as necessary.  The direction is down/right if the sign of
     num is negative, otherwise the direction is up/left.
  *)

CAST
PROCEDURE CAST (<targettype>; val: <anytype>): <targettype>;
  (* CAST is a type transfer function.  Given the expression
     denoted by val, it returns a value of the type <targettype>.
     An invalid value for the target value or a
     physical address alignment problem may raise an exception.
  *)

TSIZE
PROCEDURE TSIZE (<type>; ... ): CARDINAL;
  (* Returns the number of LOCS used to store a value of the
     specified <type>.   The extra parameters, if present,
     are used to distinguish variants in a variant record.
  *)

THROW
PROCEDURE THROW (i: INTEGER) ;
  (*
     THROW is a GNU extension and was not part of the PIM or ISO
     standards.  It throws an exception which will be caught by the
     EXCEPT block (assuming it exists).  This is a compiler builtin
     function which interfaces to the GCC exception handling runtime
     system.
     GCC uses the term throw, hence the naming distinction between
     the GCC builtin and the Modula-2 runtime library procedure Raise.
     The later library procedure Raise will call SYSTEM.THROW after
     performing various housekeeping activities.
  *)

TBITSIZE
PROCEDURE TBITSIZE (<type>) : CARDINAL ;
  (* Returns the minimum number of bits necessary to represent
     <type>.  This procedure function is only useful for determining
```

(continues on next page)

(continued from previous page)

```
     the number of bits used for any type field within a packed RECORD.
     It is not particularly useful elsewhere since <type> might be
     optimized for speed, for example a BOOLEAN could occupy a WORD.
  *)
*)

(* The following procedures are invoked by GNU Modula-2 to
   shift non word set types. They are not part of ISO Modula-2
   but are used by GNU Modula-2 to implement the SHIFT procedure
   defined above. *)

(*
   ShiftVal - is a runtime procedure whose job is to implement
              the SHIFT procedure of ISO SYSTEM. GNU Modula-2 will
              inline a SHIFT of a single WORD sized set and will only
              call this routine for larger sets.
*)

ShiftVal
PROCEDURE ShiftVal (VAR s, d: ARRAY OF BITSET;
                    SetSizeInBits: CARDINAL;
                    ShiftCount: INTEGER) ;

(*
   ShiftLeft - performs the shift left for a multi word set.
               This procedure might be called by the back end of
               GNU Modula-2 depending whether amount is known at
               compile time.
*)

ShiftLeft
PROCEDURE ShiftLeft (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     ShiftCount: CARDINAL) ;

(*
   ShiftRight - performs the shift left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

ShiftRight
PROCEDURE ShiftRight (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      ShiftCount: CARDINAL) ;

(*
   RotateVal - is a runtime procedure whose job is to implement
               the ROTATE procedure of ISO SYSTEM. GNU Modula-2 will
               inline a ROTATE of a single WORD (or less)
```

(continues on next page)

(continued from previous page)

```
               sized set and will only call this routine for larger
               sets.
*)

RotateVal
PROCEDURE RotateVal (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     RotateCount: INTEGER) ;

(*
   RotateLeft - performs the rotate left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

RotateLeft
PROCEDURE RotateLeft (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      RotateCount: CARDINAL) ;

(*
   RotateRight - performs the rotate right for a multi word set.
                 This procedure might be called by the back end of
                 GNU Modula-2 depending whether amount is known at
                 compile time.
*)

RotateRight
PROCEDURE RotateRight (VAR s, d: ARRAY OF BITSET;
                       SetSizeInBits: CARDINAL;
                       RotateCount: CARDINAL) ;

END SYSTEM.
```

The data types `CSIZE_T` and `CSSIZE_T` are also exported from the `SYSTEM` module. The type `CSIZE_T` is unsigned and is mapped onto the target C data type `size_t` whereas the type `CSSIZE_T` is mapped onto the signed C data type `ssize_t`.

It is anticipated that these should only be used to provide cross platform definition modules for C libraries.

There are also a variety of fixed sized `INTEGER` and `CARDINAL` types. The variety of the fixed sized types will depend upon the target architecture.

This document contains the user and design issues relevant to the Modula-2 front end to gcc. Throughout this document the GNU Modula-2 front end is often referred to as `gm2-1.9.5` or `gm2` for short. This corresponds to GCC version 13.0.0 and GNU Modula-2 version 1.9.5.

CHAPTER

SIXTEEN

LICENCE OF GNU MODULA-2

GNU Modula-2 is free software, the compiler is held under the GPL v3 http://www.gnu.org/licenses/gpl.txt, its libraries (pim, iso and Logitech compatible) are under the GPL v3 with the GCC runtime library exception clause.

Under Section 7 of GPL version 3, you are granted additional permissions described in the GCC Runtime Library Exception, version 3.1, as published by the Free Software Foundation.

You should have received a copy of the GNU General Public License and a copy of the GCC Runtime Library Exception along with this program; see the files COPYING3 and COPYING.RUNTIME respectively. If not, see <http://www.gnu.org/licenses/>.

More information on how these licences work is available http://www.gnu.org/licenses/licenses.html on the GNU web site.

CHAPTER

SEVENTEEN

EBNF OF GNU MODULA-2

This chapter contains the EBNF of GNU Modula-2. This grammer currently supports both PIM and ISO dialects. The rules here are automatically extracted from the grammer files in GNU Modula-2 and serve to document the syntax of the extensions described earlier and how they fit in with the base language.

Note that the first six productions are built into the lexical analysis phase.

```
Ident := is a builtin and checks for an identifier
       =:
Ident (ebnf)
```

```
Integer := is a builtin and checks for an integer
         =:
Integer (ebnf)
```

```
Real := is a builtin and checks for an real constant
      =:
Real (ebnf)
```

```
string := is a builtin and checks for an string constant
        =:
string (ebnf)
```

```
FileUnit := ( DefinitionModule  |
              ImplementationOrProgramModule  )
          =:
FileUnit (ebnf)
```

```
ProgramModule := 'MODULE' Ident [ Priority  ] ';' {
   Import  } Block Ident '.'
              =:
ProgramModule (ebnf)
```

```
ImplementationModule := 'IMPLEMENTATION' 'MODULE' Ident
                        [ Priority  ] ';' { Import
                                          } Block
                        Ident '.'
```

(continues on next page)

(continued from previous page)

```
                      =:
ImplementationModule (ebnf)
```

```
ImplementationOrProgramModule := ImplementationModule  |
                                 ProgramModule
                               =:
ImplementationOrProgramModule (ebnf)
```

```
Number := Integer  | Real
        =:
Number (ebnf)
```

```
Qualident := Ident { '.' Ident  }
           =:
Qualident (ebnf)
```

```
ConstantDeclaration := Ident '=' ConstExpression
                     =:
ConstantDeclaration (ebnf)
```

```
ConstExpression := SimpleConstExpr [ Relation SimpleConstExpr  ]
                 =:
ConstExpression (ebnf)
```

```
Relation := '='  | '#'  | '<>'  | '<'  | '<='  |
            '>'  | '>='  | 'IN'
          =:
Relation (ebnf)
```

```
SimpleConstExpr := UnaryOrConstTerm { AddOperator
                                      ConstTerm  }
                 =:
SimpleConstExpr (ebnf)
```

```
UnaryOrConstTerm := '+' ConstTerm  |
                    '-' ConstTerm  |
                    ConstTerm
                  =:
UnaryOrConstTerm (ebnf)
```

```
AddOperator := '+'  | '-'  | 'OR'
             =:
AddOperator (ebnf)
```

```
ConstTerm := ConstFactor { MulOperator ConstFactor  }
           =:
ConstTerm (ebnf)
```

```
MulOperator := '*'  | '/'  | 'DIV'  | 'MOD'  |
               'REM'  | 'AND'  | '&'
             =:
MulOperator (ebnf)
```

```
ConstFactor := Number  | ConstString  |
               ConstSetOrQualidentOrFunction  |
               '(' ConstExpression ')'  |
               'NOT' ConstFactor  |
               ConstAttribute
             =:
ConstFactor (ebnf)
```

```
ConstString := string
             =:
ConstString (ebnf)
```

```
ComponentElement := ConstExpression [ '..' ConstExpression  ]
                  =:
ComponentElement (ebnf)
```

```
ComponentValue := ComponentElement [ 'BY' ConstExpression  ]
                =:
ComponentValue (ebnf)
```

```
ArraySetRecordValue := ComponentValue { ',' ComponentValue  }
                     =:
ArraySetRecordValue (ebnf)
```

```
Constructor := '{' [ ArraySetRecordValue  ] '}'
             =:
Constructor (ebnf)
```

```
ConstSetOrQualidentOrFunction := Constructor  |
                                 Qualident [ Constructor  |
                                             ConstActualParameters  ]
                               =:
ConstSetOrQualidentOrFunction (ebnf)
```

```
ConstActualParameters := '(' [ ExpList  ] ')'
                       =:
ConstActualParameters (ebnf)
```

```
ConstAttribute := '__ATTRIBUTE__' '__BUILTIN__' '('
                  '(' ConstAttributeExpression ')'
                  ')'
                =:
ConstAttribute (ebnf)
```

```
ConstAttributeExpression := Ident  | '<' Qualident
                            ',' Ident '>'
                          =:
ConstAttributeExpression (ebnf)
```

```
ByteAlignment := '<*' AttributeExpression '*>'
               =:
ByteAlignment (ebnf)
```

```
Alignment := [ ByteAlignment  ]
           =:
Alignment (ebnf)
```

```
TypeDeclaration := Ident '=' Type Alignment
                 =:
TypeDeclaration (ebnf)
```

```
Type := SimpleType  | ArrayType  | RecordType  |
        SetType  | PointerType  | ProcedureType
      =:
Type (ebnf)
```

```
SimpleType := Qualident [ SubrangeType  ]  |
              Enumeration  | SubrangeType
            =:
SimpleType (ebnf)
```

```
Enumeration := '(' IdentList ')'
             =:
Enumeration (ebnf)
```

```
IdentList := Ident { ',' Ident  }
           =:
IdentList (ebnf)
```

```
SubrangeType := '[' ConstExpression '..' ConstExpression
                ']'
              =:
SubrangeType (ebnf)
```

```
ArrayType := 'ARRAY' SimpleType { ',' SimpleType  }
             'OF' Type
           =:
ArrayType (ebnf)
```

```
RecordType := 'RECORD' [ DefaultRecordAttributes  ]
              FieldListSequence 'END'
            =:
RecordType (ebnf)
```

```
DefaultRecordAttributes := '<*' AttributeExpression
                           '*>'
                         =:
DefaultRecordAttributes (ebnf)
```

```
RecordFieldPragma := [ '<*' FieldPragmaExpression {
   ',' FieldPragmaExpression  } '*>'  ]
                  =:
RecordFieldPragma (ebnf)
```

```
FieldPragmaExpression := Ident [ '(' ConstExpression
                                 ')'  ]
                       =:
FieldPragmaExpression (ebnf)
```

```
AttributeExpression := Ident '(' ConstExpression ')'
                     =:
AttributeExpression (ebnf)
```

```
FieldListSequence := FieldListStatement { ';' FieldListStatement  }
                   =:
FieldListSequence (ebnf)
```

```
FieldListStatement := [ FieldList  ]
                    =:
FieldListStatement (ebnf)
```

```
FieldList := IdentList ':' Type RecordFieldPragma  |
             'CASE' CaseTag 'OF' Varient { '|' Varient  }
             [ 'ELSE' FieldListSequence  ] 'END'
           =:
FieldList (ebnf)
```

```
TagIdent := [ Ident  ]
          =:
TagIdent (ebnf)
```

```
CaseTag := TagIdent [ ':' Qualident  ]
         =:
CaseTag (ebnf)
```

```
Varient := [ VarientCaseLabelList ':' FieldListSequence  ]
         =:
Varient (ebnf)
```

```
VarientCaseLabelList := VarientCaseLabels { ',' VarientCaseLabels  }
                      =:
VarientCaseLabelList (ebnf)
```

```
VarientCaseLabels := ConstExpression [ '..' ConstExpression  ]
                  =:
VarientCaseLabels (ebnf)
```

```
CaseLabelList := CaseLabels { ',' CaseLabels  }
               =:
CaseLabelList (ebnf)
```

```
CaseLabels := ConstExpression [ '..' ConstExpression  ]
            =:
CaseLabels (ebnf)
```

```
SetType := ( 'SET'  | 'PACKEDSET'  ) 'OF' SimpleType
         =:
SetType (ebnf)
```

```
PointerType := 'POINTER' 'TO' Type
             =:
PointerType (ebnf)
```

```
ProcedureType := 'PROCEDURE' [ FormalTypeList  ]
               =:
ProcedureType (ebnf)
```

```
FormalTypeList := '(' ( ')' FormalReturn  |
                        ProcedureParameters ')' FormalReturn  )
                =:
FormalTypeList (ebnf)
```

```
FormalReturn := [ ':' OptReturnType  ]
              =:
FormalReturn (ebnf)
```

```
OptReturnType := '[' Qualident ']'  |
                 Qualident
               =:
OptReturnType (ebnf)
```

```
ProcedureParameters := ProcedureParameter { ',' ProcedureParameter  }
                     =:
ProcedureParameters (ebnf)
```

```
ProcedureParameter := '...'  | 'VAR' FormalType  |
                      FormalType
                    =:
ProcedureParameter (ebnf)
```

```
VarIdent := Ident [ '[' ConstExpression ']'  ]
          =:
VarIdent (ebnf)
```

```
VariableDeclaration := VarIdentList ':' Type Alignment
                     =:
VariableDeclaration (ebnf)
```

```
VarIdentList := VarIdent { ',' VarIdent  }
              =:
VarIdentList (ebnf)
```

```
Designator := Qualident { SubDesignator  }
            =:
Designator (ebnf)
```

```
SubDesignator := '.' Ident  | '[' ExpList ']'  |
                 '^'
                =:
SubDesignator (ebnf)
```

```
ExpList := Expression { ',' Expression  }
         =:
ExpList (ebnf)
```

```
Expression := SimpleExpression [ Relation SimpleExpression  ]
            =:
Expression (ebnf)
```

```
SimpleExpression := [ '+'  | '-'  ] Term { AddOperator
                                           Term  }
                  =:
SimpleExpression (ebnf)
```

```
Term := Factor { MulOperator Factor  }
      =:
Term (ebnf)
```

```
Factor := Number  | string  | SetOrDesignatorOrFunction  |
          '(' Expression ')'  |
          'NOT' Factor  | ConstAttribute
        =:
Factor (ebnf)
```

```
SetOrDesignatorOrFunction := ( Qualident [ Constructor  |
                                           SimpleDes
                                           [ ActualParameters  ]  ]  |
                               Constructor  )
```

(continues on next page)

(continued from previous page)

```
                            =:
SetOrDesignatorOrFunction (ebnf)
```

```
SimpleDes := { '.' Ident  | '[' ExpList ']'  |
               '^'  }
           =:
SimpleDes (ebnf)
```

```
ActualParameters := '(' [ ExpList  ] ')'
                  =:
ActualParameters (ebnf)
```

```
Statement := [ AssignmentOrProcedureCall  |
               IfStatement  | CaseStatement  |
               WhileStatement  | RepeatStatement  |
               LoopStatement  | ForStatement  |
               WithStatement  | AsmStatement  |
               'EXIT'  | 'RETURN' [ Expression  ]  |
               RetryStatement  ]
           =:
Statement (ebnf)
```

```
RetryStatement := 'RETRY'
                =:
RetryStatement (ebnf)
```

```
AssignmentOrProcedureCall := Designator ( ':=' Expression  |
                                          ActualParameters  |
                                           )
                           =:
AssignmentOrProcedureCall (ebnf)
```

```
StatementSequence := Statement { ';' Statement  }
                   =:
StatementSequence (ebnf)
```

```
IfStatement := 'IF' Expression 'THEN' StatementSequence
               { 'ELSIF' Expression 'THEN' StatementSequence  }
               [ 'ELSE' StatementSequence  ] 'END'
             =:
IfStatement (ebnf)
```

```
CaseStatement := 'CASE' Expression 'OF' Case { '|'
                                               Case  }
                 [ 'ELSE' StatementSequence  ] 'END'
               =:
CaseStatement (ebnf)
```

```
Case := [ CaseLabelList ':' StatementSequence  ]
      =:
Case (ebnf)
```

```
WhileStatement := 'WHILE' Expression 'DO' StatementSequence
                  'END'
                =:
WhileStatement (ebnf)
```

```
RepeatStatement := 'REPEAT' StatementSequence 'UNTIL'
                   Expression
                 =:
RepeatStatement (ebnf)
```

```
ForStatement := 'FOR' Ident ':=' Expression 'TO' Expression
                [ 'BY' ConstExpression  ] 'DO' StatementSequence
                'END'
              =:
ForStatement (ebnf)
```

```
LoopStatement := 'LOOP' StatementSequence 'END'
               =:
LoopStatement (ebnf)
```

```
WithStatement := 'WITH' Designator 'DO' StatementSequence
                 'END'
               =:
WithStatement (ebnf)
```

```
ProcedureDeclaration := ProcedureHeading ';' ( ProcedureBlock
                                               Ident
                                                )
                      =:
ProcedureDeclaration (ebnf)
```

```
DefineBuiltinProcedure := [ '__ATTRIBUTE__' '__BUILTIN__'
                            '(' '(' Ident ')' ')'  |
                            '__INLINE__'  ]
                        =:
DefineBuiltinProcedure (ebnf)
```

```
ProcedureHeading := 'PROCEDURE' DefineBuiltinProcedure
                    ( Ident [ FormalParameters  ] AttributeNoReturn  )
                  =:
ProcedureHeading (ebnf)
```

```
AttributeNoReturn := [ '<*' Ident '*>'  ]
                   =:
AttributeNoReturn (ebnf)
```

```
Builtin := [ '__BUILTIN__'  | '__INLINE__'  ]
         =:
Builtin (ebnf)
```

```
DefProcedureHeading := 'PROCEDURE' Builtin ( Ident
                                             [ DefFormalParameters  ]
                                             AttributeNoReturn  )

                     =:
DefProcedureHeading (ebnf)
```

```
ProcedureBlock := { Declaration  } [ 'BEGIN' BlockBody  ]
                  'END'
                =:
ProcedureBlock (ebnf)
```

```
Block := { Declaration  } InitialBlock FinalBlock
         'END'
       =:
Block (ebnf)
```

```
InitialBlock := [ 'BEGIN' BlockBody  ]
              =:
InitialBlock (ebnf)
```

```
FinalBlock := [ 'FINALLY' BlockBody  ]
            =:
FinalBlock (ebnf)
```

```
BlockBody := NormalPart [ 'EXCEPT' ExceptionalPart  ]
           =:
BlockBody (ebnf)
```

```
NormalPart := StatementSequence
            =:
NormalPart (ebnf)
```

```
ExceptionalPart := StatementSequence
                 =:
ExceptionalPart (ebnf)
```

```
Declaration := 'CONST' { ConstantDeclaration ';'  }  |
               'TYPE' { TypeDeclaration ';'  }  |
               'VAR' { VariableDeclaration ';'  }  |
               ProcedureDeclaration ';'  |
               ModuleDeclaration ';'
             =:
Declaration (ebnf)
```

```
DefFormalParameters := '(' [ DefMultiFPSection  ] ')'
                       FormalReturn
                     =:
DefFormalParameters (ebnf)
```

```
DefMultiFPSection := DefExtendedFP  |
                     FPSection [ ';' DefMultiFPSection  ]
                   =:
DefMultiFPSection (ebnf)
```

```
FormalParameters := '(' [ MultiFPSection  ] ')' FormalReturn
                  =:
FormalParameters (ebnf)
```

```
MultiFPSection := ExtendedFP  | FPSection [ ';' MultiFPSection  ]
                =:
MultiFPSection (ebnf)
```

```
FPSection := NonVarFPSection  | VarFPSection
           =:
FPSection (ebnf)
```

```
DefExtendedFP := DefOptArg  | '...'
               =:
DefExtendedFP (ebnf)
```

```
ExtendedFP := OptArg  | '...'
            =:
ExtendedFP (ebnf)
```

```
VarFPSection := 'VAR' IdentList ':' FormalType
              =:
VarFPSection (ebnf)
```

```
NonVarFPSection := IdentList ':' FormalType
                 =:
NonVarFPSection (ebnf)
```

```
OptArg := '[' Ident ':' FormalType [ '=' ConstExpression  ]
          ']'
        =:
OptArg (ebnf)
```

```
DefOptArg := '[' Ident ':' FormalType '=' ConstExpression
             ']'
           =:
DefOptArg (ebnf)
```

```
FormalType := { 'ARRAY' 'OF'  } Qualident
            =:
FormalType (ebnf)
```

```
ModuleDeclaration := 'MODULE' Ident [ Priority  ] ';'
                     { Import  } [ Export  ] Block
                     Ident
                   =:
ModuleDeclaration (ebnf)
```

```
Priority := '[' ConstExpression ']'
          =:
Priority (ebnf)
```

```
Export := 'EXPORT' ( 'QUALIFIED' IdentList  |
                     'UNQUALIFIED' IdentList  |
                     IdentList  ) ';'
        =:
Export (ebnf)
```

```
Import := 'FROM' Ident 'IMPORT' IdentList ';'  |
          'IMPORT' IdentList ';'
        =:
Import (ebnf)
```

```
DefinitionModule := 'DEFINITION' 'MODULE' [ 'FOR' string
                                          ] Ident
                    ';' { Import  } [ Export  ] {
   Definition  } 'END' Ident '.'
                  =:
DefinitionModule (ebnf)
```

```
Definition := 'CONST' { ConstantDeclaration ';'  }  |
              'TYPE' { Ident ( ';'  | '=' Type Alignment
                               ';'  )  }  |
              'VAR' { VariableDeclaration ';'  }  |
              DefProcedureHeading ';'
            =:
Definition (ebnf)
```

```
AsmStatement := 'ASM' [ 'VOLATILE'  ] '(' AsmOperands
                ')'
              =:
AsmStatement (ebnf)
```

```
NamedOperand := '[' Ident ']'
              =:
NamedOperand (ebnf)
```

```
AsmOperandName := [ NamedOperand  ]
                =:
AsmOperandName (ebnf)
```

```
AsmOperands := string [ ':' AsmList [ ':' AsmList [
   ':' TrashList  ]  ]  ]
             =:
AsmOperands (ebnf)
```

```
AsmList := [ AsmElement  ] { ',' AsmElement  }
         =:
AsmList (ebnf)
```

```
AsmElement := AsmOperandName string '(' Expression
              ')'
            =:
AsmElement (ebnf)
```

```
TrashList := [ string  ] { ',' string  }
           =:
TrashList (ebnf)
```

CHAPTER

EIGHTEEN

PIM AND ISO LIBRARY DEFINITIONS

This chapter contains M2F, PIM and ISO libraries.

18.1 Base libraries

These are the base libraries for the GNU Modula-2 compiler. These modules originally came from the M2F compiler and have been cleaned up and extended. They provide a basic interface to the underlying operating system via libc. They also include a number of libraries to allow access to compiler built-ins. Perhaps the largest difference to PIM and ISO libraries is the `DynamicString` module which declares the type `String`. The heavy use of this opaque data type results in a number of equivalent modules that can either handle `ARRAY OF CHAR` or `String`.

These modules have been extensively tested and are used throughout building the GNU Modula-2 compiler.

18.1.1 gm2-libs/ASCII

```
DEFINITION MODULE ASCII ;

EXPORT QUALIFIED
     nul, soh, stx, etx, eot, enq, ack, bel,
     bs , ht , nl , vt , np , cr , so , si ,
     dle, dc1, dc2, dc3, dc4, nak, syn, etb,
     can, em , sub, esc, fs , gs , rs , us ,
     sp ,  (* All the above are in order *)
     lf, ff, eof, del, tab, EOL ;

(*
   Note that lf, eof and EOL are added.
*)

CONST
nul (const)
soh (const)
stx (const)
etx (const)
```

(continues on next page)

(continued from previous page)

```
     nul=000C; soh=001C; stx=002C; etx=003C;
eot (const)
enq (const)
ack (const)
bel (const)
     eot=004C; enq=005C; ack=006C; bel=007C;
bs  (const)
ht  (const)
nl  (const)
vt  (const)
     bs =010C; ht =011C; nl =012C; vt =013C;
np  (const)
cr  (const)
so  (const)
si  (const)
     np =014C; cr =015C; so =016C; si =017C;
dle (const)
dc1 (const)
dc2 (const)
dc3 (const)
     dle=020C; dc1=021C; dc2=022C; dc3=023C;
dc4 (const)
nak (const)
syn (const)
etb (const)
     dc4=024C; nak=025C; syn=026C; etb=027C;
can (const)
em  (const)
sub (const)
esc (const)
     can=030C; em =031C; sub=032C; esc=033C;
fs  (const)
gs  (const)
rs  (const)
us  (const)
     fs =034C; gs =035C; rs =036C; us =037C;
sp  (const)
     sp =040C; (* All the above are in order *)
lf  (const)
ff  (const)
eof (const)
tab (const)
     lf =nl  ; ff =np  ; eof=eot ; tab=ht  ;
del (const)
EOL (const)
     del=177C; EOL=nl  ;

END ASCII.
```

18.1.2 gm2-libs/Args

```
DEFINITION MODULE Args ;

EXPORT QUALIFIED GetArg, Narg ;

(*
   GetArg - returns the nth argument from the command line.
            The success of the operation is returned.
*)

GetArg
PROCEDURE GetArg (VAR a: ARRAY OF CHAR; n: CARDINAL) : BOOLEAN ;

(*
   Narg - returns the number of arguments available from
          command line.
*)

Narg
PROCEDURE Narg () : CARDINAL ;

END Args.
```

18.1.3 gm2-libs/Assertion

```
DEFINITION MODULE Assertion ;

EXPORT QUALIFIED Assert ;

(*
   Assert - tests the boolean Condition, if it fails then HALT
            is called.
*)

Assert
PROCEDURE Assert (Condition: BOOLEAN) ;

END Assertion.
```

18.1.4 gm2-libs/Break

```
DEFINITION MODULE Break ;

END Break.
```

18.1.5 gm2-libs/Builtins

```
DEFINITION MODULE Builtins ;

FROM SYSTEM IMPORT ADDRESS ;

(* floating point intrinsic procedure functions *)

isfinitef
PROCEDURE __BUILTIN__ isfinitef (x: SHORTREAL) : BOOLEAN ;
isfinite
PROCEDURE __BUILTIN__ isfinite (x: REAL) : BOOLEAN ;
isfinitel
PROCEDURE __BUILTIN__ isfinitel (x: LONGREAL) : BOOLEAN ;

sinf
PROCEDURE __BUILTIN__ sinf (x: SHORTREAL) : SHORTREAL ;
sin
PROCEDURE __BUILTIN__ sin (x: REAL) : REAL ;
sinl
PROCEDURE __BUILTIN__ sinl (x: LONGREAL) : LONGREAL ;

cosf
PROCEDURE __BUILTIN__ cosf (x: SHORTREAL) : SHORTREAL ;
cos
PROCEDURE __BUILTIN__ cos (x: REAL) : REAL ;
cosl
PROCEDURE __BUILTIN__ cosl (x: LONGREAL) : LONGREAL ;

sqrtf
PROCEDURE __BUILTIN__ sqrtf (x: SHORTREAL) : SHORTREAL ;
sqrt
PROCEDURE __BUILTIN__ sqrt (x: REAL) : REAL ;
sqrtl
PROCEDURE __BUILTIN__ sqrtl (x: LONGREAL) : LONGREAL ;

atan2f
PROCEDURE __BUILTIN__ atan2f (x, y: SHORTREAL) : SHORTREAL ;
atan2
PROCEDURE __BUILTIN__ atan2 (x, y: REAL) : REAL ;
atan2l
PROCEDURE __BUILTIN__ atan2l (x, y: LONGREAL) : LONGREAL ;

fabsf
PROCEDURE __BUILTIN__ fabsf (x: SHORTREAL) : SHORTREAL ;
fabs
PROCEDURE __BUILTIN__ fabs (x: REAL) : REAL ;
fabsl
PROCEDURE __BUILTIN__ fabsl (x: LONGREAL) : LONGREAL ;

logf
PROCEDURE __BUILTIN__ logf (x: SHORTREAL) : SHORTREAL ;
```

(continues on next page)

(continued from previous page)

```
log
PROCEDURE __BUILTIN__ log (x: REAL) : REAL ;
logl
PROCEDURE __BUILTIN__ logl (x: LONGREAL) : LONGREAL ;

expf
PROCEDURE __BUILTIN__ expf (x: SHORTREAL) : SHORTREAL ;
exp
PROCEDURE __BUILTIN__ exp (x: REAL) : REAL ;
expl
PROCEDURE __BUILTIN__ expl (x: LONGREAL) : LONGREAL ;

log10f
PROCEDURE __BUILTIN__ log10f (x: SHORTREAL) : SHORTREAL ;
log10
PROCEDURE __BUILTIN__ log10 (x: REAL) : REAL ;
log10l
PROCEDURE __BUILTIN__ log10l (x: LONGREAL) : LONGREAL ;

exp10f
PROCEDURE __BUILTIN__ exp10f (x: SHORTREAL) : SHORTREAL ;
exp10
PROCEDURE __BUILTIN__ exp10 (x: REAL) : REAL ;
exp10l
PROCEDURE __BUILTIN__ exp10l (x: LONGREAL) : LONGREAL ;

ilogbf
PROCEDURE __BUILTIN__ ilogbf (x: SHORTREAL) : INTEGER ;
ilogb
PROCEDURE __BUILTIN__ ilogb (x: REAL) : INTEGER ;
ilogbl
PROCEDURE __BUILTIN__ ilogbl (x: LONGREAL) : INTEGER ;

huge_val
PROCEDURE __BUILTIN__ huge_val () : REAL ;
huge_valf
PROCEDURE __BUILTIN__ huge_valf () : SHORTREAL ;
huge_vall
PROCEDURE __BUILTIN__ huge_vall () : LONGREAL ;

significand
PROCEDURE __BUILTIN__ significand (r: REAL) : REAL ;
significandf
PROCEDURE __BUILTIN__ significandf (s: SHORTREAL) : SHORTREAL ;
significandl
PROCEDURE __BUILTIN__ significandl (l: LONGREAL) : LONGREAL ;

modf
PROCEDURE __BUILTIN__ modf (x: REAL; VAR y: REAL) : REAL ;
modff
PROCEDURE __BUILTIN__ modff (x: SHORTREAL;
```

(continues on next page)

(continued from previous page)

```
                            VAR y: SHORTREAL) : SHORTREAL ;
modfl
PROCEDURE __BUILTIN__ modfl (x: LONGREAL; VAR y: LONGREAL) : LONGREAL ;

signbit
PROCEDURE __BUILTIN__ signbit (r: REAL) : INTEGER ;
signbitf
PROCEDURE __BUILTIN__ signbitf (s: SHORTREAL) : INTEGER ;
signbitl
PROCEDURE __BUILTIN__ signbitl (l: LONGREAL) : INTEGER ;

nextafter
PROCEDURE __BUILTIN__ nextafter (x, y: REAL) : REAL ;
nextafterf
PROCEDURE __BUILTIN__ nextafterf (x, y: SHORTREAL) : SHORTREAL ;
nextafterl
PROCEDURE __BUILTIN__ nextafterl (x, y: LONGREAL) : LONGREAL ;

nexttoward
PROCEDURE __BUILTIN__ nexttoward (x, y: REAL) : LONGREAL ;
nexttowardf
PROCEDURE __BUILTIN__ nexttowardf (x, y: SHORTREAL) : LONGREAL ;
nexttowardl
PROCEDURE __BUILTIN__ nexttowardl (x, y: LONGREAL) : LONGREAL ;

scalb
PROCEDURE __BUILTIN__ scalb (x, n: REAL) : REAL ;
scalbf
PROCEDURE __BUILTIN__ scalbf (x, n: SHORTREAL) : SHORTREAL ;
scalbl
PROCEDURE __BUILTIN__ scalbl (x, n: LONGREAL) : LONGREAL ;

scalbln
PROCEDURE __BUILTIN__ scalbln (x: REAL; n: LONGINT) : REAL ;
scalblnf
PROCEDURE __BUILTIN__ scalblnf (x: SHORTREAL; n: LONGINT) : SHORTREAL ;
scalblnl
PROCEDURE __BUILTIN__ scalblnl (x: LONGREAL; n: LONGINT) : LONGREAL ;

scalbn
PROCEDURE __BUILTIN__ scalbn (x: REAL; n: INTEGER) : REAL ;
scalbnf
PROCEDURE __BUILTIN__ scalbnf (x: SHORTREAL; n: INTEGER) : SHORTREAL ;
scalbnl
PROCEDURE __BUILTIN__ scalbnl (x: LONGREAL; n: INTEGER) : LONGREAL ;

(* complex arithmetic intrincic procedure functions *)

cabsf
PROCEDURE __BUILTIN__ cabsf (z: SHORTCOMPLEX) : SHORTREAL ;
cabs
```

(continues on next page)

(continued from previous page)

```
PROCEDURE __BUILTIN__ cabs (z: COMPLEX) : REAL ;
cabsl
PROCEDURE __BUILTIN__ cabsl (z: LONGCOMPLEX) : LONGREAL ;

cargf
PROCEDURE __BUILTIN__ cargf (z: SHORTCOMPLEX) : SHORTREAL ;
carg
PROCEDURE __BUILTIN__ carg (z: COMPLEX) : REAL ;
cargl
PROCEDURE __BUILTIN__ cargl (z: LONGCOMPLEX) : LONGREAL ;

conjf
PROCEDURE __BUILTIN__ conjf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
conj
PROCEDURE __BUILTIN__ conj (z: COMPLEX) : COMPLEX ;
conjl
PROCEDURE __BUILTIN__ conjl (z: LONGCOMPLEX) : LONGCOMPLEX ;

cpowerf
PROCEDURE __BUILTIN__ cpowerf (base: SHORTCOMPLEX;
                               exp: SHORTREAL) : SHORTCOMPLEX ;
cpower
PROCEDURE __BUILTIN__ cpower (base: COMPLEX; exp: REAL) : COMPLEX ;
cpowerl
PROCEDURE __BUILTIN__ cpowerl (base: LONGCOMPLEX;
                               exp: LONGREAL) : LONGCOMPLEX ;

csqrtf
PROCEDURE __BUILTIN__ csqrtf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
csqrt
PROCEDURE __BUILTIN__ csqrt (z: COMPLEX) : COMPLEX ;
csqrtl
PROCEDURE __BUILTIN__ csqrtl (z: LONGCOMPLEX) : LONGCOMPLEX ;

cexpf
PROCEDURE __BUILTIN__ cexpf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
cexp
PROCEDURE __BUILTIN__ cexp (z: COMPLEX) : COMPLEX ;
cexpl
PROCEDURE __BUILTIN__ cexpl (z: LONGCOMPLEX) : LONGCOMPLEX ;

clnf
PROCEDURE __BUILTIN__ clnf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
cln
PROCEDURE __BUILTIN__ cln (z: COMPLEX) : COMPLEX ;
clnl
PROCEDURE __BUILTIN__ clnl (z: LONGCOMPLEX) : LONGCOMPLEX ;

csinf
PROCEDURE __BUILTIN__ csinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
csin
```

(continues on next page)

(continued from previous page)

```
PROCEDURE __BUILTIN__ csin (z: COMPLEX) : COMPLEX ;
csinl
PROCEDURE __BUILTIN__ csinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

ccosf
PROCEDURE __BUILTIN__ ccosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
ccos
PROCEDURE __BUILTIN__ ccos (z: COMPLEX) : COMPLEX ;
ccosl
PROCEDURE __BUILTIN__ ccosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

ctanf
PROCEDURE __BUILTIN__ ctanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
ctan
PROCEDURE __BUILTIN__ ctan (z: COMPLEX) : COMPLEX ;
ctanl
PROCEDURE __BUILTIN__ ctanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

carcsinf
PROCEDURE __BUILTIN__ carcsinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
carcsin
PROCEDURE __BUILTIN__ carcsin (z: COMPLEX) : COMPLEX ;
carcsinl
PROCEDURE __BUILTIN__ carcsinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

carccosf
PROCEDURE __BUILTIN__ carccosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
carccos
PROCEDURE __BUILTIN__ carccos (z: COMPLEX) : COMPLEX ;
carccosl
PROCEDURE __BUILTIN__ carccosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

carctanf
PROCEDURE __BUILTIN__ carctanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
carctan
PROCEDURE __BUILTIN__ carctan (z: COMPLEX) : COMPLEX ;
carctanl
PROCEDURE __BUILTIN__ carctanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

(* memory and string intrincic procedure functions *)

alloca
PROCEDURE __BUILTIN__ alloca (i: CARDINAL) : ADDRESS ;
memcpy
PROCEDURE __BUILTIN__ memcpy (dest, src: ADDRESS;
                              nbytes: CARDINAL) : ADDRESS ;
index
PROCEDURE __BUILTIN__ index (s: ADDRESS; c: INTEGER) : ADDRESS ;
rindex
PROCEDURE __BUILTIN__ rindex (s: ADDRESS; c: INTEGER) : ADDRESS ;
memcmp
```

(continues on next page)

(continued from previous page)

```
PROCEDURE __BUILTIN__ memcmp (s1, s2: ADDRESS;
                              nbytes: CARDINAL) : INTEGER ;
memset
PROCEDURE __BUILTIN__ memset (s: ADDRESS; c: INTEGER;
                              nbytes: CARDINAL) : ADDRESS ;
memmove
PROCEDURE __BUILTIN__ memmove (s1, s2: ADDRESS;
                               nbytes: CARDINAL) : ADDRESS ;
strcat
PROCEDURE __BUILTIN__ strcat (dest, src: ADDRESS) : ADDRESS ;
strncat
PROCEDURE __BUILTIN__ strncat (dest, src: ADDRESS;
                               nbytes: CARDINAL) : ADDRESS ;
strcpy
PROCEDURE __BUILTIN__ strcpy (dest, src: ADDRESS) : ADDRESS ;
strncpy
PROCEDURE __BUILTIN__ strncpy (dest, src: ADDRESS;
                               nbytes: CARDINAL) : ADDRESS ;
strcmp
PROCEDURE __BUILTIN__ strcmp (s1, s2: ADDRESS) : INTEGER ;
strncmp
PROCEDURE __BUILTIN__ strncmp (s1, s2: ADDRESS;
                               nbytes: CARDINAL) : INTEGER ;
strlen
PROCEDURE __BUILTIN__ strlen (s: ADDRESS) : INTEGER ;
strstr
PROCEDURE __BUILTIN__ strstr (haystack, needle: ADDRESS) : ADDRESS ;
strpbrk
PROCEDURE __BUILTIN__ strpbrk (s, accept: ADDRESS) : ADDRESS ;
strspn
PROCEDURE __BUILTIN__ strspn (s, accept: ADDRESS) : CARDINAL ;
strcspn
PROCEDURE __BUILTIN__ strcspn (s, accept: ADDRESS) : CARDINAL ;
strchr
PROCEDURE __BUILTIN__ strchr (s: ADDRESS; c: INTEGER) : ADDRESS ;
strrchr
PROCEDURE __BUILTIN__ strrchr (s: ADDRESS; c: INTEGER) : ADDRESS ;

(*
   longjmp - this GCC builtin restricts the val to always 1.
*)
(* do not use these two builtins, as gcc, only really
   anticipates that the Ada front end should use them
   and it only uses them in its runtime exception handling.
   We leave them here in the hope that someday they will
   behave more like their libc counterparts.  *)

longjmp
PROCEDURE __BUILTIN__ longjmp (env: ADDRESS; val: INTEGER) ;
setjmp
PROCEDURE __BUILTIN__ setjmp (env: ADDRESS) : INTEGER ;
```

(continues on next page)

(continued from previous page)

```
(*
   frame_address - returns the address of the frame.
                   The current frame is obtained if level is 0,
                   the next level up if level is 1 etc.
*)

frame_address
PROCEDURE __BUILTIN__ frame_address (level: CARDINAL) : ADDRESS ;

(*
   return_address - returns the return address of function.
                    The current function return address is
                    obtained if level is 0,
                    the next level up if level is 1 etc.
*)

return_address
PROCEDURE __BUILTIN__ return_address (level: CARDINAL) : ADDRESS ;

(*
   alloca_trace - this is a no-op which is used for internal debugging.
*)

alloca_trace
PROCEDURE alloca_trace (returned: ADDRESS; nBytes: CARDINAL) : ADDRESS ;

END Builtins.
```

18.1.6 gm2-libs/COROUTINES

```
DEFINITION MODULE FOR "C" COROUTINES ;

CONST
   UnassignedPriority = 0 ;

TYPE
INTERRUPTSOURCE (type)
   INTERRUPTSOURCE = CARDINAL ;
PROTECTION (type)
   PROTECTION = [UnassignedPriority..7] ;

END COROUTINES.
```

18.1.7 gm2-libs/CmdArgs

```
DEFINITION MODULE CmdArgs ;

EXPORT QUALIFIED GetArg, Narg ;

(*
   GetArg - returns the nth argument from the command line, CmdLine
            the success of the operation is returned.
*)

GetArg
PROCEDURE GetArg (CmdLine: ARRAY OF CHAR;
                  n: CARDINAL; VAR Argi: ARRAY OF CHAR) : BOOLEAN ;

(*
   Narg - returns the number of arguments available from
          command line, CmdLine.
*)

Narg
PROCEDURE Narg (CmdLine: ARRAY OF CHAR) : CARDINAL ;

END CmdArgs.
```

18.1.8 gm2-libs/Debug

```
DEFINITION MODULE Debug ;

(*
    Description: provides some simple debugging routines.
*)

EXPORT QUALIFIED Halt, DebugString ;

(*
   Halt - writes a message in the format:
          Module:Line:Message

          It then terminates by calling HALT.
*)

Halt
PROCEDURE Halt (Message: ARRAY OF CHAR;
                LineNo: CARDINAL;
                Module: ARRAY OF CHAR) ;

(*
   DebugString - writes a string to the debugging device (Scn.Write).
                 It interprets \n as carriage return, linefeed.
```

(continues on next page)

(continued from previous page)

```
*)

DebugString
PROCEDURE DebugString (a: ARRAY OF CHAR) ;

END Debug.
```

18.1.9 gm2-libs/DynamicStrings

```
DEFINITION MODULE DynamicStrings ;

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED String,
                 InitString, KillString, Fin, InitStringCharStar,
                 InitStringChar, Index, RIndex,
                 Mark, Length, ConCat, ConCatChar, Assign, Dup, Add,
                 Equal, EqualCharStar, EqualArray, ToUpper, ToLower,
                 CopyOut, Mult, Slice,
                 RemoveWhitePrefix, RemoveWhitePostfix, RemoveComment,
                 char, string,
                 InitStringDB, InitStringCharStarDB, InitStringCharDB,
                 MultDB, DupDB, SliceDB,
                 PushAllocation, PopAllocation, PopAllocationExemption ;

TYPE
String (type)
   String ;

(*
   InitString - creates and returns a String type object.
                Initial contents are, a.
*)

InitString
PROCEDURE InitString (a: ARRAY OF CHAR) : String ;

(*
   KillString - frees String, s, and its contents.
                NIL is returned.
*)

KillString
PROCEDURE KillString (s: String) : String ;

(*
   Fin - finishes with a string, it calls KillString with, s.
         The purpose of the procedure is to provide a short cut
         to calling KillString and then testing the return result.
*)
```

(continues on next page)

(continued from previous page)

```
Fin
PROCEDURE Fin (s: String) ;

(*
   InitStringCharStar - initializes and returns a String to contain
                        the C string.
*)

InitStringCharStar
PROCEDURE InitStringCharStar (a: ADDRESS) : String ;

(*
   InitStringChar - initializes and returns a String to contain the
                    single character, ch.
*)

InitStringChar
PROCEDURE InitStringChar (ch: CHAR) : String ;

(*
   Mark - marks String, s, ready for garbage collection.
*)

Mark
PROCEDURE Mark (s: String) : String ;

(*
   Length - returns the length of the String, s.
*)

Length
PROCEDURE Length (s: String) : CARDINAL ;

(*
   ConCat - returns String, a, after the contents of, b,
            have been appended.
*)

ConCat
PROCEDURE ConCat (a, b: String) : String ;

(*
   ConCatChar - returns String, a, after character, ch,
                has been appended.
*)

ConCatChar
PROCEDURE ConCatChar (a: String; ch: CHAR) : String ;

(*
   Assign - assigns the contents of, b, into, a.
```

(continues on next page)

(continued from previous page)

```
            String, a, is returned.
*)

Assign
PROCEDURE Assign (a, b: String) : String ;

(*
   Dup - duplicate a String, s, returning the copy of s.
*)

Dup
PROCEDURE Dup (s: String) : String ;

(*
   Add - returns a new String which contains the contents of a and b.
*)

Add
PROCEDURE Add (a, b: String) : String ;

(*
   Equal - returns TRUE if String, a, and, b, are equal.
*)

Equal
PROCEDURE Equal (a, b: String) : BOOLEAN ;

(*
   EqualCharStar - returns TRUE if contents of String, s, is
                   the same as the string, a.
*)

EqualCharStar
PROCEDURE EqualCharStar (s: String; a: ADDRESS) : BOOLEAN ;

(*
   EqualArray - returns TRUE if contents of String, s, is the
                same as the string, a.
*)

EqualArray
PROCEDURE EqualArray (s: String; a: ARRAY OF CHAR) : BOOLEAN ;

(*
   Mult - returns a new string which is n concatenations of String, s.
          If n<=0 then an empty string is returned.
*)

Mult
PROCEDURE Mult (s: String; n: CARDINAL) : String ;
```

(continues on next page)

(continued from previous page)

```
(*
   Slice - returns a new string which contains the elements
           low..high-1

           strings start at element 0
           Slice(s, 0, 2)  will return elements 0, 1 but not 2
           Slice(s, 1, 3)  will return elements 1, 2 but not 3
           Slice(s, 2, 0)  will return elements 2..max
           Slice(s, 3, -1) will return elements 3..max-1
           Slice(s, 4, -2) will return elements 4..max-2
*)

Slice
PROCEDURE Slice (s: String; low, high: INTEGER) : String ;

(*
   Index - returns the indice of the first occurance of, ch, in
           String, s. -1 is returned if, ch, does not exist.
           The search starts at position, o.
*)

Index
PROCEDURE Index (s: String; ch: CHAR; o: CARDINAL) : INTEGER ;

(*
   RIndex - returns the indice of the last occurance of, ch,
            in String, s. The search starts at position, o.
            -1 is returned if, ch, is not found.
*)

RIndex
PROCEDURE RIndex (s: String; ch: CHAR; o: CARDINAL) : INTEGER ;

(*
   RemoveComment - assuming that, comment, is a comment delimiter
                   which indicates anything to its right is a comment
                   then strip off the comment and also any white space
                   on the remaining right hand side.
                   It leaves any white space on the left hand side
                   alone.
*)

RemoveComment
PROCEDURE RemoveComment (s: String; comment: CHAR) : String ;

(*
   RemoveWhitePrefix - removes any leading white space from String, s.
                       A new string is returned.
*)

RemoveWhitePrefix
```

(continues on next page)

(continued from previous page)

```
PROCEDURE RemoveWhitePrefix (s: String) : String ;

(*
   RemoveWhitePostfix - removes any leading white space from String, s.
                        A new string is returned.
*)

RemoveWhitePostfix
PROCEDURE RemoveWhitePostfix (s: String) : String ;

(*
   ToUpper - returns string, s, after it has had its lower case
             characters replaced by upper case characters.
             The string, s, is not duplicated.
*)

ToUpper
PROCEDURE ToUpper (s: String) : String ;

(*
   ToLower - returns string, s, after it has had its upper case
             characters replaced by lower case characters.
             The string, s, is not duplicated.
*)

ToLower
PROCEDURE ToLower (s: String) : String ;

(*
   CopyOut - copies string, s, to a.
*)

CopyOut
PROCEDURE CopyOut (VAR a: ARRAY OF CHAR; s: String) ;

(*
   char - returns the character, ch, at position, i, in String, s.
          As Slice the index can be negative so:

          char(s, 0) will return the first character
          char(s, 1) will return the second character
          char(s, -1) will return the last character
          char(s, -2) will return the penultimate character

          a nul character is returned if the index is out of range.
*)

char
PROCEDURE char (s: String; i: INTEGER) : CHAR ;

(*
```

(continues on next page)

(continued from previous page)

```
   string - returns the C style char * of String, s.
*)

string
PROCEDURE string (s: String) : ADDRESS ;

(*
   to easily debug an application using this library one could use
   use the following macro processing defines:

   #define InitString(X) InitStringDB(X, __FILE__, __LINE__)
   #define InitStringCharStar(X) InitStringCharStarDB(X, \
     __FILE__, __LINE__)
   #define InitStringChar(X) InitStringCharDB(X, __FILE__, __LINE__)
   #define Mult(X,Y) MultDB(X, Y, __FILE__, __LINE__)
   #define Dup(X) DupDB(X, __FILE__, __LINE__)
   #define Slice(X,Y,Z) SliceDB(X, Y, Z, __FILE__, __LINE__)

   and then invoke gm2 with the -fcpp flag.
*)

(*
   InitStringDB - the debug version of InitString.
*)

InitStringDB
PROCEDURE InitStringDB (a: ARRAY OF CHAR;
                        file: ARRAY OF CHAR; line: CARDINAL) : String ;

(*
   InitStringCharStarDB - the debug version of InitStringCharStar.
*)

InitStringCharStarDB
PROCEDURE InitStringCharStarDB (a: ADDRESS;
                                file: ARRAY OF CHAR;
                                line: CARDINAL) : String ;

(*
   InitStringCharDB - the debug version of InitStringChar.
*)

InitStringCharDB
PROCEDURE InitStringCharDB (ch: CHAR;
                            file: ARRAY OF CHAR;
                            line: CARDINAL) : String ;

(*
   MultDB - the debug version of MultDB.
*)
```

(continues on next page)

(continued from previous page)

```
MultDB
PROCEDURE MultDB (s: String; n: CARDINAL;
                  file: ARRAY OF CHAR; line: CARDINAL) : String ;

(*
   DupDB - the debug version of Dup.
*)

DupDB
PROCEDURE DupDB (s: String;
                 file: ARRAY OF CHAR; line: CARDINAL) : String ;

(*
   SliceDB - debug version of Slice.
*)

SliceDB
PROCEDURE SliceDB (s: String; low, high: INTEGER;
                   file: ARRAY OF CHAR; line: CARDINAL) : String ;

(*
   PushAllocation - pushes the current allocation/deallocation lists.
*)

PushAllocation
PROCEDURE PushAllocation ;

(*
   PopAllocation - test to see that all strings are deallocated since
                   the last push.  Then it pops to the previous
                   allocation/deallocation lists.

                   If halt is true then the application terminates
                   with an exit code of 1.
*)

PopAllocation
PROCEDURE PopAllocation (halt: BOOLEAN) ;

(*
   PopAllocationExemption - test to see that all strings are
                            deallocated, except string, e, since
                            the last push.
                            Then it pops to the previous
                            allocation/deallocation lists.

                            If halt is true then the application
                            terminates with an exit code of 1.

                            The string, e, is returned unmodified,
*)
```

(continues on next page)

(continued from previous page)

```
PopAllocationExemption
PROCEDURE PopAllocationExemption (halt: BOOLEAN; e: String) : String ;

END DynamicStrings.
```

18.1.10 gm2-libs/Environment

```
DEFINITION MODULE Environment ;

EXPORT QUALIFIED GetEnvironment, PutEnvironment ;

(*
   GetEnvironment - gets the environment variable Env and places
                         a copy of its value into string, dest.
                    It returns TRUE if the string Env was found in
                    the processes environment.
*)

GetEnvironment
PROCEDURE GetEnvironment (Env: ARRAY OF CHAR;
                          VAR dest: ARRAY OF CHAR) : BOOLEAN ;

(*
   PutEnvironment - change or add an environment variable definition
                    EnvDef.
                    TRUE is returned if the environment variable was
                    set or changed successfully.
*)

PutEnvironment
PROCEDURE PutEnvironment (EnvDef: ARRAY OF CHAR) : BOOLEAN ;

END Environment.
```

18.1.11 gm2-libs/FIO

```
DEFINITION MODULE FIO ;

(* Provides a simple buffered file input/output library.  *)

FROM SYSTEM IMPORT ADDRESS, BYTE ;

EXPORT QUALIFIED (* types *)
                 File,
                 (* procedures *)
                 OpenToRead, OpenToWrite, OpenForRandom, Close,
```

(continues on next page)

(continued from previous page)

```
                EOF, EOLN, WasEOLN, IsNoError, Exists, IsActive,
                exists, openToRead, openToWrite, openForRandom,
                SetPositionFromBeginning,
                SetPositionFromEnd,
                FindPosition,
                ReadChar, ReadString,
                WriteChar, WriteString, WriteLine,
                WriteCardinal, ReadCardinal,
                UnReadChar,
                WriteNBytes, ReadNBytes,
                FlushBuffer,
                GetUnixFileDescriptor,
                GetFileName, getFileName, getFileNameLength,
                FlushOutErr,
                (* variables *)
                StdIn, StdOut, StdErr ;

TYPE
File (type)
   File = CARDINAL ;

(* the following variables are initialized to their UNIX equivalents *)
VAR
StdIn (var)
StdOut (var)
StdErr (var)
   StdIn, StdOut, StdErr: File ;

(*
   IsNoError - returns a TRUE if no error has occured on file, f.
*)

IsNoError
PROCEDURE IsNoError (f: File) : BOOLEAN ;

(*
   IsActive - returns TRUE if the file, f, is still active.
*)

IsActive
PROCEDURE IsActive (f: File) : BOOLEAN ;

(*
   Exists - returns TRUE if a file named, fname exists for reading.
*)

Exists
PROCEDURE Exists (fname: ARRAY OF CHAR) : BOOLEAN ;

(*
   OpenToRead - attempts to open a file, fname, for reading and
```

(continues on next page)

(continued from previous page)

```
                it returns this file.
                The success of this operation can be checked by
                calling IsNoError.
*)

OpenToRead
PROCEDURE OpenToRead (fname: ARRAY OF CHAR) : File ;

(*
   OpenToWrite - attempts to open a file, fname, for write and
                 it returns this file.
                 The success of this operation can be checked by
                 calling IsNoError.
*)

OpenToWrite
PROCEDURE OpenToWrite (fname: ARRAY OF CHAR) : File ;

(*
   OpenForRandom - attempts to open a file, fname, for random access
                   read or write and it returns this file.
                   The success of this operation can be checked by
                   calling IsNoError.
                   towrite, determines whether the file should be
                   opened for writing or reading.
                   newfile, determines whether a file should be
                   created if towrite is TRUE or whether the
                   previous file should be left alone,
                   allowing this descriptor to seek
                   and modify an existing file.
*)

OpenForRandom
PROCEDURE OpenForRandom (fname: ARRAY OF CHAR;
                         towrite, newfile: BOOLEAN) : File ;

(*
   Close - close a file which has been previously opened using:
           OpenToRead, OpenToWrite, OpenForRandom.
           It is correct to close a file which has an error status.
*)

Close
PROCEDURE Close (f: File) ;

(* the following functions are functionally equivalent to the above
   except they allow C style names.
*)

exists
PROCEDURE exists        (fname: ADDRESS; flength: CARDINAL) : BOOLEAN ;
```

(continues on next page)

(continued from previous page)

```
openToRead
PROCEDURE openToRead    (fname: ADDRESS; flength: CARDINAL) : File ;
openToWrite
PROCEDURE openToWrite   (fname: ADDRESS; flength: CARDINAL) : File ;
openForRandom
PROCEDURE openForRandom (fname: ADDRESS; flength: CARDINAL;
                         towrite, newfile: BOOLEAN) : File ;

(*
   FlushBuffer - flush contents of the FIO file, f, to libc.
*)

FlushBuffer
PROCEDURE FlushBuffer (f: File) ;

(*
   ReadNBytes - reads nBytes of a file into memory area, dest, returning
                the number of bytes actually read.
                This function will consume from the buffer and then
                perform direct libc reads. It is ideal for large reads.
*)

ReadNBytes
PROCEDURE ReadNBytes (f: File; nBytes: CARDINAL;
                      dest: ADDRESS) : CARDINAL ;

(*
   ReadAny - reads HIGH(a) bytes into, a. All input
             is fully buffered, unlike ReadNBytes and thus is more
             suited to small reads.
*)

ReadAny
PROCEDURE ReadAny (f: File; VAR a: ARRAY OF BYTE) ;

(*
   WriteNBytes - writes nBytes from memory area src to a file
                 returning the number of bytes actually written.
                 This function will flush the buffer and then
                 write the nBytes using a direct write from libc.
                 It is ideal for large writes.
*)

WriteNBytes
PROCEDURE WriteNBytes (f: File; nBytes: CARDINAL;
                       src: ADDRESS) : CARDINAL ;

(*
   WriteAny - writes HIGH(a) bytes onto, file, f. All output
              is fully buffered, unlike WriteNBytes and thus is more
              suited to small writes.
```

(continues on next page)

(continued from previous page)

```
*)

WriteAny
PROCEDURE WriteAny (f: File; VAR a: ARRAY OF BYTE) ;

(*
   WriteChar - writes a single character to file, f.
*)

WriteChar
PROCEDURE WriteChar (f: File; ch: CHAR) ;

(*
   EOF - tests to see whether a file, f, has reached end of file.
*)

EOF
PROCEDURE EOF (f: File) : BOOLEAN ;

(*
   EOLN - tests to see whether a file, f, is about to read a newline.
          It does NOT consume the newline.  It reads the next character
          and then immediately unreads the character.
*)

EOLN
PROCEDURE EOLN (f: File) : BOOLEAN ;

(*
   WasEOLN - tests to see whether a file, f, has just read a newline
             character.
*)

WasEOLN
PROCEDURE WasEOLN (f: File) : BOOLEAN ;

(*
   ReadChar - returns a character read from file, f.
              Sensible to check with IsNoError or EOF after calling
              this function.
*)

ReadChar
PROCEDURE ReadChar (f: File) : CHAR ;

(*
   UnReadChar - replaces a character, ch, back into file, f.
                This character must have been read by ReadChar
                and it does not allow successive calls.  It may
                only be called if the previous read was successful,
                end of file or end of line seen.
```

(continues on next page)

(continued from previous page)

```
*)

UnReadChar
PROCEDURE UnReadChar (f: File ; ch: CHAR) ;

(*
   WriteLine - writes out a linefeed to file, f.
*)

WriteLine
PROCEDURE WriteLine (f: File) ;

(*
   WriteString - writes a string to file, f.
*)

WriteString
PROCEDURE WriteString (f: File; a: ARRAY OF CHAR) ;

(*
   ReadString - reads a string from file, f, into string, a.
                It terminates the string if HIGH is reached or
                if a newline is seen or an error occurs.
*)

ReadString
PROCEDURE ReadString (f: File; VAR a: ARRAY OF CHAR) ;

(*
   WriteCardinal - writes a CARDINAL to file, f.
                   It writes the binary image of the CARDINAL.
                   to file, f.
*)

WriteCardinal
PROCEDURE WriteCardinal (f: File; c: CARDINAL) ;

(*
   ReadCardinal - reads a CARDINAL from file, f.
                  It reads a bit image of a CARDINAL
                  from file, f.
*)

ReadCardinal
PROCEDURE ReadCardinal (f: File) : CARDINAL ;

(*
   GetUnixFileDescriptor - returns the UNIX file descriptor of a file.
                           Useful when combining FIO.mod with select
                           (in Selective.def - but note the comments in
                            Selective about using read/write primatives)
```

(continues on next page)

(continued from previous page)

```
*)

GetUnixFileDescriptor
PROCEDURE GetUnixFileDescriptor (f: File) : INTEGER ;

(*
   SetPositionFromBeginning - sets the position from the beginning
                              of the file.
*)

SetPositionFromBeginning
PROCEDURE SetPositionFromBeginning (f: File; pos: LONGINT) ;

(*
   SetPositionFromEnd - sets the position from the end of the file.
*)

SetPositionFromEnd
PROCEDURE SetPositionFromEnd (f: File; pos: LONGINT) ;

(*
   FindPosition - returns the current absolute position in file, f.
*)

FindPosition
PROCEDURE FindPosition (f: File) : LONGINT ;

(*
   GetFileName - assigns, a, with the filename associated with, f.
*)

GetFileName
PROCEDURE GetFileName (f: File; VAR a: ARRAY OF CHAR) ;

(*
   getFileName - returns the address of the filename associated with, f.
*)

getFileName
PROCEDURE getFileName (f: File) : ADDRESS ;

(*
   getFileNameLength - returns the number of characters associated with
                       filename, f.
*)

getFileNameLength
PROCEDURE getFileNameLength (f: File) : CARDINAL ;

(*
   FlushOutErr - flushes, StdOut, and, StdErr.
```

(continues on next page)

(continued from previous page)

```
*)

FlushOutErr
PROCEDURE FlushOutErr ;

END FIO.
```

18.1.12 gm2-libs/FormatStrings

```
DEFINITION MODULE FormatStrings ;

FROM SYSTEM IMPORT BYTE ;
FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED Sprintf0, Sprintf1, Sprintf2, Sprintf3, Sprintf4,
                 HandleEscape ;

(*
   Sprintf0 - returns a String containing, fmt, after it has had its
              escape sequences translated.
*)

Sprintf0
PROCEDURE Sprintf0 (fmt: String) : String ;

(*
   Sprintf1 - returns a String containing, fmt, together with
              encapsulated entity, w. It only formats the
              first %s or %d with n.
*)

Sprintf1
PROCEDURE Sprintf1 (fmt: String; w: ARRAY OF BYTE) : String ;

(*
   Sprintf2 - returns a string, fmt, which has been formatted.
*)

Sprintf2
PROCEDURE Sprintf2 (fmt: String; w1, w2: ARRAY OF BYTE) : String ;

(*
   Sprintf3 - returns a string, fmt, which has been formatted.
*)

Sprintf3
PROCEDURE Sprintf3 (fmt: String; w1, w2, w3: ARRAY OF BYTE) : String ;

(*
   Sprintf4 - returns a string, fmt, which has been formatted.
*)
```

(continues on next page)

(continued from previous page)

```
Sprintf4
PROCEDURE Sprintf4 (fmt: String;
                    w1, w2, w3, w4: ARRAY OF BYTE) : String ;

(*
   HandleEscape - translates \a, \b, \e, \f, \n, \r, \x[hex] \[octal]
                  into their respective ascii codes.  It also converts
                  \[any] into a single [any] character.
*)

HandleEscape
PROCEDURE HandleEscape (s: String) : String ;

END FormatStrings.
```

18.1.13 gm2-libs/FpuIO

```
DEFINITION MODULE FpuIO ;

EXPORT QUALIFIED ReadReal, WriteReal, StrToReal, RealToStr,
                 ReadLongReal, WriteLongReal, StrToLongReal,
                 LongRealToStr,
                 ReadLongInt, WriteLongInt, StrToLongInt,
                 LongIntToStr ;

ReadReal
PROCEDURE ReadReal (VAR x: REAL) ;
WriteReal
PROCEDURE WriteReal (x: REAL; TotalWidth, FractionWidth: CARDINAL) ;
StrToReal
PROCEDURE StrToReal (a: ARRAY OF CHAR ; VAR x: REAL) ;
RealToStr
PROCEDURE RealToStr (x: REAL; TotalWidth, FractionWidth: CARDINAL;
                     VAR a: ARRAY OF CHAR) ;

ReadLongReal
PROCEDURE ReadLongReal (VAR x: LONGREAL) ;
WriteLongReal
PROCEDURE WriteLongReal (x: LONGREAL;
                         TotalWidth, FractionWidth: CARDINAL) ;
StrToLongReal
PROCEDURE StrToLongReal (a: ARRAY OF CHAR ; VAR x: LONGREAL) ;
LongRealToStr
PROCEDURE LongRealToStr (x: LONGREAL;
                         TotalWidth, FractionWidth: CARDINAL;
                         VAR a: ARRAY OF CHAR) ;

ReadLongInt
PROCEDURE ReadLongInt (VAR x: LONGINT) ;
```

(continues on next page)

(continued from previous page)

```
WriteLongInt
PROCEDURE WriteLongInt (x: LONGINT; n: CARDINAL) ;
StrToLongInt
PROCEDURE StrToLongInt (a: ARRAY OF CHAR ; VAR x: LONGINT) ;
LongIntToStr
PROCEDURE LongIntToStr (x: LONGINT; n: CARDINAL; VAR a: ARRAY OF CHAR) ;

END FpuIO.
```

18.1.14 gm2-libs/GetOpt

```
DEFINITION MODULE GetOpt ;

FROM SYSTEM IMPORT ADDRESS ;
FROM DynamicStrings IMPORT String ;

CONST
no_argument  (const)
   no_argument = 0 ;
required_argument  (const)
   required_argument = 1 ;
optional_argument  (const)
   optional_argument = 2 ;

TYPE
LongOptions (type)
   LongOptions ;
PtrToInteger (type)
   PtrToInteger = POINTER TO INTEGER ;

(*
   GetOpt - call C getopt and fill in the parameters:
            optarg, optind, opterr and optop.
*)

GetOpt
PROCEDURE GetOpt (argc: INTEGER; argv: ADDRESS; optstring: String;
                  VAR optarg: String;
                  VAR optind, opterr, optopt: INTEGER) : CHAR ;

(*
   InitLongOptions - creates and returns a LongOptions empty array.
*)

InitLongOptions
PROCEDURE InitLongOptions () : LongOptions ;

(*
   AddLongOption - appends long option {name, has_arg, flag, val} to the
                   array of options and new long options array is
```

(continues on next page)

(continued from previous page)

```
                  returned.
                  The old array, lo, should no longer be used.

   (from man 3 getopt)
       The meanings of the different fields are:

       name   is the name of the long option.

       has_arg
              is: no_argument (or 0) if the option does not take an
              argument; required_argument (or  1) if the option
              requires an argument; or optional_argument (or 2) if
              the option takes an optional argument.

       flag   specifies how results are returned for a long option.
              If flag is NULL, then getopt_long() returns val.
              (For example, the calling program may set val to the
              equivalent short option character).  Otherwise,
              getopt_long() returns 0, and flag points to a
              variable which is set to val if the option is found,
              but left unchanged if the option is not found.

       val    is the value to return, or to load into the variable
              pointed to by flag.

       The last element of the array has to be filled with zeros.
*)

AddLongOption
PROCEDURE AddLongOption (lo: LongOptions;
                         name: String; has_arg: INTEGER;
                         flag: PtrToInteger;
                         val: INTEGER) : LongOptions ;

(*
   KillLongOptions - returns NIL and also frees up memory
                     associated with, lo.
*)

KillLongOptions
PROCEDURE KillLongOptions (lo: LongOptions) : LongOptions ;

(*
   GetOptLong - works like GetOpt but will accept long options (using
                two dashes).  If the program only accepts long options
                then optstring should be an empty string, not NIL.
*)

GetOptLong
PROCEDURE GetOptLong (argc: INTEGER; argv: ADDRESS; optstring: String;
                      longopts: LongOptions;
```

(continues on next page)

(continued from previous page)

```
                    VAR longindex: INTEGER) : INTEGER ;

(*
   GetOptLongOnly - works like GetOptLong except that a single dash
                    can be used for a long option.
*)

GetOptLongOnly
PROCEDURE GetOptLongOnly (argc: INTEGER; argv: ADDRESS;
                          optstring: String; longopts: LongOptions;
                          VAR longindex: INTEGER) : INTEGER ;

END GetOpt.
```

18.1.15 gm2-libs/IO

```
DEFINITION MODULE IO ;

(*
   Description: provides Read, Write, Errors procedures that map onto UNIX
                file descriptors 0, 1 and 2. This is achieved by using
                FIO if we are in buffered mode and using libc.write
                if not.
*)

EXPORT QUALIFIED Read, Write, Error,
                 UnBufferedMode, BufferedMode,
                 EchoOn, EchoOff ;

Read
PROCEDURE Read (VAR ch: CHAR) ;
Write
PROCEDURE Write (ch: CHAR) ;
Error
PROCEDURE Error (ch: CHAR) ;

(*
   UnBufferedMode - places file descriptor, fd, into an unbuffered mode.
*)

UnBufferedMode
PROCEDURE UnBufferedMode (fd: INTEGER; input: BOOLEAN) ;

(*
   BufferedMode - places file descriptor, fd, into a buffered mode.
*)

BufferedMode
PROCEDURE BufferedMode (fd: INTEGER; input: BOOLEAN) ;
```

(continues on next page)

(continued from previous page)

```
(*
   EchoOn - turns on echoing for file descriptor, fd.  This
            only really makes sence for a file descriptor opened
            for terminal input or maybe some specific file descriptor
            which is attached to a particular piece of hardware.
*)

EchoOn
PROCEDURE EchoOn (fd: INTEGER; input: BOOLEAN) ;

(*
   EchoOff - turns off echoing for file descriptor, fd.  This
             only really makes sence for a file descriptor opened
             for terminal input or maybe some specific file descriptor
             which is attached to a particular piece of hardware.
*)

EchoOff
PROCEDURE EchoOff (fd: INTEGER; input: BOOLEAN) ;

END IO.
```

18.1.16 gm2-libs/Indexing

```
DEFINITION MODULE Indexing ;

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED Index, InitIndex, KillIndex, GetIndice, PutIndice,
                 HighIndice, LowIndice, InBounds, IsIndiceInIndex,
                 RemoveIndiceFromIndex, IncludeIndiceIntoIndex,
                 ForeachIndiceInIndexDo, DeleteIndice, DebugIndex ;

TYPE
Index (type)
   Index ;
IndexProcedure (type)
   IndexProcedure = PROCEDURE (ADDRESS) ;

(*
   InitIndex - creates and returns an Index.
*)

InitIndex
PROCEDURE InitIndex (low: CARDINAL) : Index ;

(*
   KillIndex - returns Index to free storage.
*)

KillIndex
```

(continues on next page)

(continued from previous page)

```
PROCEDURE KillIndex (i: Index) : Index ;

(*
   DebugIndex - turns on debugging within an index.
*)

DebugIndex
PROCEDURE DebugIndex (i: Index) : Index ;

(*
   InBounds - returns TRUE if indice, n, is within the bounds
              of the dynamic array.
*)

InBounds
PROCEDURE InBounds (i: Index; n: CARDINAL) : BOOLEAN ;

(*
   HighIndice - returns the last legally accessible indice of this array.
*)

HighIndice
PROCEDURE HighIndice (i: Index) : CARDINAL ;

(*
   LowIndice - returns the first legally accessible indice of this array.
*)

LowIndice
PROCEDURE LowIndice (i: Index) : CARDINAL ;

(*
   PutIndice - places, a, into the dynamic array at position i[n]
*)

PutIndice
PROCEDURE PutIndice (i: Index; n: CARDINAL; a: ADDRESS) ;

(*
   GetIndice - retrieves, element i[n] from the dynamic array.
*)

GetIndice
PROCEDURE GetIndice (i: Index; n: CARDINAL) : ADDRESS ;

(*
   IsIndiceInIndex - returns TRUE if, a, is in the index, i.
*)

IsIndiceInIndex
PROCEDURE IsIndiceInIndex (i: Index; a: ADDRESS) : BOOLEAN ;
```

(continues on next page)

(continued from previous page)

```
(*
   RemoveIndiceFromIndex - removes, a, from Index, i.
*)

RemoveIndiceFromIndex
PROCEDURE RemoveIndiceFromIndex (i: Index; a: ADDRESS) ;

(*
   DeleteIndice - delete i[j] from the array.
*)

DeleteIndice
PROCEDURE DeleteIndice (i: Index; j: CARDINAL) ;

(*
   IncludeIndiceIntoIndex - if the indice is not in the index, then
                            add it at the end.
*)

IncludeIndiceIntoIndex
PROCEDURE IncludeIndiceIntoIndex (i: Index; a: ADDRESS) ;

(*
   ForeachIndiceInIndexDo - for each j indice of i, call procedure p(i[j])
*)

ForeachIndiceInIndexDo
PROCEDURE ForeachIndiceInIndexDo (i: Index; p: IndexProcedure) ;

END Indexing.
```

18.1.17 gm2-libs/LMathLib0

```
DEFINITION MODULE LMathLib0 ;

CONST
   pi   = 3.14159265358979323846264338327950288419716939937510;
   exp1 = 2.71828182845904523536028747135266249775724709369995;

sqrt
PROCEDURE __BUILTIN__ sqrt (x: LONGREAL) : LONGREAL ;
exp
PROCEDURE exp (x: LONGREAL) : LONGREAL ;
ln
PROCEDURE ln (x: LONGREAL) : LONGREAL ;
sin
PROCEDURE __BUILTIN__ sin (x: LONGREAL) : LONGREAL ;
cos
PROCEDURE __BUILTIN__ cos (x: LONGREAL) : LONGREAL ;
```

(continues on next page)

(continued from previous page)

```
tan
PROCEDURE tan (x: LONGREAL) : LONGREAL ;
arctan
PROCEDURE arctan (x: LONGREAL) : LONGREAL ;
entier
PROCEDURE entier (x: LONGREAL) : INTEGER ;

END LMathLib0.
```

18.1.18 gm2-libs/LegacyReal

```
DEFINITION MODULE LegacyReal ;

TYPE
   REAL = SHORTREAL ;

END LegacyReal.
```

18.1.19 gm2-libs/M2Dependent

```
DEFINITION MODULE M2Dependent ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
ArgCVEnvP (type)
   ArgCVEnvP = PROCEDURE (INTEGER, ADDRESS, ADDRESS) ;

ConstructModules
PROCEDURE ConstructModules (applicationmodule: ADDRESS;
                            argc: INTEGER; argv, envp: ADDRESS) ;

DeconstructModules
PROCEDURE DeconstructModules (applicationmodule: ADDRESS;
                              argc: INTEGER; argv, envp: ADDRESS) ;

(*
   RegisterModule - adds module name to the list of outstanding
                    modules which need to have their dependencies
                    explored to determine initialization order.
*)

RegisterModule
PROCEDURE RegisterModule (name: ADDRESS;
                          init, fini:  ArgCVEnvP;
                          dependencies: PROC) ;
```

(continues on next page)

(continued from previous page)

```
(*
   RequestDependant - used to specify that modulename is dependant upon
                      module dependantmodule.
*)

RequestDependant
PROCEDURE RequestDependant (modulename, dependantmodule: ADDRESS) ;

END M2Dependent.
```

18.1.20 gm2-libs/M2EXCEPTION

```
DEFINITION MODULE M2EXCEPTION;

(* This enumerated list of exceptions must match the exceptions in gm2-libs-iso to
   allow mixed module dialect projects.  *)

TYPE
M2Exceptions (type)
  M2Exceptions =
    (indexException,     rangeException,         caseSelectException,  invalidLocation,
     functionException,  wholeValueException,    wholeDivException,    realValueException,
     realDivException,   complexValueException,  complexDivException,  protException,
     sysException,       coException,            exException
    );

(* If the program or coroutine is in the exception state then return the enumeration
   value representing the exception cause.  If it is not in the exception state then
   raises and exception (exException).  *)

M2Exception
PROCEDURE M2Exception () : M2Exceptions;

(* Returns TRUE if the program or coroutine is in the exception state.
   Returns FALSE if the program or coroutine is not in the exception state.  *)

IsM2Exception
PROCEDURE IsM2Exception () : BOOLEAN;

END M2EXCEPTION.
```

18.1.21 gm2-libs/M2LINK

```
DEFINITION MODULE FOR "C" M2LINK ;

TYPE
PtrToChar (type)
   PtrToChar = POINTER TO CHAR ;

(* These variables are set by the compiler in the program module
   according to linking command line options.  *)

VAR
ForcedModuleInitOrder (var)
   ForcedModuleInitOrder: PtrToChar ;
StaticInitialization  (var)
   StaticInitialization : BOOLEAN ;

END M2LINK. (var)
END M2LINK.
```

18.1.22 gm2-libs/M2RTS

```
DEFINITION MODULE M2RTS ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
ArgCVEnvP (type)
   ArgCVEnvP = PROCEDURE (INTEGER, ADDRESS, ADDRESS) ;

ConstructModules
PROCEDURE ConstructModules (applicationmodule: ADDRESS;
                            argc: INTEGER; argv, envp: ADDRESS) ;

DeconstructModules
PROCEDURE DeconstructModules (applicationmodule: ADDRESS;
                              argc: INTEGER; argv, envp: ADDRESS) ;

(*
   RegisterModule - adds module name to the list of outstanding
                    modules which need to have their dependencies
                    explored to determine initialization order.
*)

RegisterModule
PROCEDURE RegisterModule (name: ADDRESS;
                          init, fini:  ArgCVEnvP;
                          dependencies: PROC) ;

(*
```

(continues on next page)

(continued from previous page)

```
   RequestDependant - used to specify that modulename is dependant upon
                      module dependantmodule.
*)

RequestDependant
PROCEDURE RequestDependant (modulename, dependantmodule: ADDRESS) ;

(*
   InstallTerminationProcedure - installs a procedure, p, which will
                                 be called when the procedure
                                 ExecuteTerminationProcedures
                                 is invoked.  It returns TRUE is the
                                 procedure is installed.
*)

InstallTerminationProcedure
PROCEDURE InstallTerminationProcedure (p: PROC) : BOOLEAN ;

(*
   ExecuteInitialProcedures - executes the initial procedures installed
                              by InstallInitialProcedure.
*)

ExecuteInitialProcedures
PROCEDURE ExecuteInitialProcedures ;

(*
   InstallInitialProcedure - installs a procedure to be executed just
                             before the BEGIN code section of the main
                             program module.
*)

InstallInitialProcedure
PROCEDURE InstallInitialProcedure (p: PROC) : BOOLEAN ;

(*
   ExecuteTerminationProcedures - calls each installed termination procedure
                                  in reverse order.
*)

ExecuteTerminationProcedures
PROCEDURE ExecuteTerminationProcedures ;

(*
   Terminate - provides compatibility for pim.  It call exit with
               the exitcode provided in a prior call to ExitOnHalt
               (or zero if ExitOnHalt was never called).  It does
               not call ExecuteTerminationProcedures.
*)

Terminate
```

(continues on next page)

(continued from previous page)

```
PROCEDURE Terminate <* noreturn *> ;

(*
   HALT - terminate the current program.  The procedure Terminate
          is called before the program is stopped.  The parameter
          exitcode is optional.  If the parameter is not supplied
          HALT will call libc 'abort', otherwise it will exit with
          the code supplied.  Supplying a parameter to HALT has the
          same effect as calling ExitOnHalt with the same code and
          then calling HALT with no parameter.
*)

HALT
PROCEDURE HALT ([exitcode: INTEGER = -1]) <* noreturn *> ;

(*
   Halt - provides a more user friendly version of HALT, which takes
          four parameters to aid debugging.
*)

Halt
PROCEDURE Halt (file: ARRAY OF CHAR; line: CARDINAL;
                function: ARRAY OF CHAR; description: ARRAY OF CHAR)
             <* noreturn *> ;

(*
   ExitOnHalt - if HALT is executed then call exit with the exit code, e.
*)

ExitOnHalt
PROCEDURE ExitOnHalt (e: INTEGER) ;

(*
   ErrorMessage - emits an error message to stderr and then calls exit (1).
*)

ErrorMessage
PROCEDURE ErrorMessage (message: ARRAY OF CHAR;
                        file: ARRAY OF CHAR;
                        line: CARDINAL;
                        function: ARRAY OF CHAR) <* noreturn *> ;

(*
   Length - returns the length of a string, a. This is called whenever
            the user calls LENGTH and the parameter cannot be calculated
            at compile time.
*)

Length
PROCEDURE Length (a: ARRAY OF CHAR) : CARDINAL ;
```

(continues on next page)

(continued from previous page)

```
(*
   The following are the runtime exception handler routines.
*)

AssignmentException
PROCEDURE AssignmentException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ReturnException
PROCEDURE ReturnException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
IncException
PROCEDURE IncException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
DecException
PROCEDURE DecException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
InclException
PROCEDURE InclException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
ExclException
PROCEDURE ExclException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
ShiftException
PROCEDURE ShiftException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
RotateException
PROCEDURE RotateException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
StaticArraySubscriptException
PROCEDURE StaticArraySubscriptException (filename: ADDRESS; line, column: CARDINAL; scope,␣
↪message: ADDRESS) ;
DynamicArraySubscriptException
PROCEDURE DynamicArraySubscriptException (filename: ADDRESS; line, column: CARDINAL; scope,␣
↪message: ADDRESS) ;
ForLoopBeginException
PROCEDURE ForLoopBeginException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ForLoopToException
PROCEDURE ForLoopToException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ForLoopEndException
PROCEDURE ForLoopEndException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
PointerNilException
PROCEDURE PointerNilException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
NoReturnException
PROCEDURE NoReturnException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS)␣
↪;
CaseException
PROCEDURE CaseException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
WholeNonPosDivException
PROCEDURE WholeNonPosDivException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeNonPosModException
PROCEDURE WholeNonPosModException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeZeroDivException
```

(continues on next page)

(continued from previous page)

```
PROCEDURE WholeZeroDivException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeZeroRemException
PROCEDURE WholeZeroRemException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeValueException
PROCEDURE WholeValueException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
RealValueException
PROCEDURE RealValueException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ParameterException
PROCEDURE ParameterException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
NoException
PROCEDURE NoException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;

END M2RTS.
```

18.1.23 gm2-libs/MathLib0

```
DEFINITION MODULE MathLib0 ;

CONST
   pi   = 3.14159265358979323846264338327950288419 72;
   exp1 = 2.71828182845904523536028747135266249775 72;

sqrt
PROCEDURE __BUILTIN__ sqrt (x: REAL) : REAL ;
exp
PROCEDURE exp (x: REAL) : REAL ;
ln
PROCEDURE ln (x: REAL) : REAL ;
sin
PROCEDURE __BUILTIN__ sin (x: REAL) : REAL ;
cos
PROCEDURE __BUILTIN__ cos (x: REAL) : REAL ;
tan
PROCEDURE tan (x: REAL) : REAL ;
arctan
PROCEDURE arctan (x: REAL) : REAL ;
entier
PROCEDURE entier (x: REAL) : INTEGER ;

END MathLib0.
```

18.1.24 gm2-libs/MemUtils

```
DEFINITION MODULE MemUtils ;

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED MemCopy, MemZero ;

(*
   MemCopy - copys a region of memory to the required destination.
*)

MemCopy
PROCEDURE MemCopy (from: ADDRESS; length: CARDINAL; to: ADDRESS) ;

(*
   MemZero - sets a region of memory: a..a+length to zero.
*)

MemZero
PROCEDURE MemZero (a: ADDRESS; length: CARDINAL) ;

END MemUtils.
```

18.1.25 gm2-libs/NumberIO

```
DEFINITION MODULE NumberIO ;

EXPORT QUALIFIED ReadCard, WriteCard, ReadHex, WriteHex, ReadInt, WriteInt,
                 CardToStr, StrToCard, StrToHex, HexToStr, StrToInt, IntToStr,
                 ReadOct, WriteOct, OctToStr, StrToOct,
                 ReadBin, WriteBin, BinToStr, StrToBin,
                 StrToBinInt, StrToHexInt, StrToOctInt ;

ReadCard
PROCEDURE ReadCard (VAR x: CARDINAL) ;

WriteCard
PROCEDURE WriteCard (x, n: CARDINAL) ;

ReadHex
PROCEDURE ReadHex (VAR x: CARDINAL) ;

WriteHex
PROCEDURE WriteHex (x, n: CARDINAL) ;

ReadInt
PROCEDURE ReadInt (VAR x: INTEGER) ;

WriteInt
PROCEDURE WriteInt (x: INTEGER ; n: CARDINAL) ;
```

(continues on next page)

(continued from previous page)

```
CardToStr
PROCEDURE CardToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

StrToCard
PROCEDURE StrToCard (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

HexToStr
PROCEDURE HexToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

StrToHex
PROCEDURE StrToHex (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

IntToStr
PROCEDURE IntToStr (x: INTEGER ; n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

StrToInt
PROCEDURE StrToInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

ReadOct
PROCEDURE ReadOct (VAR x: CARDINAL) ;

WriteOct
PROCEDURE WriteOct (x, n: CARDINAL) ;

OctToStr
PROCEDURE OctToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

StrToOct
PROCEDURE StrToOct (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

ReadBin
PROCEDURE ReadBin (VAR x: CARDINAL) ;

WriteBin
PROCEDURE WriteBin (x, n: CARDINAL) ;

BinToStr
PROCEDURE BinToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

StrToBin
PROCEDURE StrToBin (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

StrToBinInt
PROCEDURE StrToBinInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

StrToHexInt
PROCEDURE StrToHexInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

StrToOctInt
PROCEDURE StrToOctInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;
```

(continues on next page)

(continued from previous page)

```
END NumberIO.
```

18.1.26 gm2-libs/OptLib

```
DEFINITION MODULE OptLib ;

FROM SYSTEM IMPORT ADDRESS ;
FROM DynamicStrings IMPORT String ;

TYPE
Option (type)
   Option ;

(*
   InitOption - constructor for Option.
*)

InitOption
PROCEDURE InitOption (argc: INTEGER; argv: ADDRESS) : Option ;

(*
   KillOption - deconstructor for Option.
*)

KillOption
PROCEDURE KillOption (o: Option) : Option ;

(*
   Dup - duplicate the option array inside, o.
         Notice that this does not duplicate all the contents
         (strings) of argv.
         Shallow copy of the top level indices.
*)

Dup
PROCEDURE Dup (o: Option) : Option ;

(*
   Slice - return a new option which has elements [low:high] from the
           options, o.
*)

Slice
PROCEDURE Slice (o: Option; low, high: INTEGER) : Option ;

(*
   IndexStrCmp - returns the index in the argv array which matches
                 string, s.  -1 is returned if the string is not found.
*)
```

(continues on next page)

(continued from previous page)

```
IndexStrCmp
PROCEDURE IndexStrCmp (o: Option; s: String) : INTEGER ;

(*
   IndexStrNCmp - returns the index in the argv array where the first
                  characters are matched by string, s.
                  -1 is returned if the string is not found.
*)

IndexStrNCmp
PROCEDURE IndexStrNCmp (o: Option; s: String) : INTEGER ;

(*
   ConCat - returns the concatenation of a and b.
*)

ConCat
PROCEDURE ConCat (a, b: Option) : Option ;

(*
   GetArgv - return the argv component of option.
*)

GetArgv
PROCEDURE GetArgv (o: Option) : ADDRESS ;

(*
   GetArgc - return the argc component of option.
*)

GetArgc
PROCEDURE GetArgc (o: Option) : INTEGER ;

END OptLib.
```

18.1.27 gm2-libs/PushBackInput

```
DEFINITION MODULE PushBackInput ;

FROM FIO IMPORT File ;
FROM DynamicStrings IMPORT String ;

EXPORT QUALIFIED Open, PutCh, GetCh, Error, WarnError, WarnString,
                 Close, SetDebug, GetExitStatus, PutStr,
                 PutString, GetColumnPosition, GetCurrentLine ;

(*
   Open - opens a file for reading.
*)
```

(continues on next page)

(continued from previous page)

```
Open
PROCEDURE Open (a: ARRAY OF CHAR) : File ;

(*
   GetCh - gets a character from either the push back stack or
           from file, f.
*)

GetCh
PROCEDURE GetCh (f: File) : CHAR ;

(*
   PutCh - pushes a character onto the push back stack, it also
           returns the character which has been pushed.
*)

PutCh
PROCEDURE PutCh (ch: CHAR) : CHAR ;

(*
   PutString - pushes a string onto the push back stack.
*)

PutString
PROCEDURE PutString (a: ARRAY OF CHAR) ;

(*
   PutStr - pushes a dynamic string onto the push back stack.
            The string, s, is not deallocated.
*)

PutStr
PROCEDURE PutStr (s: String) ;

(*
   Error - emits an error message with the appropriate file, line combination.
*)

Error
PROCEDURE Error (a: ARRAY OF CHAR) ;

(*
   WarnError - emits an error message with the appropriate file, line combination.
               It does not terminate but when the program finishes an exit status of
               1 will be issued.
*)

WarnError
PROCEDURE WarnError (a: ARRAY OF CHAR) ;
```

(continues on next page)

(continued from previous page)

```
(*
   WarnString - emits an error message with the appropriate file, line combination.
                It does not terminate but when the program finishes an exit status of
                1 will be issued.
*)

WarnString
PROCEDURE WarnString (s: String) ;

(*
   Close - closes the opened file.
*)

Close
PROCEDURE Close (f: File) ;

(*
   GetExitStatus - returns the exit status which will be 1 if any warnings were issued.
*)

GetExitStatus
PROCEDURE GetExitStatus () : CARDINAL ;

(*
   SetDebug - sets the debug flag on or off.
*)

SetDebug
PROCEDURE SetDebug (d: BOOLEAN) ;

(*
   GetColumnPosition - returns the column position of the current character.
*)

GetColumnPosition
PROCEDURE GetColumnPosition () : CARDINAL ;

(*
   GetCurrentLine - returns the current line number.
*)

GetCurrentLine
PROCEDURE GetCurrentLine () : CARDINAL ;

END PushBackInput.
```

18.1.28 gm2-libs/RTExceptions

```
DEFINITION MODULE RTExceptions ;

(* Runtime exception handler routines.  This should
   be considered as a system module for GNU Modula-2
   and allow the compiler to interface with exception
   handling.  *)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED EHBlock,
                 Raise, SetExceptionBlock, GetExceptionBlock,
                 GetTextBuffer, GetTextBufferSize, GetNumber,
                 InitExceptionBlock, KillExceptionBlock,
                 PushHandler, PopHandler,
                 BaseExceptionsThrow, DefaultErrorCatch,
                 IsInExceptionState, SetExceptionState,
                 SwitchExceptionState, GetBaseExceptionBlock,
                 SetExceptionSource, GetExceptionSource ;

TYPE
EHBlock (type)
   EHBlock ;
ProcedureHandler (type)
   ProcedureHandler = PROCEDURE ;

(*
   Raise - invoke the exception handler associated with, number,
           in the active EHBlock.  It keeps a record of the number
           and message in the EHBlock for later use.
*)

Raise
PROCEDURE Raise (number: CARDINAL;
                 file: ADDRESS; line: CARDINAL;
                 column: CARDINAL; function: ADDRESS;
                 message: ADDRESS) ;

(*
   SetExceptionBlock - sets, source, as the active EHB.
*)

SetExceptionBlock
PROCEDURE SetExceptionBlock (source: EHBlock) ;

(*
   GetExceptionBlock - returns the active EHB.
*)

GetExceptionBlock
PROCEDURE GetExceptionBlock () : EHBlock ;
```

(continues on next page)

(continued from previous page)

```
(*
   GetTextBuffer - returns the address of the EHB buffer.
*)

GetTextBuffer
PROCEDURE GetTextBuffer (e: EHBlock) : ADDRESS ;

(*
   GetTextBufferSize - return the size of the EHB text buffer.
*)

GetTextBufferSize
PROCEDURE GetTextBufferSize (e: EHBlock) : CARDINAL ;

(*
   GetNumber - return the exception number associated with,
               source.
*)

GetNumber
PROCEDURE GetNumber (source: EHBlock) : CARDINAL ;

(*
   InitExceptionBlock - creates and returns a new exception block.
*)

InitExceptionBlock
PROCEDURE InitExceptionBlock () : EHBlock ;

(*
   KillExceptionBlock - destroys the EHB, e, and all its handlers.
*)

KillExceptionBlock
PROCEDURE KillExceptionBlock (e: EHBlock) : EHBlock ;

(*
   PushHandler - install a handler in EHB, e.
*)

PushHandler
PROCEDURE PushHandler (e: EHBlock; number: CARDINAL; p: ProcedureHandler) ;

(*
   PopHandler - removes the handler associated with, number, from
                EHB, e.
*)

PopHandler
PROCEDURE PopHandler (e: EHBlock; number: CARDINAL) ;
```

(continues on next page)

(continued from previous page)

```
(*
   DefaultErrorCatch - displays the current error message in
                       the current exception block and then
                       calls HALT.
*)

DefaultErrorCatch
PROCEDURE DefaultErrorCatch ;

(*
   BaseExceptionsThrow - configures the Modula-2 exceptions to call
                         THROW which in turn can be caught by an
                         exception block.  If this is not called then
                         a Modula-2 exception will simply call an
                         error message routine and then HALT.
*)

BaseExceptionsThrow
PROCEDURE BaseExceptionsThrow ;

(*
   IsInExceptionState - returns TRUE if the program is currently
                        in the exception state.
*)

IsInExceptionState
PROCEDURE IsInExceptionState () : BOOLEAN ;

(*
   SetExceptionState - returns the current exception state and
                       then sets the current exception state to,
                       to.
*)

SetExceptionState
PROCEDURE SetExceptionState (to: BOOLEAN) : BOOLEAN ;

(*
   SwitchExceptionState - assigns, from, with the current exception
                          state and then assigns the current exception
                          to, to.
*)

SwitchExceptionState
PROCEDURE SwitchExceptionState (VAR from: BOOLEAN; to: BOOLEAN) ;

(*
   GetBaseExceptionBlock - returns the initial language exception block
                           created.
*)
```

(continues on next page)

(continued from previous page)

```
GetBaseExceptionBlock
PROCEDURE GetBaseExceptionBlock () : EHBlock ;

(*
   SetExceptionSource - sets the current exception source to, source.
*)

SetExceptionSource
PROCEDURE SetExceptionSource (source: ADDRESS) ;

(*
   GetExceptionSource - returns the current exception source.
*)

GetExceptionSource
PROCEDURE GetExceptionSource () : ADDRESS ;

END RTExceptions.
```

18.1.29 gm2-libs/RTint

```
DEFINITION MODULE RTint ;

(* Provides users of the COROUTINES library with the
   ability to create interrupt sources based on
   file descriptors and timeouts.  *)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
DispatchVector (type)
   DispatchVector = PROCEDURE (CARDINAL, CARDINAL, ADDRESS) ;

(*
   InitInputVector - returns an interrupt vector which is associated
                     with the file descriptor, fd.
*)

InitInputVector
PROCEDURE InitInputVector (fd: INTEGER; pri: CARDINAL) : CARDINAL ;

(*
   InitOutputVector - returns an interrupt vector which is associated
                      with the file descriptor, fd.
*)

InitOutputVector
PROCEDURE InitOutputVector (fd: INTEGER; pri: CARDINAL) : CARDINAL ;

(*
```

(continues on next page)

(continued from previous page)

```
   InitTimeVector - returns an interrupt vector associated with
                    the relative time.
*)

InitTimeVector
PROCEDURE InitTimeVector (micro, secs: CARDINAL; pri: CARDINAL) : CARDINAL ;

(*
   ReArmTimeVector - reprimes the vector, vec, to deliver an interrupt
                     at the new relative time.
*)

ReArmTimeVector
PROCEDURE ReArmTimeVector (vec: CARDINAL; micro, secs: CARDINAL) ;

(*
   GetTimeVector - assigns, micro, and, secs, with the remaining
                   time before this interrupt will expire.
                   This value is only updated when a Listen
                   occurs.
*)

GetTimeVector
PROCEDURE GetTimeVector (vec: CARDINAL; VAR micro, secs: CARDINAL) ;

(*
   AttachVector - adds the pointer, p, to be associated with the interrupt
                  vector. It returns the previous value attached to this
                  vector.
*)

AttachVector
PROCEDURE AttachVector (vec: CARDINAL; p: ADDRESS) : ADDRESS ;

(*
   IncludeVector - includes, vec, into the dispatcher list of
                   possible interrupt causes.
*)

IncludeVector
PROCEDURE IncludeVector (vec: CARDINAL) ;

(*
   ExcludeVector - excludes, vec, from the dispatcher list of
                   possible interrupt causes.
*)

ExcludeVector
PROCEDURE ExcludeVector (vec: CARDINAL) ;

(*
```

(continues on next page)

(continued from previous page)

```
   Listen - will either block indefinitely (until an interrupt)
            or alteratively will test to see whether any interrupts
            are pending.
            If a pending interrupt was found then, call, is called
            and then this procedure returns.
            It only listens for interrupts > pri.
*)

Listen
PROCEDURE Listen (untilInterrupt: BOOLEAN;
                  call: DispatchVector;
                  pri: CARDINAL) ;

(*
   Init - allows the user to force the initialize order.
*)

Init
PROCEDURE Init ;

END RTint.
```

18.1.30 gm2-libs/SArgs

```
DEFINITION MODULE SArgs ;

FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED GetArg, Narg ;

(*
   GetArg - returns the nth argument from the command line.
            The success of the operation is returned.
            If TRUE is returned then the string, s, contains a
            new string, otherwise s is set to NIL.
*)

GetArg
PROCEDURE GetArg (VAR s: String ; n: CARDINAL) : BOOLEAN ;

(*
   Narg - returns the number of arguments available from
          command line.
*)

Narg
PROCEDURE Narg() : CARDINAL ;

END SArgs.
```

18.1.31 gm2-libs/SCmdArgs

```
DEFINITION MODULE SCmdArgs ;

FROM DynamicStrings IMPORT String ;

EXPORT QUALIFIED GetArg, Narg ;

(*
   GetArg - returns the nth argument from the command line, CmdLine
            the success of the operation is returned.
*)

GetArg
PROCEDURE GetArg (CmdLine: String;
                  n: CARDINAL; VAR Argi: String) : BOOLEAN ;

(*
   Narg - returns the number of arguments available from
          command line, CmdLine.
*)

Narg
PROCEDURE Narg (CmdLine: String) : CARDINAL ;

END SCmdArgs.
```

18.1.32 gm2-libs/SEnvironment

```
DEFINITION MODULE SEnvironment ;

FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED GetEnvironment ;

(*
   GetEnvironment - gets the environment variable Env and places
                          a copy of its value into String, dest.
                    It returns TRUE if the string Env was found in
                    the processes environment.
*)

GetEnvironment
PROCEDURE GetEnvironment (Env: String;
                          VAR dest: String) : BOOLEAN ;

(*
   PutEnvironment - change or add an environment variable definition EnvDef.
                    TRUE is returned if the environment variable was
                    set or changed successfully.
*)
```

(continues on next page)

(continued from previous page)

```
PutEnvironment
PROCEDURE PutEnvironment (EnvDef: String) : BOOLEAN ;

END SEnvironment.
```

18.1.33 gm2-libs/SFIO

```
DEFINITION MODULE SFIO ;

FROM DynamicStrings IMPORT String ;
FROM FIO IMPORT File ;

EXPORT QUALIFIED OpenToRead, OpenToWrite, OpenForRandom, Exists, WriteS, ReadS ;

(*
   Exists - returns TRUE if a file named, fname exists for reading.
*)

Exists
PROCEDURE Exists (fname: String) : BOOLEAN ;

(*
   OpenToRead - attempts to open a file, fname, for reading and
                it returns this file.
                The success of this operation can be checked by
                calling IsNoError.
*)

OpenToRead
PROCEDURE OpenToRead (fname: String) : File ;

(*
   OpenToWrite - attempts to open a file, fname, for write and
                 it returns this file.
                 The success of this operation can be checked by
                 calling IsNoError.
*)

OpenToWrite
PROCEDURE OpenToWrite (fname: String) : File ;

(*
   OpenForRandom - attempts to open a file, fname, for random access
                   read or write and it returns this file.
                   The success of this operation can be checked by
                   calling IsNoError.
                   towrite, determines whether the file should be
                   opened for writing or reading.
                   if towrite is TRUE or whether the previous file should
```

(continues on next page)

(continued from previous page)

```
                  be left alone, allowing this descriptor to seek
                  and modify an existing file.
*)

OpenForRandom
PROCEDURE OpenForRandom (fname: String; towrite, newfile: BOOLEAN) : File ;

(*
   WriteS - writes a string, s, to, file. It returns the String, s.
*)

WriteS
PROCEDURE WriteS (file: File; s: String) : String ;

(*
   ReadS - reads a string, s, from, file. It returns the String, s.
           It stops reading the string at the end of line or end of file.
           It consumes the newline at the end of line but does not place
           this into the returned string.
*)

ReadS
PROCEDURE ReadS (file: File) : String ;

END SFIO.
```

18.1.34 gm2-libs/SMathLib0

```
DEFINITION MODULE SMathLib0 ;

CONST
   pi   = 3.14159265358979323846264338327950288419716939937510;
   exp1 = 2.71828182845904523536028747135266249775724709369995;

sqrt
PROCEDURE __BUILTIN__ sqrt (x: SHORTREAL) : SHORTREAL ;
exp
PROCEDURE exp (x: SHORTREAL) : SHORTREAL ;
ln
PROCEDURE ln (x: SHORTREAL) : SHORTREAL ;
sin
PROCEDURE __BUILTIN__ sin (x: SHORTREAL) : SHORTREAL ;
cos
PROCEDURE __BUILTIN__ cos (x: SHORTREAL) : SHORTREAL ;
tan
PROCEDURE tan (x: SHORTREAL) : SHORTREAL ;
arctan
PROCEDURE arctan (x: SHORTREAL) : SHORTREAL ;
entier
PROCEDURE entier (x: SHORTREAL) : INTEGER ;
```

(continues on next page)

(continued from previous page)

```
END SMathLib0.
```

18.1.35 gm2-libs/SYSTEM

```
DEFINITION MODULE SYSTEM ;

EXPORT QUALIFIED BITSPERBYTE, BYTESPERWORD,
                 LOC, WORD, BYTE, ADDRESS, INTEGER8,
                 INTEGER16, INTEGER32, INTEGER64, CARDINAL8,
                 CARDINAL16, CARDINAL32, CARDINAL64, WORD16,
                 WORD32, WORD64, BITSET8, BITSET16,
                 BITSET32, REAL32, REAL64, REAL128,
                 COMPLEX32, COMPLEX64, COMPLEX128, CSIZE_T,
                 CSSIZE_T,
                 ADR, TSIZE, ROTATE, SHIFT, THROW, TBITSIZE ;
                 (* SIZE is also exported if -fpim2 is used *)

CONST
BITSPERBYTE    (const)
  BITSPERBYTE   = __ATTRIBUTE__ __BUILTIN__ ((BITS_PER_UNIT)) ;
BYTESPERWORD   (const)
  BYTESPERWORD  = __ATTRIBUTE__ __BUILTIN__ ((UNITS_PER_WORD)) ;

(* all the following types are declared internally to gm2
TYPE
LOC (type)
   LOC ;
WORD (type)
   WORD ;
BYTE (type)
   BYTE ;
ADDRESS (type)
   ADDRESS ;
INTEGER8 (type)
   INTEGER8 ;
INTEGER16 (type)
   INTEGER16 ;
INTEGER32 (type)
   INTEGER32 ;
INTEGER64 (type)
   INTEGER64 ;
CARDINAL8 (type)
   CARDINAL8 ;
CARDINAL16 (type)
   CARDINAL16 ;
CARDINAL32 (type)
   CARDINAL32 ;
CARDINAL64 (type)
   CARDINAL64 ;
```

(continues on next page)

(continued from previous page)

```
WORD16 (type)
   WORD16 ;
WORD32 (type)
   WORD32 ;
WORD64 (type)
   WORD64 ;
BITSET8 (type)
   BITSET8 ;
BITSET16 (type)
   BITSET16 ;
BITSET32 (type)
   BITSET32 ;
REAL32 (type)
   REAL32 ;
REAL64 (type)
   REAL64 ;
REAL128 (type)
   REAL128 ;
COMPLEX32 (type)
   COMPLEX32 ;
COMPLEX64 (type)
   COMPLEX64 ;
COMPLEX128 (type)
   COMPLEX128 ;
CSIZE_T (type)
   CSIZE_T ;
CSSIZE_T (type)
   CSSIZE_T ;
*)

(*
   all the functions below are declared internally to gm2
   ======================================================

ADR
PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
  (* Returns the address of variable v. *)

SIZE
PROCEDURE SIZE (v: <type>) : ZType;
  (* Returns the number of BYTES used to store a v of
     any specified <type>.  Only available if -fpim2 is used.
  *)

TSIZE
PROCEDURE TSIZE (<type>) : CARDINAL;
  (* Returns the number of BYTES used to store a value of the
     specified <type>.
  *)

ROTATE
```

(continues on next page)

(continued from previous page)

```
PROCEDURE ROTATE (val: <a set type>;
                  num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by rotating up/right
     or down/right by the absolute value of num.  The direction is
     down/right if the sign of num is negative, otherwise the direction
     is up/left.
  *)

SHIFT
PROCEDURE SHIFT (val: <a set type>;
                 num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by shifting up/left
     or down/right by the absolute value of num, introducing
     zeros as necessary.  The direction is down/right if the sign of
     num is negative, otherwise the direction is up/left.
  *)

THROW
PROCEDURE THROW (i: INTEGER) ;
  (*
     THROW is a GNU extension and was not part of the PIM or ISO
     standards.  It throws an exception which will be caught by the
     EXCEPT block (assuming it exists).  This is a compiler builtin
     function which interfaces to the GCC exception handling runtime
     system.
     GCC uses the term throw, hence the naming distinction between
     the GCC builtin and the Modula-2 runtime library procedure Raise.
     The later library procedure Raise will call SYSTEM.THROW after
     performing various housekeeping activities.
  *)

TBITSIZE
PROCEDURE TBITSIZE (<type>) : CARDINAL ;
  (* Returns the minimum number of bits necessary to represent
     <type>.  This procedure function is only useful for determining
     the number of bits used for any type field within a packed RECORD.
     It is not particularly useful elsewhere since <type> might be
     optimized for speed, for example a BOOLEAN could occupy a WORD.
  *)
*)

(* The following procedures are invoked by GNU Modula-2 to
   shift non word sized set types. They are not strictly part
   of the core PIM Modula-2, however they are used by
   GNU Modula-2 to implement the SHIFT procedure defined above,
   which are in turn used by the Logitech compatible libraries.

   Users will access these procedures by using the procedure
   SHIFT above and GNU Modula-2 will map SHIFT onto one of
   the following procedures.
*)
```

(continues on next page)

(continued from previous page)

```
(*
   ShiftVal - is a runtime procedure whose job is to implement
              the SHIFT procedure of ISO SYSTEM. GNU Modula-2 will
              inline a SHIFT of a single WORD sized set and will only
              call this routine for larger sets.
*)

ShiftVal
PROCEDURE ShiftVal (VAR s, d: ARRAY OF BITSET;
                    SetSizeInBits: CARDINAL;
                    ShiftCount: INTEGER) ;

(*
   ShiftLeft - performs the shift left for a multi word set.
               This procedure might be called by the back end of
               GNU Modula-2 depending whether amount is known at
               compile time.
*)

ShiftLeft
PROCEDURE ShiftLeft (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     ShiftCount: CARDINAL) ;

(*
   ShiftRight - performs the shift left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

ShiftRight
PROCEDURE ShiftRight (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      ShiftCount: CARDINAL) ;

(*
   RotateVal - is a runtime procedure whose job is to implement
               the ROTATE procedure of ISO SYSTEM. GNU Modula-2 will
               inline a ROTATE of a single WORD (or less)
               sized set and will only call this routine for larger
               sets.
*)

RotateVal
PROCEDURE RotateVal (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     RotateCount: INTEGER) ;

(*
```

(continues on next page)

(continued from previous page)

```
   RotateLeft - performs the rotate left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

RotateLeft
PROCEDURE RotateLeft (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      RotateCount: CARDINAL) ;

(*
   RotateRight - performs the rotate right for a multi word set.
                 This procedure might be called by the back end of
                 GNU Modula-2 depending whether amount is known at
                 compile time.
*)

RotateRight
PROCEDURE RotateRight (VAR s, d: ARRAY OF BITSET;
                       SetSizeInBits: CARDINAL;
                       RotateCount: CARDINAL) ;

END SYSTEM.
```

18.1.36 gm2-libs/Scan

```
DEFINITION MODULE Scan ;

(* Provides a primitive symbol fetching from input.
   Symbols are delimited by spaces and tabs.
   Limitation only allows one source file at
   a time to deliver symbols.  *)

EXPORT QUALIFIED GetNextSymbol, WriteError,
                 OpenSource, CloseSource,
                 TerminateOnError, DefineComments ;

(* OpenSource - opens a source file for reading.                  *)

OpenSource
PROCEDURE OpenSource (a: ARRAY OF CHAR) : BOOLEAN ;

(* CloseSource - closes the current source file from reading.     *)

CloseSource
PROCEDURE CloseSource ;

(* GetNextSymbol gets the next source symbol and returns it in a. *)
```

(continues on next page)

(continued from previous page)

```
GetNextSymbol
PROCEDURE GetNextSymbol (VAR a: ARRAY OF CHAR) ;

(* WriteError writes a message, a, under the source line, which   *)
(* attempts to pinpoint the Symbol at fault.                      *)

WriteError
PROCEDURE WriteError (a: ARRAY OF CHAR) ;

(*
   TerminateOnError - exits with status 1 if we call WriteError.
*)

TerminateOnError
PROCEDURE TerminateOnError ;

(*
   DefineComments - defines the start of comments within the source
                    file.

                    The characters in Start define the comment start
                    and characters in End define the end.
                    The BOOLEAN eoln determine whether the comment
                    is terminated by end of line. If eoln is TRUE
                    then End is ignored.

                    If this procedure is never called then no comments
                    are allowed.
*)

DefineComments
PROCEDURE DefineComments (Start, End: ARRAY OF CHAR; eoln: BOOLEAN) ;

END Scan.
```

18.1.37 gm2-libs/Selective

```
DEFINITION MODULE Selective ;

FROM SYSTEM IMPORT ADDRESS ;

EXPORT QUALIFIED SetOfFd, Timeval,
                 InitSet, KillSet, InitTime, KillTime,
                 GetTime, SetTime,
                 FdZero, FdSet, FdClr, FdIsSet, Select,
                 MaxFdsPlusOne, WriteCharRaw, ReadCharRaw,
                 GetTimeOfDay ;

TYPE
SetOfFd (type)
```

(continues on next page)

(continued from previous page)

```
   SetOfFd = ADDRESS ;    (* Hidden type in Selective.c *)
Timeval (type)
   Timeval = ADDRESS ;    (* Hidden type in Selective.c *)

Select
PROCEDURE Select (nooffds: CARDINAL;
                  readfds, writefds, exceptfds: SetOfFd;
                  timeout: Timeval) : INTEGER ;

InitTime
PROCEDURE InitTime (sec, usec: CARDINAL) : Timeval ;
KillTime
PROCEDURE KillTime (t: Timeval) : Timeval ;
GetTime
PROCEDURE GetTime (t: Timeval; VAR sec, usec: CARDINAL) ;
SetTime
PROCEDURE SetTime (t: Timeval; sec, usec: CARDINAL) ;
InitSet
PROCEDURE InitSet () : SetOfFd ;
KillSet
PROCEDURE KillSet (s: SetOfFd) : SetOfFd ;
FdZero
PROCEDURE FdZero (s: SetOfFd) ;
FdSet
PROCEDURE FdSet (fd: INTEGER; s: SetOfFd) ;
FdClr
PROCEDURE FdClr (fd: INTEGER; s: SetOfFd) ;
FdIsSet
PROCEDURE FdIsSet (fd: INTEGER; s: SetOfFd) : BOOLEAN ;
MaxFdsPlusOne
PROCEDURE MaxFdsPlusOne (a, b: INTEGER) : INTEGER ;

(* you must use the raw routines with select - not the FIO buffered routines *)
WriteCharRaw
PROCEDURE WriteCharRaw (fd: INTEGER; ch: CHAR) ;
ReadCharRaw
PROCEDURE ReadCharRaw (fd: INTEGER) : CHAR ;

(*
   GetTimeOfDay - fills in a record, Timeval, filled in with the
                  current system time in seconds and microseconds.
                  It returns zero (see man 3p gettimeofday)
*)

GetTimeOfDay
PROCEDURE GetTimeOfDay (tv: Timeval) : INTEGER ;

END Selective.
```

18.1.38 gm2-libs/StdIO

```
DEFINITION MODULE StdIO ;

EXPORT QUALIFIED ProcRead, ProcWrite,
                 Read, Write,
                 PushOutput, PopOutput, GetCurrentOutput,
                 PushInput, PopInput, GetCurrentInput ;

TYPE
ProcWrite (type)
   ProcWrite = PROCEDURE (CHAR) ;
ProcRead (type)
   ProcRead  = PROCEDURE (VAR CHAR) ;

(*
   Read - is the generic procedure that all higher application layers
          should use to receive a character.
*)

Read
PROCEDURE Read (VAR ch: CHAR) ;

(*
   Write - is the generic procedure that all higher application layers
           should use to emit a character.
*)

Write
PROCEDURE Write (ch: CHAR) ;

(*
   PushOutput - pushes the current Write procedure onto a stack,
                any future references to Write will actually invoke
                procedure, p.
*)

PushOutput
PROCEDURE PushOutput (p: ProcWrite) ;

(*
   PopOutput - restores Write to use the previous output procedure.
*)

PopOutput
PROCEDURE PopOutput ;

(*
   GetCurrentOutput - returns the current output procedure.
*)

GetCurrentOutput
```

(continues on next page)

(continued from previous page)

```
PROCEDURE GetCurrentOutput () : ProcWrite ;

(*
   PushInput - pushes the current Read procedure onto a stack,
               any future references to Read will actually invoke
               procedure, p.
*)

PushInput
PROCEDURE PushInput (p: ProcRead) ;

(*
   PopInput - restores Write to use the previous output procedure.
*)

PopInput
PROCEDURE PopInput ;

(*
   GetCurrentInput - returns the current input procedure.
*)

GetCurrentInput
PROCEDURE GetCurrentInput () : ProcRead ;

END StdIO.
```

18.1.39 gm2-libs/Storage

```
DEFINITION MODULE Storage ;

FROM SYSTEM IMPORT ADDRESS ;

EXPORT QUALIFIED ALLOCATE, DEALLOCATE, REALLOCATE, Available ;

(*
   ALLOCATE - attempt to allocate memory from the heap.
              NIL is returned in, a, if ALLOCATE fails.
*)

ALLOCATE
PROCEDURE ALLOCATE (VAR a: ADDRESS ; Size: CARDINAL) ;

(*
   DEALLOCATE - return, Size, bytes to the heap.
                The variable, a, is set to NIL.
*)

DEALLOCATE
PROCEDURE DEALLOCATE (VAR a: ADDRESS ; Size: CARDINAL) ;
```

(continues on next page)

(continued from previous page)

```
(*
   REALLOCATE - attempts to reallocate storage. The address,
                a, should either be NIL in which case ALLOCATE
                is called, or alternatively it should have already
                been initialized by ALLOCATE. The allocated storage
                is resized accordingly.
*)

REALLOCATE
PROCEDURE REALLOCATE (VAR a: ADDRESS; Size: CARDINAL) ;

(*
   Available - returns TRUE if, Size, bytes can be allocated.
*)

Available
PROCEDURE Available (Size: CARDINAL) : BOOLEAN ;

END Storage.
```

18.1.40 gm2-libs/StrCase

```
DEFINITION MODULE StrCase ;

EXPORT QUALIFIED StrToUpperCase, StrToLowerCase, Cap, Lower ;

(*
   StrToUpperCase - converts string, a, to uppercase returning the
                    result in, b.
*)

StrToUpperCase
PROCEDURE StrToUpperCase (a: ARRAY OF CHAR ; VAR b: ARRAY OF CHAR) ;

(*
   StrToLowerCase - converts string, a, to lowercase returning the
                    result in, b.
*)

StrToLowerCase
PROCEDURE StrToLowerCase (a: ARRAY OF CHAR ; VAR b: ARRAY OF CHAR) ;

(*
   Cap - converts a lower case character into a capital character.
         If the character is not a lower case character 'a'..'z'
         then the character is simply returned unaltered.
*)

Cap
```

(continues on next page)

(continued from previous page)

```
PROCEDURE Cap (ch: CHAR) : CHAR ;

(*
   Lower - converts an upper case character into a lower case character.
           If the character is not an upper case character 'A'..'Z'
           then the character is simply returned unaltered.
*)

Lower
PROCEDURE Lower (ch: CHAR) : CHAR ;

END StrCase.
```

18.1.41 gm2-libs/StrIO

```
DEFINITION MODULE StrIO ;

EXPORT QUALIFIED ReadString, WriteString,
                 WriteLn ;

(*
   WriteLn - writes a carriage return and a newline
             character.
*)

WriteLn
PROCEDURE WriteLn ;

(*
   ReadString - reads a sequence of characters into a string.
                Line editing accepts Del, Ctrl H, Ctrl W and
                Ctrl U.
*)

ReadString
PROCEDURE ReadString (VAR a: ARRAY OF CHAR) ;

(*
   WriteString - writes a string to the default output.
*)

WriteString
PROCEDURE WriteString (a: ARRAY OF CHAR) ;

END StrIO.
```

18.1.42 gm2-libs/StrLib

```
DEFINITION MODULE StrLib ;

EXPORT QUALIFIED StrConCat, StrLen, StrCopy, StrEqual, StrLess,
                       IsSubString, StrRemoveWhitePrefix ;

(*
   StrConCat - combines a and b into c.
*)

StrConCat
PROCEDURE StrConCat (a, b: ARRAY OF CHAR; VAR c: ARRAY OF CHAR) ;

(*
   StrLess - returns TRUE if string, a, alphabetically occurs before
             string, b.
*)

StrLess
PROCEDURE StrLess (a, b: ARRAY OF CHAR) : BOOLEAN ;

(*
   StrEqual - performs a = b on two strings.
*)

StrEqual
PROCEDURE StrEqual (a, b: ARRAY OF CHAR) : BOOLEAN ;

(*
   StrLen - returns the length of string, a.
*)

StrLen
PROCEDURE StrLen (a: ARRAY OF CHAR) : CARDINAL ;

(*
   StrCopy - copy string src into string dest providing dest is large enough.
             If dest is smaller than a then src then the string is truncated when
             dest is full.  Add a nul character if there is room in dest.
*)

StrCopy
PROCEDURE StrCopy (src: ARRAY OF CHAR ; VAR dest: ARRAY OF CHAR) ;

(*
   IsSubString - returns true if b is a subcomponent of a.
*)

IsSubString
PROCEDURE IsSubString (a, b: ARRAY OF CHAR) : BOOLEAN ;
```

(continues on next page)

(continued from previous page)

```
(*
   StrRemoveWhitePrefix - copies string, into string, b, excluding any white
                          space infront of a.
*)

StrRemoveWhitePrefix
PROCEDURE StrRemoveWhitePrefix (a: ARRAY OF CHAR; VAR b: ARRAY OF CHAR) ;

END StrLib.
```

18.1.43 gm2-libs/StringConvert

```
DEFINITION MODULE StringConvert ;

FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED IntegerToString, StringToInteger,
                 StringToLongInteger, LongIntegerToString,
                 StringToCardinal, CardinalToString,
                 StringToLongCardinal, LongCardinalToString,
                 StringToShortCardinal, ShortCardinalToString,
                 StringToLongreal, LongrealToString,
                 ToSigFig,
                 stoi, itos, ctos, stoc, hstoi, ostoi, bstoi,
                 hstoc, ostoc, bstoc,
                 stor, stolr ;

(*
   IntegerToString - converts INTEGER, i, into a String. The field with
                     can be specified if non zero. Leading characters
                     are defined by padding and this function will
                     prepend a + if sign is set to TRUE.
                     The base allows the caller to generate binary,
                     octal, decimal, hexidecimal numbers.
                     The value of lower is only used when hexidecimal
                     numbers are generated and if TRUE then digits
                     abcdef are used, and if FALSE then ABCDEF are used.
*)

IntegerToString
PROCEDURE IntegerToString (i: INTEGER; width: CARDINAL; padding: CHAR; sign: BOOLEAN;
                           base: CARDINAL; lower: BOOLEAN) : String ;

(*
   CardinalToString - converts CARDINAL, c, into a String. The field
                      width can be specified if non zero. Leading
                      characters are defined by padding.
                      The base allows the caller to generate binary,
                      octal, decimal, hexidecimal numbers.
                      The value of lower is only used when hexidecimal
                      numbers are generated and if TRUE then digits
```

(continues on next page)

(continued from previous page)

```
                     abcdef are used, and if FALSE then ABCDEF are used.
*)

CardinalToString
PROCEDURE CardinalToString (c: CARDINAL; width: CARDINAL; padding: CHAR;
                            base: CARDINAL; lower: BOOLEAN) : String ;

(*
   StringToInteger - converts a string, s, of, base, into an INTEGER.
                     Leading white space is ignored. It stops converting
                     when either the string is exhausted or if an illegal
                     numeral is found.
                     The parameter found is set TRUE if a number was found.
*)

StringToInteger
PROCEDURE StringToInteger (s: String; base: CARDINAL; VAR found: BOOLEAN) : INTEGER ;

(*
   StringToCardinal - converts a string, s, of, base, into a CARDINAL.
                      Leading white space is ignored. It stops converting
                      when either the string is exhausted or if an illegal
                      numeral is found.
                      The parameter found is set TRUE if a number was found.
*)

StringToCardinal
PROCEDURE StringToCardinal (s: String; base: CARDINAL; VAR found: BOOLEAN) : CARDINAL ;

(*
   LongIntegerToString - converts LONGINT, i, into a String. The field with
                         can be specified if non zero. Leading characters
                         are defined by padding and this function will
                         prepend a + if sign is set to TRUE.
                         The base allows the caller to generate binary,
                         octal, decimal, hexidecimal numbers.
                         The value of lower is only used when hexidecimal
                         numbers are generated and if TRUE then digits
                         abcdef are used, and if FALSE then ABCDEF are used.
*)

LongIntegerToString
PROCEDURE LongIntegerToString (i: LONGINT; width: CARDINAL; padding: CHAR;
                               sign: BOOLEAN; base: CARDINAL; lower: BOOLEAN) : String ;

(*
   StringToLongInteger - converts a string, s, of, base, into an LONGINT.
                         Leading white space is ignored. It stops converting
                         when either the string is exhausted or if an illegal
                         numeral is found.
                         The parameter found is set TRUE if a number was found.
```

(continues on next page)

(continued from previous page)

```
*)

StringToLongInteger
PROCEDURE StringToLongInteger (s: String; base: CARDINAL; VAR found: BOOLEAN) : LONGINT ;

(*
   LongCardinalToString - converts LONGCARD, c, into a String. The field
                          width can be specified if non zero. Leading
                          characters are defined by padding.
                          The base allows the caller to generate binary,
                          octal, decimal, hexidecimal numbers.
                          The value of lower is only used when hexidecimal
                          numbers are generated and if TRUE then digits
                          abcdef are used, and if FALSE then ABCDEF are used.
*)

LongCardinalToString
PROCEDURE LongCardinalToString (c: LONGCARD; width: CARDINAL; padding: CHAR;
                                base: CARDINAL; lower: BOOLEAN) : String ;

(*
   StringToLongCardinal - converts a string, s, of, base, into a LONGCARD.
                          Leading white space is ignored. It stops converting
                          when either the string is exhausted or if an illegal
                          numeral is found.
                          The parameter found is set TRUE if a number was found.
*)

StringToLongCardinal
PROCEDURE StringToLongCardinal (s: String; base: CARDINAL; VAR found: BOOLEAN) : LONGCARD ;

(*
   ShortCardinalToString - converts SHORTCARD, c, into a String. The field
                           width can be specified if non zero. Leading
                           characters are defined by padding.
                           The base allows the caller to generate binary,
                           octal, decimal, hexidecimal numbers.
                           The value of lower is only used when hexidecimal
                           numbers are generated and if TRUE then digits
                           abcdef are used, and if FALSE then ABCDEF are used.
*)

ShortCardinalToString
PROCEDURE ShortCardinalToString (c: SHORTCARD; width: CARDINAL; padding: CHAR;
                                 base: CARDINAL; lower: BOOLEAN) : String ;

(*
   StringToShortCardinal - converts a string, s, of, base, into a SHORTCARD.
                           Leading white space is ignored. It stops converting
                           when either the string is exhausted or if an illegal
                           numeral is found.
```

(continues on next page)

(continued from previous page)

```
                          The parameter found is set TRUE if a number was found.
*)

StringToShortCardinal
PROCEDURE StringToShortCardinal (s: String; base: CARDINAL;
                                 VAR found: BOOLEAN) : SHORTCARD ;

(*
   stoi - decimal string to INTEGER
*)

stoi
PROCEDURE stoi (s: String) : INTEGER ;

(*
   itos - integer to decimal string.
*)

itos
PROCEDURE itos (i: INTEGER; width: CARDINAL; padding: CHAR; sign: BOOLEAN) : String ;

(*
   ctos - cardinal to decimal string.
*)

ctos
PROCEDURE ctos (c: CARDINAL; width: CARDINAL; padding: CHAR) : String ;

(*
   stoc - decimal string to CARDINAL
*)

stoc
PROCEDURE stoc (s: String) : CARDINAL ;

(*
   hstoi - hexidecimal string to INTEGER
*)

hstoi
PROCEDURE hstoi (s: String) : INTEGER ;

(*
   ostoi - octal string to INTEGER
*)

ostoi
PROCEDURE ostoi (s: String) : INTEGER ;

(*
   bstoi - binary string to INTEGER
```

(continues on next page)

(continued from previous page)

```
*)

bstoi
PROCEDURE bstoi (s: String) : INTEGER ;

(*
   hstoc - hexidecimal string to CARDINAL
*)

hstoc
PROCEDURE hstoc (s: String) : CARDINAL ;

(*
   ostoc - octal string to CARDINAL
*)

ostoc
PROCEDURE ostoc (s: String) : CARDINAL ;

(*
   bstoc - binary string to CARDINAL
*)

bstoc
PROCEDURE bstoc (s: String) : CARDINAL ;

(*
   StringToLongreal - returns a LONGREAL and sets found to TRUE
                      if a legal number is seen.
*)

StringToLongreal
PROCEDURE StringToLongreal (s: String; VAR found: BOOLEAN) : LONGREAL ;

(*
   LongrealToString - converts a LONGREAL number, Real, which has,
                      TotalWidth, and FractionWidth into a string.

                      So for example:

                      LongrealToString(1.0, 4, 2)  -> '1.00'
                      LongrealToString(12.3, 5, 2) -> '12.30'
                      LongrealToString(12.3, 6, 2) -> ' 12.30'
                      LongrealToString(12.3, 6, 3) -> '12.300'

                      if total width is too small then the fraction
                      becomes truncated.

                      LongrealToString(12.3, 5, 3) -> '12.30'

                      If TotalWidth is 0 then the function
```

(continues on next page)

(continued from previous page)

```
                    will return the value of x which is converted
                    into as a fixed point number with exhaustive
                    precision.
*)

LongrealToString
PROCEDURE LongrealToString (x: LONGREAL;
                            TotalWidth, FractionWidth: CARDINAL) : String ;

(*
   stor - returns a REAL given a string.
*)

stor
PROCEDURE stor (s: String) : REAL ;

(*
   stolr - returns a LONGREAL given a string.
*)

stolr
PROCEDURE stolr (s: String) : LONGREAL ;

(*
   ToSigFig - returns a floating point or base 10 integer
              string which is accurate to, n, significant
              figures.  It will return a new String
              and, s, will be destroyed.

              So:  12.345

              rounded to the following significant figures yields

              5      12.345
              4      12.34
              3      12.3
              2      12
              1      10
*)

ToSigFig
PROCEDURE ToSigFig (s: String; n: CARDINAL) : String ;

(*
   ToDecimalPlaces - returns a floating point or base 10 integer
                     string which is accurate to, n, decimal
                     places.  It will return a new String
                     and, s, will be destroyed.
                     Decimal places yields, n, digits after
                     the .
```

(continues on next page)

(continued from previous page)

```
                    So:  12.345

                    rounded to the following decimal places yields

                    5      12.34500
                    4      12.3450
                    3      12.345
                    2      12.34
                    1      12.3
*)

ToDecimalPlaces
PROCEDURE ToDecimalPlaces (s: String; n: CARDINAL) : String ;

END StringConvert.
```

18.1.44 gm2-libs/SysExceptions

```
DEFINITION MODULE SysExceptions ;

(* Provides a mechanism for the underlying libraries to
   configure the exception routines.  This mechanism
   is used by both the ISO and PIM libraries.
   It is written to be ISO compliant and this also
   allows for mixed dialect projects.  *)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
PROCEXCEPTION (type)
   PROCEXCEPTION = PROCEDURE (ADDRESS) ;

InitExceptionHandlers
PROCEDURE InitExceptionHandlers (indexf, range, casef, invalidloc,
                                 function, wholevalue, wholediv,
                                 realvalue, realdiv, complexvalue,
                                 complexdiv, protection, systemf,
                                 coroutine, exception: PROCEXCEPTION) ;

END SysExceptions.
```

18.1.45 gm2-libs/SysStorage

```
DEFINITION MODULE SysStorage ;

(*  Provides dynamic allocation for the system components.
    This allows the application to use the traditional Storage module
    which can be handled differently.  *)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED ALLOCATE, DEALLOCATE, REALLOCATE, Available, Init ;

(*
   ALLOCATE - attempt to allocate memory from the heap.
              NIL is returned in, a, if ALLOCATE fails.
*)

ALLOCATE
PROCEDURE ALLOCATE (VAR a: ADDRESS ; size: CARDINAL) ;

(*
   DEALLOCATE - return, size, bytes to the heap.
                The variable, a, is set to NIL.
*)

DEALLOCATE
PROCEDURE DEALLOCATE (VAR a: ADDRESS ; size: CARDINAL) ;

(*
   REALLOCATE - attempts to reallocate storage. The address,
                a, should either be NIL in which case ALLOCATE
                is called, or alternatively it should have already
                been initialized by ALLOCATE. The allocated storage
                is resized accordingly.
*)

REALLOCATE
PROCEDURE REALLOCATE (VAR a: ADDRESS; size: CARDINAL) ;

(*
   Available - returns TRUE if, size, bytes can be allocated.
*)

Available
PROCEDURE Available (size: CARDINAL) : BOOLEAN;

(*
   Init - initializes the heap.
          This does nothing on a GNU/Linux system.
          But it remains here since it might be used in an
          embedded system.
*)
```

(continues on next page)

(continued from previous page)

```
Init
PROCEDURE Init ;

END SysStorage.
```

18.1.46 gm2-libs/TimeString

```
DEFINITION MODULE TimeString ;

EXPORT QUALIFIED GetTimeString ;

(*
   GetTimeString - places the time in ascii format into array, a.

*)

GetTimeString
PROCEDURE GetTimeString (VAR a: ARRAY OF CHAR) ;

END TimeString.
```

18.1.47 gm2-libs/UnixArgs

```
DEFINITION MODULE UnixArgs ;

FROM SYSTEM IMPORT ADDRESS ;

EXPORT QUALIFIED GetArgC, GetArgV, GetEnvV ;

GetArgC
PROCEDURE GetArgC () : INTEGER ;
GetArgV
PROCEDURE GetArgV () : ADDRESS ;
GetEnvV
PROCEDURE GetEnvV () : ADDRESS ;

END UnixArgs.
```

18.1.48 gm2-libs/cbuiltin

```
DEFINITION MODULE FOR "C" cbuiltin ;

FROM SYSTEM IMPORT ADDRESS ;
EXPORT UNQUALIFIED alloca, memcpy,
                   isfinite, isfinitef, isfinitel,
                   isinf_sign, isinf_signf, isinf_signl,
                     sinf, sinl, sin,
                     cosf, cosl, cos,
                     atan2f, atan2l, atan2,
                     sqrtf, sqrtl, sqrt,
                     fabsf, fabsl, fabs,
                     logf, logl, log,
                     expf, expl, exp,
                     log10f, log10l, log10,
                     exp10f, exp10l, exp10,
                     ilogbf, ilogbl, ilogb,
                     significand, significandf, significandl,
                     modf, modff, modfl,
                     nextafter, nextafterf, nextafterl,
                     nexttoward, nexttowardf, nexttowardl,
                     scalb, scalbf, scalbl,
                     scalbn, scalbnf, scalbnl,
                     scalbln, scalblnf, scalblnl,

                     cabsf, cabsl, cabs,
                     cargf, carg, cargl,
                     conjf, conj, conjl,
                     cpowf, cpow, cpowl,
                     csqrtf, csqrt, csqrtl,
                     cexpf, cexp, cexpl,
                     clogf, clog, clogl,
                     csinf, csin, csinl,
                     ccosf, ccos, ccosl,
                     ctanf, ctan, ctanl,
                     casinf, casin, casinl,
                     cacosf, cacos, cacosl,
                     catanf, catan, catanl,

                     index, rindex,
                     memcmp, memset, memmove,
                     strcat, strncat, strcpy, strncpy, strcmp, strncmp,
                     strlen, strstr, strpbrk, strspn, strcspn, strchr, strrchr ;

alloca
PROCEDURE alloca (i: CARDINAL) : ADDRESS ;
memcpy
PROCEDURE memcpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
isfinite
PROCEDURE isfinite (x: REAL) : BOOLEAN ;
isfinitel
```

(continues on next page)

(continued from previous page)

```
PROCEDURE isfinitel (x: LONGREAL) : BOOLEAN ;
isfinitef
PROCEDURE isfinitef (x: SHORTREAL) : BOOLEAN ;
isinf_sign
PROCEDURE isinf_sign (x: REAL) : BOOLEAN ;
isinf_signl
PROCEDURE isinf_signl (x: LONGREAL) : BOOLEAN ;
isinf_signf
PROCEDURE isinf_signf (x: SHORTREAL) : BOOLEAN ;
sinf
PROCEDURE sinf (x: SHORTREAL) : SHORTREAL ;
sin
PROCEDURE sin (x: REAL) : REAL ;
sinl
PROCEDURE sinl (x: LONGREAL) : LONGREAL ;
cosf
PROCEDURE cosf (x: SHORTREAL) : SHORTREAL ;
cos
PROCEDURE cos (x: REAL) : REAL ;
cosl
PROCEDURE cosl (x: LONGREAL) : LONGREAL ;
atan2f
PROCEDURE atan2f (x, y: SHORTREAL) : SHORTREAL ;
atan2
PROCEDURE atan2 (x, y: REAL) : REAL ;
atan2l
PROCEDURE atan2l (x, y: LONGREAL) : LONGREAL ;
sqrtf
PROCEDURE sqrtf (x: SHORTREAL) : SHORTREAL ;
sqrt
PROCEDURE sqrt (x: REAL) : REAL ;
sqrtl
PROCEDURE sqrtl (x: LONGREAL) : LONGREAL ;
fabsf
PROCEDURE fabsf (x: SHORTREAL) : SHORTREAL ;
fabs
PROCEDURE fabs (x: REAL) : REAL ;
fabsl
PROCEDURE fabsl (x: LONGREAL) : LONGREAL ;
logf
PROCEDURE logf (x: SHORTREAL) : SHORTREAL ;
log
PROCEDURE log (x: REAL) : REAL ;
logl
PROCEDURE logl (x: LONGREAL) : LONGREAL ;
expf
PROCEDURE expf (x: SHORTREAL) : SHORTREAL ;
exp
PROCEDURE exp (x: REAL) : REAL ;
expl
PROCEDURE expl (x: LONGREAL) : LONGREAL ;
```

(continues on next page)

(continued from previous page)

```
log10f
PROCEDURE log10f (x: SHORTREAL) : SHORTREAL ;
log10
PROCEDURE log10 (x: REAL) : REAL ;
log10l
PROCEDURE log10l (x: LONGREAL) : LONGREAL ;
exp10f
PROCEDURE exp10f (x: SHORTREAL) : SHORTREAL ;
exp10
PROCEDURE exp10 (x: REAL) : REAL ;
exp10l
PROCEDURE exp10l (x: LONGREAL) : LONGREAL ;
ilogbf
PROCEDURE ilogbf (x: SHORTREAL) : INTEGER ;
ilogb
PROCEDURE ilogb (x: REAL) : INTEGER ;
ilogbl
PROCEDURE ilogbl (x: LONGREAL) : INTEGER ;

significand
PROCEDURE significand (r: REAL) : REAL ;
significandf
PROCEDURE significandf (s: SHORTREAL) : SHORTREAL ;
significandl
PROCEDURE significandl (l: LONGREAL) : LONGREAL ;

modf
PROCEDURE modf (x: REAL; VAR y: REAL) : REAL ;
modff
PROCEDURE modff (x: SHORTREAL; VAR y: SHORTREAL) : SHORTREAL ;
modfl
PROCEDURE modfl (x: LONGREAL; VAR y: LONGREAL) : LONGREAL ;

nextafter
PROCEDURE nextafter (x, y: REAL) : REAL ;
nextafterf
PROCEDURE nextafterf (x, y: SHORTREAL) : SHORTREAL ;
nextafterl
PROCEDURE nextafterl (x, y: LONGREAL) : LONGREAL ;

nexttoward
PROCEDURE nexttoward (x, y: REAL) : REAL ;
nexttowardf
PROCEDURE nexttowardf (x, y: SHORTREAL) : SHORTREAL ;
nexttowardl
PROCEDURE nexttowardl (x, y: LONGREAL) : LONGREAL ;

scalb
PROCEDURE scalb (x, n: REAL) : REAL ;
scalbf
PROCEDURE scalbf (x, n: SHORTREAL) : SHORTREAL ;
```

(continues on next page)

(continued from previous page)

```
scalbl
PROCEDURE scalbl (x, n: LONGREAL) : LONGREAL ;

scalbn
PROCEDURE scalbn (x: REAL; n: INTEGER) : REAL ;
scalbnf
PROCEDURE scalbnf (x: SHORTREAL; n: INTEGER) : SHORTREAL ;
scalbnl
PROCEDURE scalbnl (x: LONGREAL; n: INTEGER) : LONGREAL ;

scalbln
PROCEDURE scalbln (x: REAL; n: LONGINT) : REAL ;
scalblnf
PROCEDURE scalblnf (x: SHORTREAL; n: LONGINT) : SHORTREAL ;
scalblnl
PROCEDURE scalblnl (x: LONGREAL; n: LONGINT) : LONGREAL ;

cabsf
PROCEDURE cabsf (z: SHORTCOMPLEX) : SHORTREAL ;
cabs
PROCEDURE cabs (z: COMPLEX) : REAL ;
cabsl
PROCEDURE cabsl (z: LONGCOMPLEX) : LONGREAL ;

cargf
PROCEDURE cargf (z: SHORTCOMPLEX) : SHORTREAL ;
carg
PROCEDURE carg (z: COMPLEX) : REAL ;
cargl
PROCEDURE cargl (z: LONGCOMPLEX) : LONGREAL ;

conjf
PROCEDURE conjf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
conj
PROCEDURE conj (z: COMPLEX) : COMPLEX ;
conjl
PROCEDURE conjl (z: LONGCOMPLEX) : LONGCOMPLEX ;

cpowf
PROCEDURE cpowf (base: SHORTCOMPLEX; exp: SHORTREAL) : SHORTCOMPLEX ;
cpow
PROCEDURE cpow (base: COMPLEX; exp: REAL) : COMPLEX ;
cpowl
PROCEDURE cpowl (base: LONGCOMPLEX; exp: LONGREAL) : LONGCOMPLEX ;

csqrtf
PROCEDURE csqrtf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
csqrt
PROCEDURE csqrt (z: COMPLEX) : COMPLEX ;
csqrtl
PROCEDURE csqrtl (z: LONGCOMPLEX) : LONGCOMPLEX ;
```

(continues on next page)

(continued from previous page)

```
cexpf
PROCEDURE cexpf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
cexp
PROCEDURE cexp (z: COMPLEX) : COMPLEX ;
cexpl
PROCEDURE cexpl (z: LONGCOMPLEX) : LONGCOMPLEX ;

clogf
PROCEDURE clogf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
clog
PROCEDURE clog (z: COMPLEX) : COMPLEX ;
clogl
PROCEDURE clogl (z: LONGCOMPLEX) : LONGCOMPLEX ;

csinf
PROCEDURE csinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
csin
PROCEDURE csin (z: COMPLEX) : COMPLEX ;
csinl
PROCEDURE csinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

ccosf
PROCEDURE ccosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
ccos
PROCEDURE ccos (z: COMPLEX) : COMPLEX ;
ccosl
PROCEDURE ccosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

ctanf
PROCEDURE ctanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
ctan
PROCEDURE ctan (z: COMPLEX) : COMPLEX ;
ctanl
PROCEDURE ctanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

casinf
PROCEDURE casinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
casin
PROCEDURE casin (z: COMPLEX) : COMPLEX ;
casinl
PROCEDURE casinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

cacosf
PROCEDURE cacosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
cacos
PROCEDURE cacos (z: COMPLEX) : COMPLEX ;
cacosl
PROCEDURE cacosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

catanf
```

(continues on next page)

(continued from previous page)

```
PROCEDURE catanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
catan
PROCEDURE catan (z: COMPLEX) : COMPLEX ;
catanl
PROCEDURE catanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

index
PROCEDURE index (s: ADDRESS; c: INTEGER) : ADDRESS ;
rindex
PROCEDURE rindex (s: ADDRESS; c: INTEGER) : ADDRESS ;
memcmp
PROCEDURE memcmp (s1, s2: ADDRESS; n: CARDINAL) : INTEGER ;
memmove
PROCEDURE memmove (s1, s2: ADDRESS; n: CARDINAL) : ADDRESS ;
memset
PROCEDURE memset (s: ADDRESS; c: INTEGER; n: CARDINAL) : ADDRESS ;
strcat
PROCEDURE strcat (dest, src: ADDRESS) : ADDRESS ;
strncat
PROCEDURE strncat (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
strcpy
PROCEDURE strcpy (dest, src: ADDRESS) : ADDRESS ;
strncpy
PROCEDURE strncpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
strcmp
PROCEDURE strcmp (s1, s2: ADDRESS) : INTEGER ;
strncmp
PROCEDURE strncmp (s1, s2: ADDRESS; n: CARDINAL) : INTEGER ;
strlen
PROCEDURE strlen (s: ADDRESS) : INTEGER ;
strstr
PROCEDURE strstr (haystack, needle: ADDRESS) : ADDRESS ;
strpbrk
PROCEDURE strpbrk (s, accept: ADDRESS) : ADDRESS ;
strspn
PROCEDURE strspn (s, accept: ADDRESS) : CARDINAL ;
strcspn
PROCEDURE strcspn (s, accept: ADDRESS) : CARDINAL ;
strchr
PROCEDURE strchr (s: ADDRESS; c: INTEGER) : ADDRESS ;
strrchr
PROCEDURE strrchr (s: ADDRESS; c: INTEGER) : ADDRESS ;

END cbuiltin.
```

18.1.49 gm2-libs/cgetopt

```
DEFINITION MODULE cgetopt ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
Options (type)
   Options = ADDRESS ;

VAR
optarg                 (var)
   optarg                : ADDRESS ;
optind (var)
opterr (var)
optopt (var)
   optind, opterr, optopt: INTEGER ;

(*
   getopt - the getopt() function parses the command-line arguments.
            Its arguments argc and argv are the argument count and array as
            passed to the main() function on program invocation.  An element of
            argv that starts with '-' (and is not exactly "-" or "--") is an
            option element.  The characters of this element (aside from the
            initial '-') are option characters.  If getopt() is called
            repeatedly, it returns successively each of the option characters
            from each of the option elements.
*)

getopt
PROCEDURE getopt (argc: INTEGER; argv: ADDRESS; optstring: ADDRESS) : CHAR ;

(*
   getopt_long - works like getopt() except that it also accepts long options,
                 started with two dashes.  (If the program accepts only long
                 options, then optstring should be specified as an empty string (""),
                 not NULL.)  Long option names may be abbreviated if the abbreviation
                 is unique or is an exact match for some defined option.  A
                 long option may take a parameter, of the form --arg=param or
                 --arg param.
*)

getopt_long
PROCEDURE getopt_long (argc: INTEGER; argv: ADDRESS; optstring: ADDRESS;
                       longopts: ADDRESS; VAR longindex: INTEGER) : INTEGER ;

(*
   getopt_long_only - a wrapper for the C getopt_long_only.
*)

getopt_long_only
PROCEDURE getopt_long_only (argc: INTEGER; argv: ADDRESS; optstring: ADDRESS;
```

(continues on next page)

(continued from previous page)

```
                          longopts: ADDRESS; VAR longindex: INTEGER) : INTEGER ;

(*
   InitOptions - constructor for empty Options.
*)

InitOptions
PROCEDURE InitOptions () : Options ;

(*
   KillOptions - deconstructor for empty Options.
*)

KillOptions
PROCEDURE KillOptions (o: Options) : Options ;

(*
   SetOption - set option[index] with {name, has_arg, flag, val}.
*)

SetOption
PROCEDURE SetOption (o: Options; index: CARDINAL;
                     name: ADDRESS; has_arg: BOOLEAN;
                     VAR flag: INTEGER; val: INTEGER) ;

(*
   GetLongOptionArray - return a pointer to the C array containing all
                        long options.
*)

GetLongOptionArray
PROCEDURE GetLongOptionArray (o: Options) : ADDRESS ;

END cgetopt.
```

18.1.50 gm2-libs/cxxabi

```
DEFINITION MODULE FOR "C" cxxabi ;

(* This should only be used by the compiler and it matches the
    g++ implementation.  *)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT UNQUALIFIED __cxa_begin_catch, __cxa_end_catch, __cxa_rethrow ;

__cxa_begin_catch
PROCEDURE __cxa_begin_catch (a: ADDRESS) : ADDRESS ;
__cxa_end_catch
PROCEDURE __cxa_end_catch ;
__cxa_rethrow
```

(continues on next page)

(continued from previous page)

```
PROCEDURE __cxa_rethrow ;

END cxxabi.
```

18.1.51 gm2-libs/dtoa

```
DEFINITION MODULE dtoa ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
Mode (type)
   Mode = (maxsignificant, decimaldigits) ;

(*
   strtod - returns a REAL given a string, s.  It will set
            error to TRUE if the number is too large.
*)

strtod
PROCEDURE strtod (s: ADDRESS; VAR error: BOOLEAN) : REAL ;

(*
   dtoa - converts a REAL, d, into a string.  The address of the
          string is returned.
          mode       indicates the type of conversion required.
          ndigits    determines the number of digits according to mode.
          decpt      the position of the decimal point.
          sign       does the string have a sign?
*)

dtoa
PROCEDURE dtoa (d        : REAL;
                mode     : Mode;
                ndigits  : INTEGER;
              VAR decpt: INTEGER;
              VAR sign : BOOLEAN) : ADDRESS ;

END dtoa.
```

18.1.52 gm2-libs/errno

```
DEFINITION MODULE errno ;

CONST
    EINTR  =  4 ;   (* system call interrupted *)
    ERANGE = 34 ;   (* result is too large     *)
    EAGAIN = 11 ;   (* retry the system call   *)

geterrno
PROCEDURE geterrno () : INTEGER ;

END errno.
```

18.1.53 gm2-libs/gdbif

```
DEFINITION MODULE gdbif ;

(*  Provides interactive connectivity with gdb useful for debugging
    Modula-2 shared libraries.  *)

EXPORT UNQUALIFIED sleepSpin, finishSpin, connectSpin ;

(*
   finishSpin - sets boolean mustWait to FALSE.
*)

finishSpin
PROCEDURE finishSpin ;

(*
   sleepSpin - waits for the boolean variable mustWait to become FALSE.
               It sleeps for a second between each test of the variable.
*)

sleepSpin
PROCEDURE sleepSpin ;

(*
   connectSpin - breakpoint placeholder.  Its only purpose is to allow users
                 to set a breakpoint.  This procedure is called once
                 sleepSpin is released from its spin (via a call from
                 finishSpin).
*)

connectSpin
PROCEDURE connectSpin ;

END gdbif.
```

18.1.54 gm2-libs/ldtoa

```
DEFINITION MODULE ldtoa ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
Mode (type)
   Mode = (maxsignificant, decimaldigits) ;

(*
   strtold - returns a LONGREAL given a C string, s.  It will set
             error to TRUE if the number is too large or badly formed.
*)

strtold
PROCEDURE strtold (s: ADDRESS; VAR error: BOOLEAN) : LONGREAL ;

(*
   ldtoa - converts a LONGREAL, d, into a string.  The address of the
           string is returned.
           mode       indicates the type of conversion required.
           ndigits    determines the number of digits according to mode.
           decpt      the position of the decimal point.
           sign       does the string have a sign?
*)

ldtoa
PROCEDURE ldtoa (d        : LONGREAL;
                 mode     : Mode;
                 ndigits  : INTEGER;
                 VAR decpt: INTEGER;
                 VAR sign : BOOLEAN) : ADDRESS ;

END ldtoa.
```

18.1.55 gm2-libs/libc

```
DEFINITION MODULE FOR "C" libc ;

FROM SYSTEM IMPORT ADDRESS, CSIZE_T, CSSIZE_T ;

EXPORT UNQUALIFIED time_t, timeb, tm, ptrToTM,
                   write, read,
                   system, abort,
                   malloc, free,
                   exit, isatty,
                   getenv, putenv, getpid,
                   dup, close, open, lseek,
                   readv, writev,
```

(continues on next page)

(continued from previous page)

```
                  perror, creat,
                  getcwd, chown, strlen, strcpy, strncpy,
                  unlink, setenv,
                  memcpy, memset, memmove, printf, realloc,
                  rand, srand,
                  time, localtime, ftime,
                  shutdown, rename, setjmp, longjmp, atexit,
                  ttyname, sleep, execv ;

TYPE
time_t (type)
   time_t = LONGINT ;

ptrToTM (type)
   ptrToTM = POINTER TO tm ;
tm (type)
   tm = RECORD
           tm_sec: INTEGER ;     (* Seconds.     [0-60] (1 leap second) *)
           tm_min: INTEGER ;     (* Minutes.     [0-59]   *)
           tm_hour: INTEGER ;    (* Hours.       [0-23]   *)
           tm_mday: INTEGER ;    (* Day.         [1-31]   *)
           tm_mon: INTEGER ;     (* Month.       [0-11]   *)
           tm_year: INTEGER ;    (* Year - 1900.          *)
           tm_wday: INTEGER ;    (* Day of week. [0-6]    *)
           tm_yday: INTEGER ;    (* Days in year.[0-365]  *)
           tm_isdst: INTEGER ;   (* DST.         [-1/0/1] *)
           tm_gmtoff: LONGINT ;  (* Seconds east of UTC.  *)
           tm_zone: ADDRESS ;    (* char * zone name      *)
END (type)
        END ;

timeb (type)
   timeb = RECORD
              time    : time_t ;
              millitm : SHORTCARD ;
              timezone: SHORTCARD ;
              dstflag : SHORTCARD ;
END (type)
           END ;

exitP (type)
   exitP = PROCEDURE () : INTEGER ;

(*
     ssize_t write (int d, void *buf, size_t nbytes)
*)

write
PROCEDURE write (d: INTEGER; buf: ADDRESS; nbytes: CSIZE_T) : [ CSSIZE_T ] ;

(*
```

(continues on next page)

(continued from previous page)

```
     ssize_t read (int d, void *buf, size_t nbytes)
*)

read
PROCEDURE read (d: INTEGER; buf: ADDRESS; nbytes: CSIZE_T) : [ CSSIZE_T ] ;

(*
     int system(string)
     char *string;
*)

system
PROCEDURE system (a: ADDRESS) : [ INTEGER ] ;

(*
     abort - generate a fault

     abort() first closes all open files if possible, then sends
     an IOT signal to the process.  This signal usually results
     in termination with a core dump, which may be used for
     debugging.

     It is possible for abort() to return control if is caught or
     ignored, in which case the value returned is that of the
     kill(2V) system call.
*)

abort
PROCEDURE abort <* noreturn *> ;

(*
     malloc - memory allocator.

     void *malloc(size_t size);

     malloc() returns a pointer to a block of at least size
     bytes, which is appropriately aligned.  If size is zero,
     malloc() returns a non-NULL pointer, but this pointer should
     not be dereferenced.
*)

malloc
PROCEDURE malloc (size: CSIZE_T) : ADDRESS ;

(*
     free - memory deallocator.

     free (void *ptr);

     free() releases a previously allocated block.  Its argument
     is a pointer to a block previously allocated by malloc,
```

(continues on next page)

(continued from previous page)

```
     calloc, realloc, malloc, or memalign.
*)

free
PROCEDURE free (ptr: ADDRESS) ;

(*
     void *realloc (void *ptr, size_t size);

     realloc changes the size of the memory block pointed to
     by ptr to size bytes. The contents will be  unchanged  to
     the minimum of the old and new sizes; newly allocated memory
     will be uninitialized. If ptr is NIL, the call is
     equivalent  to malloc(size); if size is equal to zero, the
     call is equivalent to free(ptr). Unless ptr is NIL, it
     must have been returned by an earlier call to malloc(),
     realloc.
*)

realloc
PROCEDURE realloc (ptr: ADDRESS; size: CSIZE_T) : ADDRESS ;

(*
   isatty - does this descriptor refer to a terminal.
*)

isatty
PROCEDURE isatty (fd: INTEGER) : INTEGER ;

(*
   exit - returns control to the invoking process. Result, r, is
          returned.
*)

exit
PROCEDURE exit (r: INTEGER) <* noreturn *> ;

(*
   getenv - returns the C string for the equivalent C environment
            variable.
*)

getenv
PROCEDURE getenv (s: ADDRESS) : ADDRESS ;

(*
   putenv - change or add an environment variable.
*)

putenv
PROCEDURE putenv (s: ADDRESS) : INTEGER ;
```

(continues on next page)

(continued from previous page)

```
(*
   getpid - returns the UNIX process identification number.
*)

getpid
PROCEDURE getpid () : INTEGER ;

(*
   dup - duplicates the file descriptor, d.
*)

dup
PROCEDURE dup (d: INTEGER) : INTEGER ;

(*
   close - closes the file descriptor, d.
*)

close
PROCEDURE close (d: INTEGER) : [ INTEGER ] ;

(*
   open - open the file, filename with flag and mode.
*)

open
PROCEDURE open (filename: ADDRESS; oflag: INTEGER; ...) : INTEGER ;

(*
   creat - creates a new file
*)

creat
PROCEDURE creat (filename: ADDRESS; mode: CARDINAL) : INTEGER;

(*
   lseek - calls unix lseek:

           off_t lseek(int fildes, off_t offset, int whence);
*)

lseek
PROCEDURE lseek (fd: INTEGER; offset: LONGINT; whence: INTEGER) : LONGINT ;

(*
   perror - writes errno and string. (ARRAY OF CHAR is translated onto ADDRESS).
*)

perror
PROCEDURE perror (string: ARRAY OF CHAR);
```

(continues on next page)

(continued from previous page)

```
(*
   readv - reads an io vector of bytes.
*)

readv
PROCEDURE readv (fd: INTEGER; v: ADDRESS; n: INTEGER) : [ INTEGER ] ;

(*
   writev - writes an io vector of bytes.
*)

writev
PROCEDURE writev (fd: INTEGER; v: ADDRESS; n: INTEGER) : [ INTEGER ] ;

(*
   getcwd - copies the absolute pathname of the
            current working directory to the array pointed to by buf,
            which is of length size.

            If the current absolute path name would require a buffer
            longer than size elements, NULL is returned, and errno is
            set to ERANGE; an application should check for this error,
            and allocate a larger buffer if necessary.
*)

getcwd
PROCEDURE getcwd (buf: ADDRESS; size: CSIZE_T) : ADDRESS ;

(*
   chown - The  owner  of  the  file  specified  by  path or by fd is
           changed.  Only the super-user may change the  owner  of  a
           file.   The  owner  of  a file may change the group of the
           file to any group of which that owner is  a  member.   The
           super-user may change the group arbitrarily.

           If  the owner or group is specified as -1, then that ID is
           not changed.

           On success, zero is returned.  On error, -1  is  returned,
           and errno is set appropriately.
*)

chown
PROCEDURE chown (filename: ADDRESS; uid, gid: INTEGER) : [ INTEGER ] ;

(*
   strlen - returns the length of string, a.
*)

strlen
```

(continues on next page)

(continued from previous page)

```
PROCEDURE strlen (a: ADDRESS) : CSIZE_T ;

(*
   strcpy - copies string, src, into, dest.
            It returns dest.
*)

strcpy
PROCEDURE strcpy (dest, src: ADDRESS) : [ ADDRESS ] ;

(*
   strncpy - copies string, src, into, dest, copying at most, n, bytes.
             It returns dest.
*)

strncpy
PROCEDURE strncpy (dest, src: ADDRESS; n: CARDINAL) : [ ADDRESS ] ;

(*
   unlink - removes file and returns 0 if successful.
*)

unlink
PROCEDURE unlink (file: ADDRESS) : [ INTEGER ] ;

(*
   memcpy - copy memory area

   SYNOPSIS

   #include <string.h>

   void *memcpy(void *dest, const void *src, size_t n);
   It returns dest.
*)

memcpy
PROCEDURE memcpy (dest, src: ADDRESS; size: CSIZE_T) : [ ADDRESS ] ;

(*
   memset - fill memory with a constant byte

   SYNOPSIS

   #include <string.h>

   void *memset(void *s, int c, size_t n);
   It returns s.
*)

memset
```

(continues on next page)

(continued from previous page)

```
PROCEDURE memset (s: ADDRESS; c: INTEGER; size: CSIZE_T) : [ ADDRESS ] ;

(*
   memmove - copy memory areas which may overlap

   SYNOPSIS

   #include <string.h>

   void *memmove(void *dest, const void *src, size_t n);
   It returns dest.
*)

memmove
PROCEDURE memmove (dest, src: ADDRESS; size: CSIZE_T) : [ ADDRESS ] ;

(*
   int printf(const char *format, ...);
*)

printf
PROCEDURE printf (format: ARRAY OF CHAR; ...) : [ INTEGER ] ;

(*
   setenv - sets environment variable, name, to value.
            It will overwrite an existing value if, overwrite,
            is true.  It returns 0 on success and -1 for an error.
*)

setenv
PROCEDURE setenv (name: ADDRESS; value: ADDRESS; overwrite: INTEGER) : [ INTEGER ] ;

(*
   srand - initialize the random number seed.
*)

srand
PROCEDURE srand (seed: INTEGER) ;

(*
   rand - return a random integer.
*)

rand
PROCEDURE rand () : INTEGER ;

(*
   time - returns a pointer to the time_t value. If, a,
          is not NIL then the libc value is copied into
          memory at address, a.
*)
```

(continues on next page)

(continued from previous page)

```
time
PROCEDURE time (a: ADDRESS) : time_t ;

(*
   localtime - returns a pointer to the libc copy of the tm
               structure.
*)

localtime
PROCEDURE localtime (VAR t: time_t) : ADDRESS ;

(*
   ftime - return date and time.
*)

ftime
PROCEDURE ftime (VAR t: timeb) : [ INTEGER ] ;

(*
   shutdown - shutdown a socket, s.
              if how = 0, then no more reads are allowed.
              if how = 1, then no more writes are allowed.
              if how = 2, then mo more reads or writes are allowed.
*)

shutdown
PROCEDURE shutdown (s: INTEGER; how: INTEGER) : [ INTEGER ] ;

(*
   rename - change the name or location of a file
*)

rename
PROCEDURE rename (oldpath, newpath: ADDRESS) : [ INTEGER ] ;

(*
   setjmp - returns 0 if returning directly, and non-zero
            when returning from longjmp using the saved
            context.
*)

setjmp
PROCEDURE setjmp (env: ADDRESS) : INTEGER ;

(*
   longjmp - restores the environment saved by the last call
             of setjmp with the corresponding env argument.
             After longjmp is completed, program execution
             continues as if the corresponding call of setjmp
             had just returned the value val.  The value of
```

(continues on next page)

(continued from previous page)

```
             val must not be zero.
*)

longjmp
PROCEDURE longjmp (env: ADDRESS; val: INTEGER) ;

(*
   atexit - execute, proc, when the function exit is called.
*)

atexit
PROCEDURE atexit (proc: exitP) : [ INTEGER ] ;

(*
   ttyname - returns a pointer to a string determining the ttyname.
*)

ttyname
PROCEDURE ttyname (filedes: INTEGER) : ADDRESS ;

(*
   sleep - calling thread sleeps for seconds.
*)

sleep
PROCEDURE sleep (seconds: CARDINAL) : [ CARDINAL ] ;

(*
   execv - execute a file.
*)

execv
PROCEDURE execv (pathname: ADDRESS; argv: ADDRESS) : [ INTEGER ] ;

END libc.
```

18.1.56 gm2-libs/libm

```
DEFINITION MODULE FOR "C" libm ;

(* Users are strongly advised to use MathLib0 or RealMath as calls
   to functions within these modules will generate inline code.
   This module is used by MathLib0 and RealMath when inline code cannot
   be generated.  *)

EXPORT UNQUALIFIED sin, sinl, sinf,
                   cos, cosl, cosf,
                   tan, tanl, tanf,
                   sqrt, sqrtl, sqrtf,
                   asin, asinl, asinf,
```

(continues on next page)

(continued from previous page)

```
                    acos, acosl, acosf,
                    atan, atanl, atanf,
                    atan2, atan2l, atan2f,
                    exp, expl, expf,
                    log, logl, logf,
                    exp10, exp10l, exp10f,
                    pow, powl, powf,
                    floor, floorl, floorf,
                    ceil, ceill, ceilf ;

sin
PROCEDURE sin (x: REAL) : REAL ;
sinl
PROCEDURE sinl (x: LONGREAL) : LONGREAL ;
sinf
PROCEDURE sinf (x: SHORTREAL) : SHORTREAL ;
cos
PROCEDURE cos (x: REAL) : REAL ;
cosl
PROCEDURE cosl (x: LONGREAL) : LONGREAL ;
cosf
PROCEDURE cosf (x: SHORTREAL) : SHORTREAL ;
tan
PROCEDURE tan (x: REAL) : REAL ;
tanl
PROCEDURE tanl (x: LONGREAL) : LONGREAL ;
tanf
PROCEDURE tanf (x: SHORTREAL) : SHORTREAL ;
sqrt
PROCEDURE sqrt (x: REAL) : REAL ;
sqrtl
PROCEDURE sqrtl (x: LONGREAL) : LONGREAL ;
sqrtf
PROCEDURE sqrtf (x: SHORTREAL) : SHORTREAL ;
asin
PROCEDURE asin (x: REAL) : REAL ;
asinl
PROCEDURE asinl (x: LONGREAL) : LONGREAL ;
asinf
PROCEDURE asinf (x: SHORTREAL) : SHORTREAL ;
acos
PROCEDURE acos (x: REAL) : REAL ;
acosl
PROCEDURE acosl (x: LONGREAL) : LONGREAL ;
acosf
PROCEDURE acosf (x: SHORTREAL) : SHORTREAL ;
atan
PROCEDURE atan (x: REAL) : REAL ;
atanl
PROCEDURE atanl (x: LONGREAL) : LONGREAL ;
atanf
```

(continues on next page)

(continued from previous page)

```
PROCEDURE atanf (x: SHORTREAL) : SHORTREAL ;
atan2
PROCEDURE atan2 (x, y: REAL) : REAL ;
atan2l
PROCEDURE atan2l (x, y: LONGREAL) : LONGREAL ;
atan2f
PROCEDURE atan2f (x, y: SHORTREAL) : SHORTREAL ;
exp
PROCEDURE exp (x: REAL) : REAL ;
expl
PROCEDURE expl (x: LONGREAL) : LONGREAL ;
expf
PROCEDURE expf (x: SHORTREAL) : SHORTREAL ;
log
PROCEDURE log (x: REAL) : REAL ;
logl
PROCEDURE logl (x: LONGREAL) : LONGREAL ;
logf
PROCEDURE logf (x: SHORTREAL) : SHORTREAL ;
exp10
PROCEDURE exp10 (x: REAL) : REAL ;
exp10l
PROCEDURE exp10l (x: LONGREAL) : LONGREAL ;
exp10f
PROCEDURE exp10f (x: SHORTREAL) : SHORTREAL ;
pow
PROCEDURE pow (x, y: REAL) : REAL ;
powl
PROCEDURE powl (x, y: LONGREAL) : LONGREAL ;
powf
PROCEDURE powf (x, y: SHORTREAL) : SHORTREAL ;
floor
PROCEDURE floor (x: REAL) : REAL ;
floorl
PROCEDURE floorl (x: LONGREAL) : LONGREAL ;
floorf
PROCEDURE floorf (x: SHORTREAL) : SHORTREAL ;
ceil
PROCEDURE ceil (x: REAL) : REAL ;
ceill
PROCEDURE ceill (x: LONGREAL) : LONGREAL ;
ceilf
PROCEDURE ceilf (x: SHORTREAL) : SHORTREAL ;

END libm.
```

18.1.57 gm2-libs/sckt

```
DEFINITION MODULE sckt ;

FROM SYSTEM IMPORT ADDRESS ;
EXPORT UNQUALIFIED tcpServerState,
                   tcpServerEstablish, tcpServerEstablishPort,
                   tcpServerAccept, getLocalIP,
                   tcpServerPortNo, tcpServerIP, tcpServerSocketFd,
                   tcpServerClientIP, tcpServerClientPortNo,
                   tcpClientState,
                   tcpClientSocket, tcpClientSocketIP, tcpClientConnect,
                   tcpClientPortNo, tcpClientIP, tcpClientSocketFd ;

TYPE
tcpServerState (type)
   tcpServerState = ADDRESS ;
tcpClientState (type)
   tcpClientState = ADDRESS ;

(*
   tcpServerEstablish - returns a tcpState containing the relevant
                        information about a socket declared to receive
                        tcp connections.
*)

tcpServerEstablish
PROCEDURE tcpServerEstablish () : tcpServerState ;

(*
   tcpServerEstablishPort - returns a tcpState containing the relevant
                            information about a socket declared to receive
                            tcp connections.  This method attempts to use
                            the port specified by the parameter.
*)

tcpServerEstablishPort
PROCEDURE tcpServerEstablishPort (port: CARDINAL) : tcpServerState ;

(*
   tcpServerAccept - returns a file descriptor once a client has connected and
                     been accepted.
*)

tcpServerAccept
PROCEDURE tcpServerAccept (s: tcpServerState) : INTEGER ;

(*
   tcpServerPortNo - returns the portNo from structure, s.
*)

tcpServerPortNo
```

(continues on next page)

(continued from previous page)

```
PROCEDURE tcpServerPortNo (s: tcpServerState) : CARDINAL ;

(*
   tcpSocketFd - returns the sockFd from structure, s.
*)

tcpServerSocketFd
PROCEDURE tcpServerSocketFd (s: tcpServerState) : INTEGER ;

(*
   getLocalIP - returns the IP address of this machine.
*)

getLocalIP
PROCEDURE getLocalIP (s: tcpServerState) : CARDINAL ;

(*
   tcpServerIP - returns the IP address from structure, s.
*)

tcpServerIP
PROCEDURE tcpServerIP (s: tcpServerState) : CARDINAL ;

(*
   tcpServerClientIP - returns the IP address of the client who
                       has connected to server, s.
*)

tcpServerClientIP
PROCEDURE tcpServerClientIP (s: tcpServerState) : CARDINAL ;

(*
   tcpServerClientPortNo - returns the port number of the client who
                           has connected to server, s.
*)

tcpServerClientPortNo
PROCEDURE tcpServerClientPortNo (s: tcpServerState) : CARDINAL ;

(*
   tcpClientSocket - returns a file descriptor (socket) which has
                     connected to, serverName:portNo.
*)

tcpClientSocket
PROCEDURE tcpClientSocket (serverName: ADDRESS; portNo: CARDINAL) : tcpClientState ;

(*
   tcpClientSocketIP - returns a file descriptor (socket) which has
                       connected to, ip:portNo.
*)
```

(continues on next page)

(continued from previous page)

```
tcpClientSocketIP
PROCEDURE tcpClientSocketIP (ip: CARDINAL; portNo: CARDINAL) : tcpClientState ;

(*
   tcpClientConnect - returns the file descriptor associated with, s,
                      once a connect has been performed.
*)

tcpClientConnect
PROCEDURE tcpClientConnect (s: tcpClientState) : INTEGER ;

(*
   tcpClientPortNo - returns the portNo from structure, s.
*)

tcpClientPortNo
PROCEDURE tcpClientPortNo (s: tcpClientState) : INTEGER ;

(*
   tcpClientSocketFd - returns the sockFd from structure, s.
*)

tcpClientSocketFd
PROCEDURE tcpClientSocketFd (s: tcpClientState) : INTEGER ;

(*
   tcpClientIP - returns the IP address from structure, s.
*)

tcpClientIP
PROCEDURE tcpClientIP (s: tcpClientState) : CARDINAL ;

END sckt.
```

18.1.58 gm2-libs/termios

```
DEFINITION MODULE termios ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
TERMIOS (type)
   TERMIOS = ADDRESS ;

ControlChar (type)
   ControlChar = (vintr, vquit, verase, vkill, veof, vtime, vmin,
                  vswtc, vstart, vstop, vsusp, veol, vreprint, vdiscard,
                  vwerase, vlnext, veol2) ;
```

(continues on next page)

(continued from previous page)

```
Flag (type)
   Flag = (
            (* input flag bits *)
            ignbrk, ibrkint, ignpar, iparmrk, inpck, istrip, inlcr,
            igncr, icrnl, iuclc, ixon, ixany, ixoff, imaxbel,
            (* output flag bits *)
            opost, olcuc, onlcr, ocrnl, onocr, onlret, ofill, ofdel,
            onl0, onl1, ocr0, ocr1, ocr2, ocr3,
            otab0, otab1, otab2, otab3, obs0, obs1, off0, off1, ovt0, ovt1,
            (* baud rate *)
            b0, b50, b75, b110, b135, b150, b200, b300, b600, b1200,
            b1800, b2400, b4800, b9600, b19200, b38400,
            b57600, b115200, b240400, b460800, b500000, b576000,
            b921600, b1000000, b1152000, b1500000, b2000000, b2500000,
            b3000000, b3500000, b4000000, maxbaud, crtscts,
            (* character size *)
            cs5, cs6, cs7, cs8, cstopb, cread, parenb, parodd, hupcl, clocal,
            (* local flags *)
            lisig, licanon, lxcase, lecho, lechoe, lechok, lechonl, lnoflsh,
            ltopstop, lechoctl, lechoprt, lechoke, lflusho, lpendin, liexten) ;

(*
   InitTermios - new data structure.
*)

InitTermios
PROCEDURE InitTermios () : TERMIOS ;

(*
   KillTermios - delete data structure.
*)

KillTermios
PROCEDURE KillTermios (t: TERMIOS) : TERMIOS ;

(*
   cfgetospeed - return output baud rate.
*)

cfgetospeed
PROCEDURE cfgetospeed (t: TERMIOS) : INTEGER ;

(*
   cfgetispeed - return input baud rate.
*)

cfgetispeed
PROCEDURE cfgetispeed (t: TERMIOS) : INTEGER ;

(*
   cfsetospeed - set output baud rate.
```

(continues on next page)

(continued from previous page)

```
*)

cfsetospeed
PROCEDURE cfsetospeed (t: TERMIOS; b: CARDINAL) : INTEGER ;

(*
   cfsetispeed - set input baud rate.
*)

cfsetispeed
PROCEDURE cfsetispeed (t: TERMIOS; b: CARDINAL) : INTEGER ;

(*
   cfsetspeed - set input and output baud rate.
*)

cfsetspeed
PROCEDURE cfsetspeed (t: TERMIOS; b: CARDINAL) : INTEGER ;

(*
   tcgetattr - get state of, fd, into, t.
*)

tcgetattr
PROCEDURE tcgetattr (fd: INTEGER; t: TERMIOS) : INTEGER ;

(*
   The following three functions return the different option values.
*)

tcsnow
PROCEDURE tcsnow () : INTEGER ;   (* alter fd now *)
tcsdrain
PROCEDURE tcsdrain () : INTEGER ; (* alter when all output has been sent *)
tcsflush
PROCEDURE tcsflush () : INTEGER ; (* like drain, except discard any pending input *)

(*
   tcsetattr - set state of, fd, to, t, using option.
*)

tcsetattr
PROCEDURE tcsetattr (fd: INTEGER; option: INTEGER; t: TERMIOS) : INTEGER ;

(*
   cfmakeraw - sets, t, to raw mode.
*)

cfmakeraw
PROCEDURE cfmakeraw (t: TERMIOS) ;
```

(continues on next page)

(continued from previous page)

```
(*
   tcsendbreak - send zero bits for duration.
*)

tcsendbreak
PROCEDURE tcsendbreak (fd: INTEGER; duration: INTEGER) : INTEGER ;

(*
   tcdrain - waits for pending output to be written on, fd.
*)

tcdrain
PROCEDURE tcdrain (fd: INTEGER) : INTEGER ;

(*
   tcflushi - flush input.
*)

tcflushi
PROCEDURE tcflushi (fd: INTEGER) : INTEGER ;

(*
   tcflusho - flush output.
*)

tcflusho
PROCEDURE tcflusho (fd: INTEGER) : INTEGER ;

(*
   tcflushio - flush input and output.
*)

tcflushio
PROCEDURE tcflushio (fd: INTEGER) : INTEGER ;

(*
   tcflowoni - restart input on, fd.
*)

tcflowoni
PROCEDURE tcflowoni (fd: INTEGER) : INTEGER ;

(*
   tcflowoffi - stop input on, fd.
*)

tcflowoffi
PROCEDURE tcflowoffi (fd: INTEGER) : INTEGER ;

(*
   tcflowono - restart output on, fd.
```

(continues on next page)

(continued from previous page)

```
*)

tcflowono
PROCEDURE tcflowono (fd: INTEGER) : INTEGER ;

(*
   tcflowoffo - stop output on, fd.
*)

tcflowoffo
PROCEDURE tcflowoffo (fd: INTEGER) : INTEGER ;

(*
   GetFlag - sets a flag value from, t, in, b, and returns TRUE
             if, t, supports, f.
*)

GetFlag
PROCEDURE GetFlag (t: TERMIOS; f: Flag; VAR b: BOOLEAN) : BOOLEAN ;

(*
   SetFlag - sets a flag value in, t, to, b, and returns TRUE if
             this flag value is supported.
*)

SetFlag
PROCEDURE SetFlag (t: TERMIOS; f: Flag; b: BOOLEAN) : BOOLEAN ;

(*
   GetChar - sets a CHAR, ch, value from, t, and returns TRUE if
             this value is supported.
*)

GetChar
PROCEDURE GetChar (t: TERMIOS; c: ControlChar; VAR ch: CHAR) : BOOLEAN ;

(*
   SetChar - sets a CHAR value in, t, and returns TRUE if, c,
             is supported.
*)

SetChar
PROCEDURE SetChar (t: TERMIOS; c: ControlChar; ch: CHAR) : BOOLEAN ;

END termios.
```

18.1.59 gm2-libs/wrapc

```
DEFINITION MODULE wrapc ;

FROM SYSTEM IMPORT ADDRESS ;

EXPORT QUALIFIED strtime, filesize, fileinode,
                 getrand, getusername, filemtime,
                 getnameuidgid, signbit, signbitf, signbitl,
                isfinite, isfinitel, isfinitef ;

(*
   strtime - returns the C string for the equivalent C asctime
             function.
*)

strtime
PROCEDURE strtime () : ADDRESS ;

(*
   filesize - assigns the size of a file, f, into low, high and
              returns zero if successful.
*)

filesize
PROCEDURE filesize (f: INTEGER; VAR low, high: CARDINAL) : INTEGER ;

(*
   fileinode - return the inode associated with file, f.
*)

fileinode
PROCEDURE fileinode (f: INTEGER; VAR low, high: CARDINAL) : INTEGER ;

(*
   filemtime - returns the mtime of a file, f.
*)

filemtime
PROCEDURE filemtime (f: INTEGER) : INTEGER ;

(*
   getrand - returns a random number between 0..n-1
*)

getrand
PROCEDURE getrand (n: INTEGER) : INTEGER ;

(*
   getusername - returns a C string describing the current user.
*)
```

(continues on next page)

(continued from previous page)

```
getusername
PROCEDURE getusername () : ADDRESS ;

(*
   getnameuidgid - fills in the, uid, and, gid, which represents
                   user, name.
*)

getnameuidgid
PROCEDURE getnameuidgid (name: ADDRESS; VAR uid, gid: INTEGER) ;

(*
   in C these procedure functions are really macros, so we provide
   real C functions and let gm2 call these if the builtins
   are unavailable.
*)

signbit
PROCEDURE signbit (r: REAL) : INTEGER ;
signbitf
PROCEDURE signbitf (s: SHORTREAL) : INTEGER ;
signbitl
PROCEDURE signbitl (l: LONGREAL) : INTEGER ;

(*
   isfinite - provide non builtin alternative to the gcc builtin isfinite.
              Returns 1 if x is finite and 0 if it is not.
*)

isfinite
PROCEDURE isfinite (x: REAL) : INTEGER ;

(*
   isfinitef - provide non builtin alternative to the gcc builtin isfinite.
               Returns 1 if x is finite and 0 if it is not.
*)

isfinitef
PROCEDURE isfinitef (x: SHORTREAL) : INTEGER ;

(*
   isfinitel - provide non builtin alternative to the gcc builtin isfinite.
               Returns 1 if x is finite and 0 if it is not.
*)

isfinitel
PROCEDURE isfinitel (x: LONGREAL) : INTEGER ;

END wrapc.
```

18.2 PIM coroutine support

This directory contains a PIM `SYSTEM` containing the `PROCESS` primitives built on top of GNU Pthreads.

The justification for this approach is that it provides a `SYSTEM` compatible with Programming in Modula-2 [234] and the Logitech 3.0 compiler. It also allows higher level executives to be ported onto GM2 with little effort. The disadvantage with this approach is that `IOTRANSFER` is not preemptive. `IOTRANSFER` will only context switch when a call to `LISTEN` is made or a call to `SYSTEM.TurnInterrupts` is made.

In practice this limitation can be tolerated as long as processes perform IO at some point (or wait for a timer interrupt) or call `SYSTEM.TurnInterrupts`. But nevertheless a `LOOP END` will starve all other processes. However the great advantage is that GNU Modula-2 can offer users the ability to use `IOTRANSFER`, `TRANSFER`, `NEWPROCESS` in user space, on a multi-user operating system and across a range of platforms.

The GNU Modula-2 `SYSTEM` works by utilizing the user context switching mechanism provided by GNU Pthreads. `NEWPROCESS` creates a new context, `TRANSFER` switches contexts. `IOTRANSFER` is more complex. There is a support module `SysVec` which provides pseudo interrupt vectors. These can be created from input/output file descriptors or timer events `timeval`. This vector is then passed to `IOTRANSFER` which keeps track of which file descriptors and timevals are active. When a call to `TurnInterrupts` or `LISTEN` is made the sub system calls `pth_select` and tests for any ready file descriptor or timeout. A ready file descriptor or timeout will ultimately cause the backwards `TRANSFER` inside `IOTRANSFER` to take effect.

See the `gm2/examples/executive` directory for an executive and timerhandler module which provide higher level process creation, synchronisation and interrupt handling routines. These libraries have been tested with the examples shown in `gm2/examples/executive` and `gm2/gm2-libs-coroutines`.

Users of these libraries and the libraries in `gm2/examples/executive` must link their application against the GNU Pthread library (typically by using `-lpth`).

18.2.1 gm2-libs-coroutines/Debug

```
DEFINITION MODULE Debug ;

(*
    Description: provides some simple debugging routines.
*)

EXPORT QUALIFIED Halt, DebugString, PushOutput ;

TYPE
WriteP (type)
   WriteP = PROCEDURE (CHAR) ;

(*
   Halt - writes a message in the format:
          Module:Line:Message
```

(continues on next page)

(continued from previous page)

```
          It then terminates by calling HALT.
*)

Halt
PROCEDURE Halt (File    : ARRAY OF CHAR;
                LineNo  : CARDINAL;
                Function,
                Message : ARRAY OF CHAR) ;

(*
   DebugString - writes a string to the debugging device (Scn.Write).
                 It interprets \n as carriage return, linefeed.
*)

DebugString
PROCEDURE DebugString (a: ARRAY OF CHAR) ;

(*
   PushOutput - pushes the output procedure, p, which is used Debug.
*)

PushOutput
PROCEDURE PushOutput (p: WriteP) ;

(*
   PopOutput - pops the current output procedure from the stack.
*)

PopOutput
PROCEDURE PopOutput ;

END Debug.
```

18.2.2 gm2-libs-coroutines/Executive

```
DEFINITION MODULE Executive ;

EXPORT QUALIFIED SEMAPHORE, DESCRIPTOR,
                 InitProcess, KillProcess, Resume, Suspend, InitSemaphore,
                 Wait, Signal, WaitForIO, Ps, GetCurrentProcess,
                 RotateRunQueue, ProcessName, DebugProcess ;

TYPE
SEMAPHORE (type)
   SEMAPHORE ;         (* defines Dijkstra's semaphores *)
DESCRIPTOR (type)
   DESCRIPTOR ;        (* handle onto a process         *)

(*
```

(continues on next page)

(continued from previous page)

```
   InitProcess - initializes a process which is held in the suspended
                 state. When the process is resumed it will start executing
                 procedure, p. The process has a maximum stack size of,
                 StackSize, bytes and its textual name is, Name.
                 The StackSize should be at least 5000 bytes.
*)

InitProcess
PROCEDURE InitProcess (p: PROC; StackSize: CARDINAL;
                       Name: ARRAY OF CHAR) : DESCRIPTOR ;

(*
   KillProcess - kills the current process. Notice that if InitProcess
                 is called again, it might reuse the DESCRIPTOR of the
                 killed process. It is the responsibility of the caller
                 to ensure all other processes understand this process
                 is different.
*)

KillProcess
PROCEDURE KillProcess ;

(*
   Resume - resumes a suspended process. If all is successful then the process, p,
            is returned. If it fails then NIL is returned.
*)

Resume
PROCEDURE Resume (d: DESCRIPTOR) : DESCRIPTOR ;

(*
   Suspend - suspend the calling process.
             The process can only continue running if another process
             Resumes it.
*)

Suspend
PROCEDURE Suspend ;

(*
   InitSemaphore - creates a semaphore whose initial value is, v, and
                   whose name is, Name.
*)

InitSemaphore
PROCEDURE InitSemaphore (v: CARDINAL; Name: ARRAY OF CHAR) : SEMAPHORE ;

(*
   Wait - performs dijkstra's P operation on a semaphore.
          A process which calls this procedure will
          wait until the value of the semaphore is > 0
```

(continues on next page)

(continued from previous page)

```
          and then it will decrement this value.
*)

Wait
PROCEDURE Wait (s: SEMAPHORE) ;

(*
   Signal - performs dijkstra's V operation on a semaphore.
            A process which calls the procedure will increment
            the semaphores value.
*)

Signal
PROCEDURE Signal (s: SEMAPHORE) ;

(*
   WaitForIO - waits for an interrupt to occur on vector, VectorNo.
*)

WaitForIO
PROCEDURE WaitForIO (VectorNo: CARDINAL) ;

(*
   Ps - displays a process list together with process status.
*)

Ps
PROCEDURE Ps ;

(*
   GetCurrentProcess - returns the descriptor of the current running
                       process.
*)

GetCurrentProcess
PROCEDURE GetCurrentProcess () : DESCRIPTOR ;

(*
   RotateRunQueue - rotates the process run queue.
                    It does not call the scheduler.
*)

RotateRunQueue
PROCEDURE RotateRunQueue ;

(*
   ProcessName - displays the name of process, d, through
                 DebugString.
*)

ProcessName
```

(continues on next page)

(continued from previous page)

```
PROCEDURE ProcessName (d: DESCRIPTOR) ;

(*
   DebugProcess - gdb debug handle to enable users to debug deadlocked
                  semaphore processes.
*)

DebugProcess
PROCEDURE DebugProcess (d: DESCRIPTOR) ;

END Executive.
```

18.2.3 gm2-libs-coroutines/KeyBoardLEDs

```
DEFINITION MODULE KeyBoardLEDs ;

EXPORT QUALIFIED SwitchLeds,
                 SwitchScroll, SwitchNum, SwitchCaps ;

(*
   SwitchLeds - switch the keyboard LEDs to the state defined
                by the BOOLEAN variables. TRUE = ON.
*)

SwitchLeds
PROCEDURE SwitchLeds (NumLock, CapsLock, ScrollLock: BOOLEAN) ;

(*
   SwitchScroll - switchs the scroll LED on or off.
*)

SwitchScroll
PROCEDURE SwitchScroll (Scroll: BOOLEAN) ;

(*
   SwitchNum - switches the Num LED on or off.
*)

SwitchNum
PROCEDURE SwitchNum (Num: BOOLEAN) ;

(*
   SwitchCaps - switches the Caps LED on or off.
*)

SwitchCaps
PROCEDURE SwitchCaps (Caps: BOOLEAN) ;

END KeyBoardLEDs.
```

18.2.4 gm2-libs-coroutines/SYSTEM

```
DEFINITION MODULE SYSTEM ;

(* This module is designed to be used on a native operating system
   rather than an embedded system as it implements the coroutine
   primitives TRANSFER, IOTRANSFER and
   NEWPROCESS through the GNU Pthread library.  *)

FROM COROUTINES IMPORT PROTECTION ;

EXPORT QUALIFIED (* the following are built into the compiler: *)
                 LOC, WORD, BYTE, ADDRESS, INTEGER8,
                 INTEGER16, INTEGER32, INTEGER64, CARDINAL8,
                 CARDINAL16, CARDINAL32, CARDINAL64, WORD16,
                 WORD32, WORD64, BITSET8, BITSET16,
                 BITSET32, REAL32, REAL64, REAL128,
                 COMPLEX32, COMPLEX64, COMPLEX128, CSIZE_T,
                 CSSIZE_T,
                 ADR, TSIZE, ROTATE, SHIFT, THROW, TBITSIZE,
                 (* SIZE is exported depending upon -fpim2 and
                    -fpedantic *)
                 (* and the rest are implemented in SYSTEM.mod *)
                 PROCESS, TRANSFER, NEWPROCESS, IOTRANSFER,
                 LISTEN,
                 ListenLoop, TurnInterrupts,
                 (* Internal GM2 compiler functions *)
                 ShiftVal, ShiftLeft, ShiftRight,
                 RotateVal, RotateLeft, RotateRight ;

TYPE
PROCESS (type)
   PROCESS  = RECORD
                 context: INTEGER ;
END (type)
              END ;
(* all the following types are declared internally to gm2
   LOC ;
   WORD ;
   BYTE ;
   ADDRESS ;
   INTEGER8 ;
   INTEGER16 ;
   INTEGER32 ;
   INTEGER64 ;
   CARDINAL8 ;
   CARDINAL16 ;
   CARDINAL32 ;
   CARDINAL64 ;
   WORD16 ;
   WORD32 ;
   WORD64 ;
```

(continues on next page)

(continued from previous page)

```
   BITSET8 ;
   BITSET16 ;
   BITSET32 ;
   REAL32 ;
   REAL64 ;
   REAL128 ;
   COMPLEX32 ;
   COMPLEX64 ;
   COMPLEX128 ;
   CSIZE_T ;
   CSSIZE_T ;
*)

(*
   TRANSFER - save the current volatile environment into, p1.
              Restore the volatile environment from, p2.
*)

TRANSFER
PROCEDURE TRANSFER (VAR p1: PROCESS; p2: PROCESS) ;

(*
   NEWPROCESS - p is a parameterless procedure, a, is the origin of
                the workspace used for the process stack and containing
                the volatile environment of the process.  StackSize, is
                the maximum size of the stack in bytes which can be used
                by this process.  new, is the new process.
*)

NEWPROCESS
PROCEDURE NEWPROCESS (p: PROC; a: ADDRESS; StackSize: CARDINAL; VAR new: PROCESS) ;

(*
   IOTRANSFER - saves the current volatile environment into, First,
                and restores volatile environment, Second.
                When an interrupt, InterruptNo, is encountered then
                the reverse takes place. (The then current volatile
                environment is shelved onto Second and First is resumed).

                NOTE: that upon interrupt the Second might not be the
                      same process as that before the original call to
                      IOTRANSFER.
*)

IOTRANSFER
PROCEDURE IOTRANSFER (VAR First, Second: PROCESS; InterruptNo: CARDINAL) ;

(*
   LISTEN - briefly listen for any interrupts.
*)
```

(continues on next page)

(continued from previous page)

```
LISTEN
PROCEDURE LISTEN ;

(*
   ListenLoop - should be called instead of users writing:

                LOOP
                   LISTEN
                END

                It performs the same function but yields
                control back to the underlying operating system
                via a call to pth_select.
                It also checks for deadlock.
                This function returns when an interrupt occurs ie
                a file descriptor becomes ready or a time event expires.
                See the module RTint.
*)

ListenLoop
PROCEDURE ListenLoop ;

(*
   TurnInterrupts - switches processor interrupts to the protection
                    level, to.  It returns the old value.
*)

TurnInterrupts
PROCEDURE TurnInterrupts (to: PROTECTION) : PROTECTION ;

(*
   all the functions below are declared internally to gm2
   ======================================================

ADR
PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
  (* Returns the address of variable v. *)

SIZE
PROCEDURE SIZE (v: <type>) : ZType;
  (* Returns the number of BYTES used to store a v of
     any specified <type>.  Only available if -fpim2 is used.
  *)

TSIZE
PROCEDURE TSIZE (<type>) : CARDINAL;
  (* Returns the number of BYTES used to store a value of the
     specified <type>.
  *)

ROTATE
```

(continues on next page)

(continued from previous page)

```
PROCEDURE ROTATE (val: <a set type>;
                  num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by rotating up or down
     (left or right) by the absolute value of num.  The direction is
     down if the sign of num is negative, otherwise the direction is up.
  *)

SHIFT
PROCEDURE SHIFT (val: <a set type>;
                 num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by shifting up or down
     (left or right) by the absolute value of num, introducing
     zeros as necessary.  The direction is down if the sign of
     num is negative, otherwise the direction is up.
  *)

THROW
PROCEDURE THROW (i: INTEGER) ;
  (*
     THROW is a GNU extension and was not part of the PIM or ISO
     standards.  It throws an exception which will be caught by the EXCEPT
     block (assuming it exists).  This is a compiler builtin function which
     interfaces to the GCC exception handling runtime system.
     GCC uses the term throw, hence the naming distinction between
     the GCC builtin and the Modula-2 runtime library procedure Raise.
     The later library procedure Raise will call SYSTEM.THROW after
     performing various housekeeping activities.
  *)

TBITSIZE
PROCEDURE TBITSIZE (<type>) : CARDINAL ;
  (* Returns the minimum number of bits necessary to represent
     <type>.  This procedure function is only useful for determining
     the number of bits used for any type field within a packed RECORD.
     It is not particularly useful elsewhere since <type> might be
     optimized for speed, for example a BOOLEAN could occupy a WORD.
  *)
*)

(* The following procedures are invoked by GNU Modula-2 to
   shift non word sized set types. They are not strictly part
   of the core PIM Modula-2, however they are used by
   GNU Modula-2 to implement the SHIFT procedure defined above,
   which are in turn used by the Logitech compatible libraries.

   Users will access these procedures by using the procedure
   SHIFT above and GNU Modula-2 will map SHIFT onto one of
   the following procedures.
*)

(*
```

(continues on next page)

(continued from previous page)

```
   ShiftVal - is a runtime procedure whose job is to implement
              the SHIFT procedure of ISO SYSTEM. GNU Modula-2 will
              inline a SHIFT of a single WORD sized set and will only
              call this routine for larger sets.
*)

ShiftVal
PROCEDURE ShiftVal (VAR s, d: ARRAY OF BITSET;
                    SetSizeInBits: CARDINAL;
                    ShiftCount: INTEGER) ;

(*
   ShiftLeft - performs the shift left for a multi word set.
               This procedure might be called by the back end of
               GNU Modula-2 depending whether amount is known at compile
               time.
*)

ShiftLeft
PROCEDURE ShiftLeft (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     ShiftCount: CARDINAL) ;

(*
   ShiftRight - performs the shift left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at compile
                time.
*)

ShiftRight
PROCEDURE ShiftRight (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      ShiftCount: CARDINAL) ;

(*
   RotateVal - is a runtime procedure whose job is to implement
               the ROTATE procedure of ISO SYSTEM. GNU Modula-2 will
               inline a ROTATE of a single WORD (or less)
               sized set and will only call this routine for larger sets.
*)

RotateVal
PROCEDURE RotateVal (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     RotateCount: INTEGER) ;

(*
   RotateLeft - performs the rotate left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at compile
```

(continues on next page)

(continued from previous page)

```
                time.
*)

RotateLeft
PROCEDURE RotateLeft (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      RotateCount: CARDINAL) ;

(*
   RotateRight - performs the rotate right for a multi word set.
                 This procedure might be called by the back end of
                 GNU Modula-2 depending whether amount is known at compile
                 time.
*)

RotateRight
PROCEDURE RotateRight (VAR s, d: ARRAY OF BITSET;
                       SetSizeInBits: CARDINAL;
                       RotateCount: CARDINAL) ;

END SYSTEM.
```

18.2.5 gm2-libs-coroutines/TimerHandler

```
DEFINITION MODULE TimerHandler ;

(* It also provides the Executive with a basic round robin scheduler.  *)

EXPORT QUALIFIED TicksPerSecond, GetTicks,
                 EVENT,
                 Sleep, ArmEvent, WaitOn, Cancel, ReArmEvent ;

CONST
TicksPerSecond  (const)
   TicksPerSecond =   25 ;  (* Number of ticks per second.  *)

TYPE
EVENT (type)
   EVENT ;

(*
   GetTicks - returns the number of ticks since boottime.
*)

GetTicks
PROCEDURE GetTicks () : CARDINAL ;

(*
   Sleep - suspends the current process for a time, t.
           The time is measured in ticks.
```

(continues on next page)

(continued from previous page)

```
*)

Sleep
PROCEDURE Sleep (t: CARDINAL) ;

(*
   ArmEvent - initializes an event, e, to occur at time, t.
              The time, t, is measured in ticks.
              The event is NOT placed onto the event queue.
*)

ArmEvent
PROCEDURE ArmEvent (t: CARDINAL) : EVENT ;

(*
   WaitOn - places event, e, onto the event queue and then the calling
            process suspends. It is resumed up by either the event
            expiring or the event, e, being cancelled.
            TRUE is returned if the event was cancelled
            FALSE is returned if the event expires.
            The event, e, is always assigned to NIL when the function
            finishes.
*)

WaitOn
PROCEDURE WaitOn (VAR e: EVENT) : BOOLEAN ;

(*
   Cancel - cancels the event, e, on the event queue and makes
            the appropriate process runnable again.
            TRUE is returned if the event was cancelled and
            FALSE is returned is the event was not found or
                  no process was waiting on this event.
*)

Cancel
PROCEDURE Cancel (e: EVENT) : BOOLEAN ;

(*
   ReArmEvent - removes an event, e, from the event queue. A new time
                is given to this event and it is then re-inserted onto the
                event queue in the correct place.
                TRUE is returned if this occurred
                FALSE is returned if the event was not found.
*)

ReArmEvent
PROCEDURE ReArmEvent (e: EVENT; t: CARDINAL) : BOOLEAN ;

END TimerHandler.
```

18.3 M2 ISO Libraries

This directory contains the ISO definition modules and some corresponding implementation modules. The definition files: `ChanConsts.def`, `CharClass.def`, `ComplexMath.def`, `ConvStringLong.def`, `ConvStringReal.def`, `ConvTypes.def`, `COROUTINES.def`, `EXCEPTIONS.def`, `GeneralUserExceptions.def`, `IOChan.def`, `IOConsts.def`, `IOLink.def`, `IOLink.def`, `IOResult.def`, `LongComplexMath.def`, `LongConv.def`, `LongIO.def`, `LongMath.def`, `LongStr.def`, `LowLong.def`, `LowReal.def`, `M2EXCEPTION.def`, `Processes.def`, `ProgramArgs.def`, `RawIO.def`, `RealConv.def`, `RealIO.def`, `RealMath.def`, `RealStr.def`, `RndFile.def`, `Semaphores.def`, `SeqFile.def`, `SIOResult.def`, `SLongIO.def`, `SRawIO.def`, `SRealIO.def`, `StdChans.def`, `STextIO.def`, `Storage.def`, `StreamFile.def`, `Strings.def`, `SWholeIO.def`, `SysClock.def`, `SYSTEM.def`, `TERMINATION.def`, `TextIO.def`, `WholeConv.def`, `WholeIO.def` and `WholeStr.def` were defined by the International Standard Information technology - programming languages BS ISO/IEC 10514-1:1996E Part 1: Modula-2, Base Language.

The Copyright to the definition files `ChanConsts.def`, `CharClass.def`, `ComplexMath.def`, `ConvStringLong.def`, `ConvStringReal.def`, `ConvTypes.def`, `COROUTINES.def`, `EXCEPTIONS.def`, `GeneralUserExceptions.def`, `IOChan.def`, `IOConsts.def`, `IOLink.def`, `IOLink.def`, `IOResult.def`, `LongComplexMath.def`, `LongConv.def`, `LongIO.def`, `LongMath.def`, `LongStr.def`, `LowLong.def`, `LowReal.def`, `M2EXCEPTION.def`, `Processes.def`, `ProgramArgs.def`, `RawIO.def`, `RealConv.def`, `RealIO.def`, `RealMath.def`, `RealStr.def`, `RndFile.def`, `Semaphores.def`, `SeqFile.def`, `SIOResult.def`, `SLongIO.def`, `SRawIO.def`, `SRealIO.def`, `StdChans.def`, `STextIO.def`, `Storage.def`, `StreamFile.def`, `Strings.def`, `SWholeIO.def`, `SysClock.def`, `SYSTEM.def`, `TERMINATION.def`, `TextIO.def`, `WholeConv.def`, `WholeIO.def` and `WholeStr.def` belong to ISO/IEC (International Organization for Standardization and International Electrotechnical Commission). The licence allows them to be distributed with the compiler (as described on page 707 of the Information technology - Programming languages Part 1: Modula-2, Base Language. BS ISO/IEC 10514-1:1996).

All implementation modules and `ClientSocket.def`, `LongWholeIO.def`, `M2RTS.def`, `MemStream.def`, `pth.def`, `RandomNumber.def`, `RTdata.def`, `RTentity.def`, `RTfio.def`, `RTio.def`, `ShortComplexMath.def`, `ShortIO.def`, `ShortWholeIO.def`, `SimpleCipher.def`, `SLongWholeIO.def`, `SShortIO.def`, `SShortWholeIO.def`, `StringChan.def` and `wraptime.def` are Copyright of the FSF and are held under the GPLv3 with runtime exceptions.

Under Section 7 of GPL version 3, you are granted additional permissions described in the GCC Runtime Library Exception, version 3.1, as published by the Free Software Foundation.

You should have received a copy of the GNU General Public License and a copy of the GCC Runtime Library Exception along with this program; see the files COPYING3 and COPYING.RUNTIME respectively. If not, see http://www.gnu.org/licenses/.

Notice that GNU Modula-2 contains additional libraries for input/output of `SHORTREAL`, `SHORTCARD`, `SHORTINT`, `LONGCARD`, `LONGINT` data types. It also provides a `RandomNumber`, `SimpleCipher` and `ClientSocket` modules as well as low level modules which allow the IO libraries to coexist with their PIM counterparts.

18.3.1 gm2-libs-iso/COROUTINES

```
DEFINITION MODULE COROUTINES;

(* Facilities for coroutines and the handling of interrupts *)

IMPORT SYSTEM ;

CONST
UnassignedPriority  (const)
  UnassignedPriority = 0 ;

TYPE
COROUTINE (type)
  COROUTINE ; (* Values of this type are created dynamically by NEWCOROUTINE
                 and identify the coroutine in subsequent operations *)
INTERRUPTSOURCE (type)
  INTERRUPTSOURCE = CARDINAL ;
PROTECTION (type)
  PROTECTION = [UnassignedPriority..7] ;

NEWCOROUTINE
PROCEDURE NEWCOROUTINE (procBody: PROC;
                        workspace: SYSTEM.ADDRESS;
                        size: CARDINAL;
                        VAR cr: COROUTINE;
                        [initProtection: PROTECTION = UnassignedPriority]);
  (* Creates a new coroutine whose body is given by procBody, and
     returns the identity of the coroutine in cr. workspace is a
     pointer to the work space allocated to the coroutine; size
     specifies the size of this workspace in terms of SYSTEM.LOC.

     The optarg, initProtection, may contain a single parameter which
     specifies the initial protection level of the coroutine.
  *)

TRANSFER
PROCEDURE TRANSFER (VAR from: COROUTINE; to: COROUTINE);
  (* Returns the identity of the calling coroutine in from, and
     transfers control to the coroutine specified by to.
  *)

IOTRANSFER
PROCEDURE IOTRANSFER (VAR from: COROUTINE; to: COROUTINE);
  (* Returns the identity of the calling coroutine in from and
     transfers control to the coroutine specified by to.  On
     occurrence of an interrupt, associated with the caller, control
     is transferred back to the caller, and the identity of the
     interrupted coroutine is returned in from.  The calling coroutine
     must be associated with a source of interrupts.
  *)
```

(continues on next page)

(continued from previous page)

```
ATTACH
PROCEDURE ATTACH (source: INTERRUPTSOURCE);
  (* Associates the specified source of interrupts with the calling
     coroutine. *)

DETACH
PROCEDURE DETACH (source: INTERRUPTSOURCE);
  (* Dissociates the specified source of interrupts from the calling
     coroutine. *)

IsATTACHED
PROCEDURE IsATTACHED (source: INTERRUPTSOURCE): BOOLEAN;
  (* Returns TRUE if and only if the specified source of interrupts is
     currently associated with a coroutine; otherwise returns FALSE.
  *)

HANDLER
PROCEDURE HANDLER (source: INTERRUPTSOURCE): COROUTINE;
  (* Returns the coroutine, if any, that is associated with the source
     of interrupts. The result is undefined if IsATTACHED(source) =
     FALSE.
  *)

CURRENT
PROCEDURE CURRENT (): COROUTINE;
  (* Returns the identity of the calling coroutine. *)

LISTEN
PROCEDURE LISTEN (p: PROTECTION);
  (* Momentarily changes the protection of the calling coroutine to
     p. *)

PROT
PROCEDURE PROT (): PROTECTION;
  (* Returns the protection of the calling coroutine. *)

(*
   TurnInterrupts - switches processor interrupts to the protection
                    level, to.  It returns the old value.
*)

TurnInterrupts
PROCEDURE TurnInterrupts (to: PROTECTION) : PROTECTION ;

(*
   ListenLoop - should be called instead of users writing:

                LOOP
                   LISTEN
                END
```

(continues on next page)

(continued from previous page)

```
                It performs the same function but yields
                control back to the underlying operating system.
                It also checks for deadlock.
                Note that this function does return when an interrupt occurs.
                (File descriptor becomes ready or time event expires).
*)

ListenLoop
PROCEDURE ListenLoop ;

END COROUTINES.
```

18.3.2 gm2-libs-iso/ChanConsts

```
DEFINITION MODULE ChanConsts;

  (* Common types and values for channel open requests and results *)

TYPE
ChanFlags (type)
  ChanFlags =        (* Request flags possibly given when a channel is opened *)
  ( readFlag,        (* input operations are requested/available *)
    writeFlag,       (* output operations are requested/available *)
    oldFlag,         (* a file may/must/did exist before the channel is opened *)
    textFlag,        (* text operations are requested/available *)
    rawFlag,         (* raw operations are requested/available *)
    interactiveFlag, (* interactive use is requested/applies *)
    echoFlag         (* echoing by interactive device on removal of characters from input
                        stream requested/applies *)
  );

FlagSet (type)
  FlagSet = SET OF ChanFlags;

  (* Singleton values of FlagSet, to allow for example, read + write *)

CONST
read  (const)
  read = FlagSet{readFlag};   (* input operations are requested/available *)
write  (const)
  write = FlagSet{writeFlag}; (* output operations are requested/available *)
old  (const)
  old = FlagSet{oldFlag};     (* a file may/must/did exist before the channel is opened *)
text  (const)
  text = FlagSet{textFlag};   (* text operations are requested/available *)
raw  (const)
  raw = FlagSet{rawFlag};     (* raw operations are requested/available *)
interactive  (const)
  interactive = FlagSet{interactiveFlag}; (* interactive use is requested/applies *)
echo  (const)
```

(continues on next page)

(continued from previous page)

```
  echo = FlagSet{echoFlag};   (* echoing by interactive device on removal of characters from
                                 input stream requested/applies *)

TYPE
OpenResults (type)
  OpenResults =        (* Possible results of open requests *)
    (opened,           (* the open succeeded as requested *)
     wrongNameFormat,  (* given name is in the wrong format for the implementation *)
     wrongFlags,       (* given flags include a value that does not apply to the device *)
     tooManyOpen,      (* this device cannot support any more open channels *)
     outOfChans,       (* no more channels can be allocated *)
     wrongPermissions, (* file or directory permissions do not allow request *)
     noRoomOnDevice,   (* storage limits on the device prevent the open *)
     noSuchFile,       (* a needed file does not exist *)
     fileExists,       (* a file of the given name already exists when a new one is required *)
     wrongFileType,    (* the file is of the wrong type to support the required operations *)
     noTextOperations, (* text operations have been requested, but are not supported *)
     noRawOperations,  (* raw operations have been requested, but are not supported *)
     noMixedOperations,(* text and raw operations have been requested, but they
                          are not supported in combination *)
     alreadyOpen,      (* the source/destination is already open for operations not supported
                          in combination with the requested operations *)
     otherProblem      (* open failed for some other reason *)
    );

END ChanConsts.
```

18.3.3 gm2-libs-iso/CharClass

```
DEFINITION MODULE CharClass;

  (* Classification of values of the type CHAR *)

IsNumeric
PROCEDURE IsNumeric (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a numeric character *)

IsLetter
PROCEDURE IsLetter (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a letter *)

IsUpper
PROCEDURE IsUpper (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as an upper case letter *)

IsLower
PROCEDURE IsLower (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a lower case letter *)

IsControl
```

(continues on next page)

(continued from previous page)

```
PROCEDURE IsControl (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch represents a control function *)

IsWhiteSpace
PROCEDURE IsWhiteSpace (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch represents a space character or a format effector *)

END CharClass.
```

18.3.4 gm2-libs-iso/ClientSocket

```
DEFINITION MODULE ClientSocket ;

FROM IOChan IMPORT ChanId ;
FROM ChanConsts IMPORT FlagSet, OpenResults ;

(*
   OpenSocket - opens a TCP client connection to host:port.
*)

OpenSocket
PROCEDURE OpenSocket (VAR cid: ChanId;
                      host: ARRAY OF CHAR; port: CARDINAL;
                      f: FlagSet; VAR res: OpenResults) ;

(*
   Close - if the channel identified by cid is not open to
           a socket stream, the exception wrongDevice is
           raised; otherwise closes the channel, and assigns
           the value identifying the invalid channel to cid.
*)

Close
PROCEDURE Close (VAR cid: ChanId) ;

(*
   IsSocket - tests if the channel identified by cid is open as
              a client socket stream.
*)

IsSocket
PROCEDURE IsSocket (cid: ChanId) : BOOLEAN ;

END ClientSocket.
```

18.3.5 gm2-libs-iso/ComplexMath

```
DEFINITION MODULE ComplexMath;

  (* Mathematical functions for the type COMPLEX *)

CONST
i  (const)
  i =    CMPLX (0.0, 1.0);
one  (const)
  one =  CMPLX (1.0, 0.0);
zero  (const)
  zero = CMPLX (0.0, 0.0);

abs
PROCEDURE __BUILTIN__ abs (z: COMPLEX): REAL;
  (* Returns the length of z *)

arg
PROCEDURE __BUILTIN__ arg (z: COMPLEX): REAL;
  (* Returns the angle that z subtends to the positive real axis *)

conj
PROCEDURE __BUILTIN__ conj (z: COMPLEX): COMPLEX;
  (* Returns the complex conjugate of z *)

power
PROCEDURE __BUILTIN__ power (base: COMPLEX; exponent: REAL): COMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

sqrt
PROCEDURE __BUILTIN__ sqrt (z: COMPLEX): COMPLEX;
  (* Returns the principal square root of z *)

exp
PROCEDURE __BUILTIN__ exp (z: COMPLEX): COMPLEX;
  (* Returns the complex exponential of z *)

ln
PROCEDURE __BUILTIN__ ln (z: COMPLEX): COMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

sin
PROCEDURE __BUILTIN__ sin (z: COMPLEX): COMPLEX;
  (* Returns the sine of z *)

cos
PROCEDURE __BUILTIN__ cos (z: COMPLEX): COMPLEX;
  (* Returns the cosine of z *)

tan
PROCEDURE __BUILTIN__ tan (z: COMPLEX): COMPLEX;
```

(continues on next page)

(continued from previous page)

```
  (* Returns the tangent of z *)

arcsin
PROCEDURE __BUILTIN__ arcsin (z: COMPLEX): COMPLEX;
  (* Returns the arcsine of z *)

arccos
PROCEDURE __BUILTIN__ arccos (z: COMPLEX): COMPLEX;
  (* Returns the arccosine of z *)

arctan
PROCEDURE __BUILTIN__ arctan (z: COMPLEX): COMPLEX;
  (* Returns the arctangent of z *)

polarToComplex
PROCEDURE polarToComplex (abs, arg: REAL): COMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

scalarMult
PROCEDURE scalarMult (scalar: REAL; z: COMPLEX): COMPLEX;
  (* Returns the scalar product of scalar with z *)

IsCMathException
PROCEDURE IsCMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception in a
     routine from this module; otherwise returns FALSE.
  *)

END ComplexMath.
```

18.3.6 gm2-libs-iso/ConvStringLong

```
DEFINITION MODULE ConvStringLong ;

FROM DynamicStrings IMPORT String ;

(*
   RealToFloatString - converts a real with, sigFigs, into a string
                       and returns the result as a string.
*)

RealToFloatString
PROCEDURE RealToFloatString (real: LONGREAL; sigFigs: CARDINAL) : String ;

(*
   RealToEngString - converts the value of real to floating-point
                     string form, with sigFigs significant figures.
                     The number is scaled with one to three digits
                     in the whole number part and with an exponent
```

(continues on next page)

(continued from previous page)

```
                    that is a multiple of three.
*)

RealToEngString
PROCEDURE RealToEngString (real: LONGREAL; sigFigs: CARDINAL) : String ;

(*
   RealToFixedString - returns the number of characters in the fixed-point
                       string representation of real rounded to the given
                       place relative to the decimal point.
*)

RealToFixedString
PROCEDURE RealToFixedString (real: LONGREAL; place: INTEGER) : String ;

END ConvStringLong.
```

18.3.7 gm2-libs-iso/ConvStringReal

```
DEFINITION MODULE ConvStringReal ;

FROM DynamicStrings IMPORT String ;

(*
   RealToFloatString - converts a real with, sigFigs, into a string
                       and returns the result as a string.
*)

RealToFloatString
PROCEDURE RealToFloatString (real: REAL; sigFigs: CARDINAL) : String ;

(*
   RealToEngString - converts the value of real to floating-point
                     string form, with sigFigs significant figures.
                     The number is scaled with one to three digits
                     in the whole number part and with an exponent
                     that is a multiple of three.
*)

RealToEngString
PROCEDURE RealToEngString (real: REAL; sigFigs: CARDINAL) : String ;

(*
   RealToFixedString - returns the number of characters in the fixed-point
                       string representation of real rounded to the given
                       place relative to the decimal point.
*)

RealToFixedString
PROCEDURE RealToFixedString (real: REAL; place: INTEGER) : String ;
```

(continues on next page)

(continued from previous page)

```
END ConvStringReal.
```

18.3.8 gm2-libs-iso/ConvTypes

```
DEFINITION MODULE ConvTypes;

  (* Common types used in the string conversion modules *)

TYPE
ConvResults (type)
  ConvResults =     (* Values of this type are used to express the format of a string *)
  (
    strAllRight,    (* the string format is correct for the corresponding conversion *)
    strOutOfRange,  (* the string is well-formed but the value cannot be represented *)
    strWrongFormat, (* the string is in the wrong format for the conversion *)
    strEmpty        (* the given string is empty *)
  );

ScanClass (type)
  ScanClass =  (* Values of this type are used to classify input to finite state scanners *)
  (
    padding,   (* a leading or padding character at this point in the scan - ignore it *)
    valid,     (* a valid character at this point in the scan - accept it *)
    invalid,   (* an invalid character at this point in the scan - reject it *)
    terminator (* a terminating character at this point in the scan (not part of token) *)
  );

ScanState (type)
  ScanState =  (* The type of lexical scanning control procedures *)
    PROCEDURE (CHAR, VAR ScanClass, VAR ScanState);

END ConvTypes.
```

18.3.9 gm2-libs-iso/EXCEPTIONS

```
DEFINITION MODULE EXCEPTIONS;

(* Provides facilities for raising user exceptions
   and for making enquiries concerning the current execution state.
*)

TYPE
  ExceptionSource;   (* values of this type are used within library
                        modules to identify the source of raised
                        exceptions *)
ExceptionNumber (type)
```

(continues on next page)

(continued from previous page)

```
  ExceptionNumber = CARDINAL;

AllocateSource
PROCEDURE AllocateSource(VAR newSource: ExceptionSource);
  (* Allocates a unique value of type ExceptionSource *)

RAISE
PROCEDURE RAISE (source: ExceptionSource;
                 number: ExceptionNumber; message: ARRAY OF CHAR);
  (* Associates the given values of source, number and message with
     the current context and raises an exception.
  *)

CurrentNumber
PROCEDURE CurrentNumber (source: ExceptionSource): ExceptionNumber;
  (* If the current coroutine is in the exceptional execution state
     because of the raising of an exception from source, returns
     the corresponding number, and otherwise raises an exception.
  *)

GetMessage
PROCEDURE GetMessage (VAR text: ARRAY OF CHAR);
  (* If the current coroutine is in the exceptional execution state,
     returns the possibly truncated string associated with the
     current context.  Otherwise, in normal execution state,
     returns the empty string.
  *)

IsCurrentSource
PROCEDURE IsCurrentSource (source: ExceptionSource): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state
     because of the raising of an exception from source, returns
     TRUE, and otherwise returns FALSE.
  *)

IsExceptionalExecution
PROCEDURE IsExceptionalExecution (): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state
     because of the raising of an exception, returns TRUE, and
     otherwise returns FALSE.
  *)

END EXCEPTIONS.
```

18.3.10 gm2-libs-iso/ErrnoCategory

```
DEFINITION MODULE ErrnoCategory ;

(*
   provides an interface to errno (if the system
   supports it) which determines whether the current
   errno is a hard or soft error.  These distinctions
   are needed by the ISO Modula-2 libraries.  Not all
   errno values are tested, only those which could be
   related to a device.
*)

IMPORT ChanConsts ;

(*
   IsErrnoHard - returns TRUE if the value of errno is associated with
                 a hard device error.
*)

IsErrnoHard
PROCEDURE IsErrnoHard (e: INTEGER) : BOOLEAN ;

(*
   IsErrnoSoft - returns TRUE if the value of errno is associated with
                 a soft device error.
*)

IsErrnoSoft
PROCEDURE IsErrnoSoft (e: INTEGER) : BOOLEAN ;

(*
   UnAvailable - returns TRUE if the value of errno indicates that
                 the resource or device is unavailable for some
                 reason.
*)

UnAvailable
PROCEDURE UnAvailable (e: INTEGER) : BOOLEAN ;

(*
   GetOpenResults - maps errno onto the ISO Modula-2 enumerated
                    type, OpenResults.
*)

GetOpenResults
PROCEDURE GetOpenResults (e: INTEGER) : ChanConsts.OpenResults ;

END ErrnoCategory.
```

18.3.11 gm2-libs-iso/GeneralUserExceptions

```
DEFINITION MODULE GeneralUserExceptions;

(* Provides facilities for general user-defined exceptions *)

TYPE
GeneralExceptions (type)
  GeneralExceptions = (problem, disaster);

RaiseGeneralException
PROCEDURE RaiseGeneralException (exception: GeneralExceptions;
                                 text: ARRAY OF CHAR);
  (* Raises exception using text as the associated message *)

IsGeneralException
PROCEDURE IsGeneralException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception from
     GeneralExceptions; otherwise returns FALSE.
  *)

GeneralException
PROCEDURE GeneralException(): GeneralExceptions;
  (* If the current coroutine is in the exceptional execution
     state because of the raising of an exception from
     GeneralExceptions, returns the corresponding enumeration value,
     and otherwise raises an exception.
  *)

END GeneralUserExceptions.
```

18.3.12 gm2-libs-iso/IOChan

```
DEFINITION MODULE IOChan;

  (* Types and procedures forming the interface to channels for
     device-independent data transfer modules
  *)

IMPORT IOConsts, ChanConsts, SYSTEM;

TYPE
  ChanId; (* Values of this type are used to identify channels *)

  (* There is one pre-defined value identifying an invalid channel
     on which no data transfer operations are available.  It may
     be used to initialize variables of type ChanId.
  *)
```

(continues on next page)

(continued from previous page)

```
InvalidChan
PROCEDURE InvalidChan (): ChanId;
  (* Returns the value identifying the invalid channel. *)

  (* For each of the following operations, if the device supports
     the operation on the channel, the behaviour of the procedure
     conforms with the description below.  The full behaviour is
     defined for each device module.  If the device does not
     support the operation on the channel, the behaviour of the
     procedure is to raise the exception notAvailable.
  *)

  (* Text operations - these perform any required translation between the
     internal and external representation of text.
  *)

Look
PROCEDURE Look (cid: ChanId; VAR ch: CHAR; VAR res: IOConsts.ReadResults);
  (* If there is a character as the next item in the input stream
     cid, assigns its value to ch without removing it from the stream;
     otherwise the value of ch is not defined.  res (and the stored
     read result) are set to the value allRight, endOfLine, or endOfInput.
  *)

Skip
PROCEDURE Skip (cid: ChanId);
  (* If the input stream cid has ended, the exception skipAtEnd
     is raised; otherwise the next character or line mark in cid is
     removed, and the stored read result is set to the value
     allRight.
  *)

SkipLook
PROCEDURE SkipLook (cid: ChanId; VAR ch: CHAR; VAR res: IOConsts.ReadResults);
  (* If the input stream cid has ended, the exception skipAtEnd is
     raised; otherwise the next character or line mark in cid is
     removed.  If there is a character as the next item in cid
     stream, assigns its value to ch without removing it from the
     stream.  Otherwise, the value of ch is not defined.  res
     (and the stored read result) are set to the value allRight,
     endOfLine, or endOfInput.
  *)

WriteLn
PROCEDURE WriteLn (cid: ChanId);
  (* Writes a line mark over the channel cid. *)

TextRead
PROCEDURE TextRead (cid: ChanId; to: SYSTEM.ADDRESS; maxChars: CARDINAL;
                    VAR charsRead: CARDINAL);
  (* Reads at most maxChars characters from the current line in cid,
```

(continues on next page)

(continued from previous page)

```
     and assigns corresponding values to successive components of
     an ARRAY OF CHAR variable for which the address of the first
     component is to. The number of characters read is assigned to charsRead.
     The stored read result is set to allRight, endOfLine, or endOfInput.
  *)

TextWrite
PROCEDURE TextWrite (cid: ChanId; from: SYSTEM.ADDRESS;
                     charsToWrite: CARDINAL);
  (* Writes a number of characters given by the value of charsToWrite,
     from successive components of an ARRAY OF CHAR variable for which
     the address of the first component is from, to the channel cid.
  *)

  (* Direct raw operations  - these do not effect translation between
     the internal and external representation of data
  *)

RawRead
PROCEDURE RawRead (cid: ChanId; to: SYSTEM.ADDRESS; maxLocs: CARDINAL;
                   VAR locsRead: CARDINAL);
  (* Reads at most maxLocs items from cid, and assigns corresponding
     values to successive components of an ARRAY OF LOC variable for
     which the address of the first component is to. The number of
     characters read is assigned to charsRead. The stored read result
     is set to the value allRight, or endOfInput.
  *)

RawWrite
PROCEDURE RawWrite (cid: ChanId; from: SYSTEM.ADDRESS; locsToWrite: CARDINAL);
  (* Writes a number of items given by the value of charsToWrite,
     from successive components of an ARRAY OF LOC variable for
     which the address of the first component is from, to the channel cid.
  *)

  (* Common operations *)

GetName
PROCEDURE GetName (cid: ChanId; VAR s: ARRAY OF CHAR);
  (* Copies to s a name associated with the channel cid, possibly truncated
     (depending on the capacity of s).
  *)

Reset
PROCEDURE Reset (cid: ChanId);
  (* Resets the channel cid to a state defined by the device module. *)

Flush
PROCEDURE Flush (cid: ChanId);
  (* Flushes any data buffered by the device module out to the channel cid. *)
```

(continues on next page)

(continued from previous page)

```
  (* Access to read results *)

SetReadResult
PROCEDURE SetReadResult (cid: ChanId; res: IOConsts.ReadResults);
  (* Sets the read result value for the channel cid to the value res. *)

ReadResult
PROCEDURE ReadResult (cid: ChanId): IOConsts.ReadResults;
  (* Returns the stored read result value for the channel cid.
     (This is initially the value notKnown).
  *)

  (* Users can discover which flags actually apply to a channel *)

CurrentFlags
PROCEDURE CurrentFlags (cid: ChanId): ChanConsts.FlagSet;
  (* Returns the set of flags that currently apply to the channel cid. *)

  (* The following exceptions are defined for this module and its clients *)

TYPE
ChanExceptions (type)
  ChanExceptions =
    (wrongDevice,      (* device specific operation on wrong device *)
     notAvailable,     (* operation attempted that is not available on that
                          channel *)
     skipAtEnd,        (* attempt to skip data from a stream that has ended *)
     softDeviceError,  (* device specific recoverable error *)
     hardDeviceError,  (* device specific non-recoverable error *)
     textParseError,   (* input data does not correspond to a character or
                          line mark - optional detection *)
     notAChannel       (* given value does not identify a channel -
                          optional detection *)
    );

IsChanException
PROCEDURE IsChanException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception from
     ChanExceptions; otherwise returns FALSE.
  *)

ChanException
PROCEDURE ChanException (): ChanExceptions;
  (* If the current coroutine is in the exceptional execution state
     because of the raising of an exception from ChanExceptions,
     returns the corresponding enumeration value, and otherwise
     raises an exception.
  *)

  (* When a device procedure detects a device error, it raises the
```

(continues on next page)

(continued from previous page)

```
     exception softDeviceError or hardDeviceError.  If these
     exceptions are handled, the following facilities may be
     used to discover an implementation-defined error number for
     the channel.
  *)

TYPE
DeviceErrNum (type)
  DeviceErrNum = INTEGER;

DeviceError
PROCEDURE DeviceError (cid: ChanId): DeviceErrNum;
  (* If a device error exception has been raised for the channel cid,
     returns the error number stored by the device module.
  *)

END IOChan.
```

18.3.13 gm2-libs-iso/IOConsts

```
DEFINITION MODULE IOConsts;

  (* Types and constants for input/output modules *)

TYPE
ReadResults (type)
  ReadResults =  (* This type is used to classify the result of an input operation *)
  (
    notKnown,    (* no read result is set *)
    allRight,    (* data is as expected or as required *)
    outOfRange,  (* data cannot be represented *)
    wrongFormat, (* data not in expected format *)
    endOfLine,   (* end of line seen before expected data *)
    endOfInput   (* end of input seen before expected data *)
  );

END IOConsts.
```

18.3.14 gm2-libs-iso/IOLink

```
DEFINITION MODULE IOLink;

(* Types and procedures for the standard implementation of channels *)

IMPORT IOChan, IOConsts, ChanConsts, SYSTEM;

TYPE
```

(continues on next page)

(continued from previous page)

```
  DeviceId;
    (* Values of this type are used to identify new device modules,
       and are normally obtained by them during their initialization.
    *)

AllocateDeviceId
PROCEDURE AllocateDeviceId (VAR did: DeviceId);
  (* Allocates a unique value of type DeviceId, and assigns this
     value to did. *)

MakeChan
PROCEDURE MakeChan (did: DeviceId; VAR cid: IOChan.ChanId);
  (* Attempts to make a new channel for the device module identified
     by did. If no more channels can be made, the identity of
     the invalid channel is assigned to cid.  Otherwise, the identity
     of a new channel is assigned to cid.
  *)

UnMakeChan
PROCEDURE UnMakeChan (did: DeviceId; VAR cid: IOChan.ChanId);
  (* If the device module identified by did is not the module that
     made the channel identified by cid, the exception wrongDevice is
     raised; otherwise the channel is deallocated, and the value
     identifying the invalid channel is assigned to cid.
  *)

TYPE
DeviceTablePtr (type)
  DeviceTablePtr = POINTER TO DeviceTable;
    (* Values of this type are used to refer to device tables *)

TYPE
LookProc (type)
  LookProc      = PROCEDURE (DeviceTablePtr, VAR CHAR, VAR IOConsts.ReadResults) ;
SkipProc (type)
  SkipProc      = PROCEDURE (DeviceTablePtr) ;
SkipLookProc (type)
  SkipLookProc  = PROCEDURE (DeviceTablePtr, VAR CHAR, VAR IOConsts.ReadResults) ;
WriteLnProc (type)
  WriteLnProc   = PROCEDURE (DeviceTablePtr) ;
TextReadProc (type)
  TextReadProc  = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL, VAR CARDINAL) ;
TextWriteProc (type)
  TextWriteProc = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL) ;
RawReadProc (type)
  RawReadProc   = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL, VAR CARDINAL) ;
RawWriteProc (type)
  RawWriteProc  = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL) ;
GetNameProc (type)
  GetNameProc   = PROCEDURE (DeviceTablePtr, VAR ARRAY OF CHAR) ;
ResetProc (type)
```

(continues on next page)

(continued from previous page)

```
  ResetProc     = PROCEDURE (DeviceTablePtr) ;
FlushProc (type)
  FlushProc     = PROCEDURE (DeviceTablePtr) ;
FreeProc (type)
  FreeProc      = PROCEDURE (DeviceTablePtr) ;
     (* Carry out the operations involved in closing the corresponding
        channel, including flushing buffers, but do not unmake the
        channel.
     *)

TYPE
DeviceData (type)
  DeviceData = SYSTEM.ADDRESS;

DeviceTable (type)
  DeviceTable =
    RECORD                          (* Initialized by MakeChan to: *)
      cd: DeviceData;               (* the value NIL *)
      did: DeviceId;                (* the value given in the call of MakeChan *)
      cid: IOChan.ChanId;           (* the identity of the channel *)
      result: IOConsts.ReadResults;(* the value notKnown *)
      errNum: IOChan.DeviceErrNum; (* undefined *)
      flags: ChanConsts.FlagSet;    (* ChanConsts.FlagSet{} *)
      doLook: LookProc;             (* raise exception notAvailable *)
      doSkip: SkipProc;             (* raise exception notAvailable *)
      doSkipLook: SkipLookProc;     (* raise exception notAvailable *)
      doLnWrite: WriteLnProc;       (* raise exception notAvailable *)
      doTextRead: TextReadProc;     (* raise exception notAvailable *)
      doTextWrite: TextWriteProc;   (* raise exception notAvailable *)
      doRawRead: RawReadProc;       (* raise exception notAvailable *)
      doRawWrite: RawWriteProc;     (* raise exception notAvailable *)
      doGetName: GetNameProc;       (* return the empty string *)
      doReset: ResetProc;           (* do nothing *)
      doFlush: FlushProc;           (* do nothing *)
      doFree: FreeProc;             (* do nothing *)
    END;

  (* The pointer to the device table for a channel is obtained using the
     following procedure: *)

(*
   If the device module identified by did is not the module that made
   the channel identified by cid, the exception wrongDevice is raised.
*)

DeviceTablePtrValue
PROCEDURE DeviceTablePtrValue (cid: IOChan.ChanId; did: DeviceId): DeviceTablePtr;

(*
   Tests if the device module identified by did is the module
   that made the channel identified by cid.
```

(continues on next page)

(continued from previous page)

```
*)

IsDevice
PROCEDURE IsDevice (cid: IOChan.ChanId; did: DeviceId) : BOOLEAN;

TYPE
DevExceptionRange (type)
  DevExceptionRange = IOChan.ChanExceptions;

(*
  ISO standard states defines

  DevExceptionRange = [IOChan.notAvailable ..  IOChan.textParseError];

  however this must be a bug as other modules need to raise
  IOChan.wrongDevice exceptions.
*)

RAISEdevException
PROCEDURE RAISEdevException (cid: IOChan.ChanId; did: DeviceId;
                             x: DevExceptionRange; s: ARRAY OF CHAR);

  (* If the device module identified by did is not the module that made the channel
     identified by cid, the exception wrongDevice is raised; otherwise the given exception
     is raised, and the string value in s is included in the exception message.
  *)

IsIOException
PROCEDURE IsIOException () : BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising af an exception from ChanExceptions;
     otherwise FALSE.
  *)

IOException
PROCEDURE IOException () : IOChan.ChanExceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     raising af an exception from ChanExceptions, returns the corresponding
     enumeration value, and otherwise raises an exception.
  *)

END IOLink.
```

18.3.15 gm2-libs-iso/IOResult

```
DEFINITION MODULE IOResult;

  (* Read results for specified channels *)

IMPORT IOConsts, IOChan;

TYPE
ReadResults (type)
  ReadResults = IOConsts.ReadResults;

  (*
ReadResults (type)
    ReadResults =  (* This type is used to classify the result of an input operation *)
    (
      notKnown,    (* no read result is set *)
      allRight,    (* data is as expected or as required *)
      outOfRange,  (* data cannot be represented *)
      wrongFormat, (* data not in expected format *)
      endOfLine,   (* end of line seen before expected data *)
      endOfInput   (* end of input seen before expected data *)
    );
  *)

ReadResult
PROCEDURE ReadResult (cid: IOChan.ChanId): ReadResults;
  (* Returns the result for the last read operation on the channel cid. *)

END IOResult.
```

18.3.16 gm2-libs-iso/LongComplexMath

```
DEFINITION MODULE LongComplexMath;

  (* Mathematical functions for the type LONGCOMPLEX *)

CONST
i  (const)
  i =    CMPLX (0.0, 1.0);
one  (const)
  one =  CMPLX (1.0, 0.0);
zero  (const)
  zero = CMPLX (0.0, 0.0);

abs
PROCEDURE abs (z: LONGCOMPLEX): LONGREAL;
  (* Returns the length of z *)

arg
```

(continues on next page)

(continued from previous page)

```
PROCEDURE arg (z: LONGCOMPLEX): LONGREAL;
  (* Returns the angle that z subtends to the positive real axis *)

conj
PROCEDURE conj (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the complex conjugate of z *)

power
PROCEDURE power (base: LONGCOMPLEX; exponent: LONGREAL): LONGCOMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

sqrt
PROCEDURE sqrt (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the principal square root of z *)

exp
PROCEDURE exp (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the complex exponential of z *)

ln
PROCEDURE ln (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

sin
PROCEDURE sin (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the sine of z *)

cos
PROCEDURE cos (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the cosine of z *)

tan
PROCEDURE tan (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the tangent of z *)

arcsin
PROCEDURE arcsin (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arcsine of z *)

arccos
PROCEDURE arccos (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arccosine of z *)

arctan
PROCEDURE arctan (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arctangent of z *)

polarToComplex
PROCEDURE polarToComplex (abs, arg: LONGREAL): LONGCOMPLEX;
  (* Returns the complex number with the specified polar coordinates *)
```

(continues on next page)

(continued from previous page)

```
scalarMult
PROCEDURE scalarMult (scalar: LONGREAL; z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the scalar product of scalar with z *)

IsCMathException
PROCEDURE IsCMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
     returns FALSE.
  *)

END LongComplexMath.
```

18.3.17 gm2-libs-iso/LongConv

```
DEFINITION MODULE LongConv;

  (* Low-level LONGREAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
ConvResults (type)
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange,
                                          strWrongFormat, strEmpty *)

ScanReal
PROCEDURE ScanReal (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                    VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for real
     numbers - assigns class of inputCh to chClass and a procedure
     representing the next state to nextState.
  *)

FormatReal
PROCEDURE FormatReal (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to LONGREAL. *)

ValueReal
PROCEDURE ValueReal (str: ARRAY OF CHAR): LONGREAL;
  (* Returns the value corresponding to the real number string value
     str if str is well-formed; otherwise raises the LongConv exception.
  *)

LengthFloatReal
PROCEDURE LengthFloatReal (real: LONGREAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point string
     representation of real with sigFigs significant figures.
  *)
```

(continues on next page)

(continued from previous page)

```
LengthEngReal
PROCEDURE LengthEngReal (real: LONGREAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point engineering
     string representation of real with sigFigs significant figures.
  *)

LengthFixedReal
PROCEDURE LengthFixedReal (real: LONGREAL; place: INTEGER): CARDINAL;
  (* Returns the number of characters in the fixed-point string
     representation of real rounded to the given place relative to the
     decimal point.
  *)

IsRConvException
PROCEDURE IsRConvException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception in a
     routine from this module; otherwise returns FALSE.
  *)

END LongConv.
```

18.3.18 gm2-libs-iso/LongIO

```
DEFINITION MODULE LongIO;

  (* Input and output of long real numbers in decimal text form
     over specified channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit}, [".",
       {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E", ["+" | "-"], decimal digit, {decimal digit}
  *)

ReadReal
PROCEDURE ReadReal (cid: IOChan.ChanId; VAR real: LONGREAL);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of a signed fixed or floating
     point number.  The value of this number is assigned to real.
     The read result is set to the value allRight, outOfRange,
     wrongFormat, endOfLine, or endOfInput.
```

(continues on next page)

(continued from previous page)

```
  *)

WriteFloat
PROCEDURE WriteFloat (cid: IOChan.ChanId; real: LONGREAL;
                      sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to cid in floating-point text form,
     with sigFigs significant figures, in a field of the given
     minimum width.
  *)

WriteEng
PROCEDURE WriteEng (cid: IOChan.ChanId; real: LONGREAL;
                    sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with
     one to three digits in the whole number part, and with an
     exponent that is a multiple of three.
  *)

WriteFixed
PROCEDURE WriteFixed (cid: IOChan.ChanId; real: LONGREAL;
                      place: INTEGER; width: CARDINAL);
  (* Writes the value of real to cid in fixed-point text form,
     rounded to the given place relative to the decimal point,
     in a field of the given minimum width.
  *)

WriteReal
PROCEDURE WriteReal (cid: IOChan.ChanId; real: LONGREAL;
                     width: CARDINAL);
  (* Writes the value of real to cid, as WriteFixed if the
     sign and magnitude can be shown in the given width, or
     otherwise as WriteFloat.  The number of places or
     significant digits depends on the given width.
  *)

END LongIO.
```

18.3.19 gm2-libs-iso/LongMath

```
DEFINITION MODULE LongMath;

  (* Mathematical functions for the type LONGREAL *)

CONST
pi    (const)
  pi   = 3.14159265358979323846264338327950288419 72;
exp1  (const)
  exp1 = 2.71828182845904523536028747135266249775 72;

sqrt
```

(continues on next page)

(continued from previous page)

```
PROCEDURE __BUILTIN__ sqrt (x: LONGREAL): LONGREAL;
  (* Returns the positive square root of x *)

exp
PROCEDURE __BUILTIN__ exp (x: LONGREAL): LONGREAL;
  (* Returns the exponential of x *)

ln
PROCEDURE __BUILTIN__ ln (x: LONGREAL): LONGREAL;
  (* Returns the natural logarithm of x *)

  (* The angle in all trigonometric functions is measured in radians *)

sin
PROCEDURE __BUILTIN__ sin (x: LONGREAL): LONGREAL;
  (* Returns the sine of x *)

cos
PROCEDURE __BUILTIN__ cos (x: LONGREAL): LONGREAL;
  (* Returns the cosine of x *)

tan
PROCEDURE tan (x: LONGREAL): LONGREAL;
  (* Returns the tangent of x *)

arcsin
PROCEDURE arcsin (x: LONGREAL): LONGREAL;
  (* Returns the arcsine of x *)

arccos
PROCEDURE arccos (x: LONGREAL): LONGREAL;
  (* Returns the arccosine of x *)

arctan
PROCEDURE arctan (x: LONGREAL): LONGREAL;
  (* Returns the arctangent of x *)

power
PROCEDURE power (base, exponent: LONGREAL): LONGREAL;
  (* Returns the value of the number base raised to the power exponent *)

round
PROCEDURE round (x: LONGREAL): INTEGER;
  (* Returns the value of x rounded to the nearest integer *)

IsRMathException
PROCEDURE IsRMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception in a
     routine from this module; otherwise returns FALSE.
  *)
```

(continues on next page)

(continued from previous page)

```
END LongMath.
```

18.3.20 gm2-libs-iso/LongStr

```
DEFINITION MODULE LongStr;

  (* LONGREAL/string conversions *)

IMPORT
   ConvTypes;

TYPE
   (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)
ConvResults (type)
   ConvResults = ConvTypes.ConvResults;

(* the string form of a signed fixed-point real number is
     ["+" | "-"], decimal digit, {decimal digit}, [".",
     {decimal digit}]
*)

(* the string form of a signed floating-point real number is
     signed fixed-point real number, "E", ["+" | "-"],
     decimal digit, {decimal digit}
*)

StrToReal
PROCEDURE StrToReal (str: ARRAY OF CHAR; VAR real: LONGREAL;
                     VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent characters
     in str are in the format of a signed real number, assigns a
     corresponding value to real.  Assigns a value indicating the
     format of str to res.
  *)

RealToFloat
PROCEDURE RealToFloat (real: LONGREAL; sigFigs: CARDINAL;
                       VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with
     sigFigs significant figures, and copies the possibly truncated
     result to str.
  *)

RealToEng
PROCEDURE RealToEng (real: LONGREAL; sigFigs: CARDINAL;
                     VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with
     sigFigs significant figures, and copies the possibly truncated
     result to str. The number is scaled with one to three digits
```

(continues on next page)

(continued from previous page)

```
     in the whole number part and with an exponent that is a
     multiple of three.
  *)

RealToFixed
PROCEDURE RealToFixed (real: LONGREAL; place: INTEGER;
                       VAR str: ARRAY OF CHAR);
  (* Converts the value of real to fixed-point string form, rounded
     to the given place relative to the decimal point, and copies
     the possibly truncated result to str.
  *)

RealToStr
PROCEDURE RealToStr (real: LONGREAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real as RealToFixed if the sign and
     magnitude can be shown within the capacity of str, or
     otherwise as RealToFloat, and copies the possibly truncated
     result to str. The number of places or significant digits
     depend on the capacity of str.
  *)

END LongStr.
```

18.3.21 gm2-libs-iso/LongWholeIO

```
DEFINITION MODULE LongWholeIO;

  (* Input and output of whole numbers in decimal text form
     over specified channels.  The read result is of the
     type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

ReadInt
PROCEDURE ReadInt (cid: IOChan.ChanId; VAR int: LONGINT);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of a signed whole number.  The
     value of this number is assigned to int.  The read result
     is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)
```

(continues on next page)

(continued from previous page)

```
WriteInt
PROCEDURE WriteInt (cid: IOChan.ChanId; int: LONGINT;
                    width: CARDINAL);
  (* Writes the value of int to cid in text form, in a field of
     the given minimum width. *)

ReadCard
PROCEDURE ReadCard (cid: IOChan.ChanId; VAR card: LONGCARD);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of an unsigned whole number.  The
     value of this number is assigned to card. The read result
     is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)

WriteCard
PROCEDURE WriteCard (cid: IOChan.ChanId; card: LONGCARD;
                     width: CARDINAL);
  (* Writes the value of card to cid in text form, in a field
     of the given minimum width. *)

END LongWholeIO.
```

18.3.22 gm2-libs-iso/LowLong

```
DEFINITION MODULE LowLong;

  (* Access to underlying properties of the type LONGREAL *)

CONST
radix       (const)
  radix      = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, radix> )) ;      (* ZType *)
places      (const)
  places     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, places> )) ;     (* ZType *)
expoMin     (const)
  expoMin    = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, expoMin> )) ;    (* ZType *)
expoMax     (const)
  expoMax    = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, expoMax> )) ;    (* ZType *)
large       (const)
  large      = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, large> )) ;      (* RType *)
small       (const)
  small      = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, small> )) ;      (* RType *)
IEC559      (const)
  IEC559     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, IEC559> )) ;     (* BOOLEAN *)
LIA1        (const)
  LIA1       = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, LIA1> )) ;       (* BOOLEAN *)
ISO         (const)
  ISO        = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, ISO> )) ;        (* BOOLEAN *)
IEEE        (const)
  IEEE       = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, IEEE> )) ;       (* BOOLEAN *)
```

(continues on next page)

(continued from previous page)

```
rounds      (const)
  rounds     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, rounds> )) ;     (* BOOLEAN *)
gUnderflow  (const)
  gUnderflow = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, gUnderflow> )) ; (* BOOLEAN *)
exception   (const)
  exception  = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, exception> )) ;  (* BOOLEAN *)
extend      (const)
  extend     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, extend> )) ;     (* BOOLEAN *)
nModes      (const)
  nModes     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, nModes> )) ;     (* ZType *)

TYPE
Modes (type)
  Modes = PACKEDSET OF [0 .. nModes-1];

exponent
PROCEDURE exponent (x: LONGREAL): INTEGER;
  (* Returns the exponent value of x *)

fraction
PROCEDURE fraction (x: LONGREAL): LONGREAL;
  (* Returns the significand (or significant part) of x *)

sign
PROCEDURE sign (x: LONGREAL): LONGREAL;
  (* Returns the signum of x *)

succ
PROCEDURE succ (x: LONGREAL): LONGREAL;
  (* Returns the next value of the type LONGREAL greater than x *)

ulp
PROCEDURE ulp (x: LONGREAL): LONGREAL;
  (* Returns the value of a unit in the last place of x *)

pred
PROCEDURE pred (x: LONGREAL): LONGREAL;
  (* Returns the previous value of the type LONGREAL less than x *)

intpart
PROCEDURE intpart (x: LONGREAL): LONGREAL;
  (* Returns the integer part of x *)

fractpart
PROCEDURE fractpart (x: LONGREAL): LONGREAL;
  (* Returns the fractional part of x *)

scale
PROCEDURE scale (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of x * radix ** n *)
```

(continues on next page)

(continued from previous page)

```
trunc
PROCEDURE trunc (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of the first n places of x *)

round
PROCEDURE round (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of x rounded to the first n places *)

synthesize
PROCEDURE synthesize (expart: INTEGER; frapart: LONGREAL): LONGREAL;
  (* Returns a value of the type LONGREAL constructed from the given expart and frapart *)

setMode
PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type LONGREAL *)

currentMode
PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

IsLowException
PROCEDURE IsLowException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
     returns FALSE.
  *)

END LowLong.
```

18.3.23 gm2-libs-iso/LowReal

```
DEFINITION MODULE LowReal;

  (* Access to underlying properties of the type REAL *)

CONST
radix       (const)
  radix      = __ATTRIBUTE__ __BUILTIN__ (( <REAL, radix> )) ;      (* ZType *)
places      (const)
  places     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, places> )) ;     (* ZType *)
expoMin     (const)
  expoMin    = __ATTRIBUTE__ __BUILTIN__ (( <REAL, expoMin> )) ;    (* ZType *)
expoMax     (const)
  expoMax    = __ATTRIBUTE__ __BUILTIN__ (( <REAL, expoMax> )) ;    (* ZType *)
large       (const)
  large      = __ATTRIBUTE__ __BUILTIN__ (( <REAL, large> )) ;      (* RType *)
small       (const)
  small      = __ATTRIBUTE__ __BUILTIN__ (( <REAL, small> )) ;      (* RType *)
IEC559      (const)
  IEC559     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, IEC559> )) ;     (* BOOLEAN *)
```

(continues on next page)

(continued from previous page)

```
LIA1        (const)
  LIA1       = __ATTRIBUTE__ __BUILTIN__ (( <REAL, LIA1> )) ;       (* BOOLEAN *)
ISO         (const)
  ISO        = __ATTRIBUTE__ __BUILTIN__ (( <REAL, ISO> )) ;        (* BOOLEAN *)
IEEE        (const)
  IEEE       = __ATTRIBUTE__ __BUILTIN__ (( <REAL, IEEE> )) ;       (* BOOLEAN *)
rounds      (const)
  rounds     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, rounds> )) ;     (* BOOLEAN *)
gUnderflow  (const)
  gUnderflow = __ATTRIBUTE__ __BUILTIN__ (( <REAL, gUnderflow> )) ; (* BOOLEAN *)
exception   (const)
  exception  = __ATTRIBUTE__ __BUILTIN__ (( <REAL, exception> )) ;  (* BOOLEAN *)
extend      (const)
  extend     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, extend> )) ;     (* BOOLEAN *)
nModes      (const)
  nModes     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, nModes> )) ;     (* ZType *)

TYPE
Modes (type)
  Modes = PACKEDSET OF [0..nModes-1];

exponent
PROCEDURE exponent (x: REAL): INTEGER;
  (* Returns the exponent value of x *)

fraction
PROCEDURE fraction (x: REAL): REAL;
  (* Returns the significand (or significant part) of x *)

sign
PROCEDURE sign (x: REAL): REAL;
  (* Returns the signum of x *)

succ
PROCEDURE succ (x: REAL): REAL;
  (* Returns the next value of the type REAL greater than x *)

ulp
PROCEDURE ulp (x: REAL): REAL;
  (* Returns the value of a unit in the last place of x *)

pred
PROCEDURE pred (x: REAL): REAL;
  (* Returns the previous value of the type REAL less than x *)

intpart
PROCEDURE intpart (x: REAL): REAL;
  (* Returns the integer part of x *)

fractpart
PROCEDURE fractpart (x: REAL): REAL;
```

(continues on next page)

(continued from previous page)

```
  (* Returns the fractional part of x *)

scale
PROCEDURE scale (x: REAL; n: INTEGER): REAL;
  (* Returns the value of x * radix ** n *)

trunc
PROCEDURE trunc (x: REAL; n: INTEGER): REAL;
  (* Returns the value of the first n places of x *)

round
PROCEDURE round (x: REAL; n: INTEGER): REAL;
  (* Returns the value of x rounded to the first n places *)

synthesize
PROCEDURE synthesize (expart: INTEGER; frapart: REAL): REAL;
  (* Returns a value of the type REAL constructed from the given expart and frapart *)

setMode
PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type REAL *)

currentMode
PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

IsLowException
PROCEDURE IsLowException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
     returns FALSE.
  *)

END LowReal.
```

18.3.24 gm2-libs-iso/LowShort

```
DEFINITION MODULE LowShort;

  (* Access to underlying properties of the type SHORTREAL *)

CONST
radix       (const)
  radix      = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, radix> )) ;      (* ZType *)
places      (const)
  places     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, places> )) ;     (* ZType *)
expoMin     (const)
  expoMin    = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, expoMin> )) ;    (* ZType *)
expoMax     (const)
  expoMax    = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, expoMax> )) ;    (* ZType *)
```

(continues on next page)

(continued from previous page)

```
large       (const)
  large      = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, large> )) ;      (* RType *)
small       (const)
  small      = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, small> )) ;      (* RType *)
IEC559      (const)
  IEC559     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, IEC559> )) ;     (* BOOLEAN *)
LIA1        (const)
  LIA1       = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, LIA1> )) ;       (* BOOLEAN *)
ISO         (const)
  ISO        = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, ISO> )) ;        (* BOOLEAN *)
IEEE        (const)
  IEEE       = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, IEEE> )) ;       (* BOOLEAN *)
rounds      (const)
  rounds     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, rounds> )) ;     (* BOOLEAN *)
gUnderflow  (const)
  gUnderflow = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, gUnderflow> )) ; (* BOOLEAN *)
exception   (const)
  exception  = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, exception> )) ;  (* BOOLEAN *)
extend      (const)
  extend     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, extend> )) ;     (* BOOLEAN *)
nModes      (const)
  nModes     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, nModes> )) ;     (* ZType *)

TYPE
Modes (type)
  Modes = PACKEDSET OF [0 .. nModes-1];

exponent
PROCEDURE exponent (x: SHORTREAL): INTEGER;
  (* Returns the exponent value of x *)

fraction
PROCEDURE fraction (x: SHORTREAL): SHORTREAL;
  (* Returns the significand (or significant part) of x *)

sign
PROCEDURE sign (x: SHORTREAL): SHORTREAL;
  (* Returns the signum of x *)

succ
PROCEDURE succ (x: SHORTREAL): SHORTREAL;
  (* Returns the next value of the type SHORTREAL greater than x *)

ulp
PROCEDURE ulp (x: SHORTREAL): SHORTREAL;
  (* Returns the value of a unit in the last place of x *)

pred
PROCEDURE pred (x: SHORTREAL): SHORTREAL;
  (* Returns the previous value of the type SHORTREAL less than x *)
```

(continues on next page)

(continued from previous page)

```
intpart
PROCEDURE intpart (x: SHORTREAL): SHORTREAL;
  (* Returns the integer part of x *)

fractpart
PROCEDURE fractpart (x: SHORTREAL): SHORTREAL;
  (* Returns the fractional part of x *)

scale
PROCEDURE scale (x: SHORTREAL; n: INTEGER): SHORTREAL;
  (* Returns the value of x * radix ** n *)

trunc
PROCEDURE trunc (x: SHORTREAL; n: INTEGER): SHORTREAL;
  (* Returns the value of the first n places of x *)

round
PROCEDURE round (x: SHORTREAL; n: INTEGER): SHORTREAL;
  (* Returns the value of x rounded to the first n places *)

synthesize
PROCEDURE synthesize (expart: INTEGER; frapart: SHORTREAL): SHORTREAL;
  (* Returns a value of the type SHORTREAL constructed from the given expart and frapart *)

setMode
PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type SHORTREAL *)

currentMode
PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

IsLowException
PROCEDURE IsLowException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
     returns FALSE.
  *)

END LowShort.
```

18.3.25 gm2-libs-iso/M2EXCEPTION

```
DEFINITION MODULE M2EXCEPTION;

(* Provides facilities for identifying language exceptions *)

TYPE
M2Exceptions (type)
  M2Exceptions =
```

(continues on next page)

(continued from previous page)

```
    (indexException,     rangeException,         caseSelectException,  invalidLocation,
     functionException,  wholeValueException,    wholeDivException,    realValueException,
     realDivException,   complexValueException,  complexDivException,  protException,
     sysException,       coException,            exException
    );

M2Exception
PROCEDURE M2Exception (): M2Exceptions;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of a language exception, returns the corresponding enumeration value, and otherwise
     raises an exception.
  *)

IsM2Exception
PROCEDURE IsM2Exception (): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state because of the raising
     of a language exception, returns TRUE, and otherwise returns FALSE.
  *)

END M2EXCEPTION.
```

18.3.26 gm2-libs-iso/M2RTS

```
DEFINITION MODULE M2RTS ;

FROM SYSTEM IMPORT ADDRESS ;

TYPE
ArgCVEnvP (type)
   ArgCVEnvP = PROCEDURE (INTEGER, ADDRESS, ADDRESS) ;

ConstructModules
PROCEDURE ConstructModules (applicationmodule: ADDRESS;
                            argc: INTEGER; argv, envp: ADDRESS) ;

DeconstructModules
PROCEDURE DeconstructModules (applicationmodule: ADDRESS;
                              argc: INTEGER; argv, envp: ADDRESS) ;

(*
   RegisterModule - adds module name to the list of outstanding
                    modules which need to have their dependencies
                    explored to determine initialization order.
*)

RegisterModule
PROCEDURE RegisterModule (name: ADDRESS;
                          init, fini:  ArgCVEnvP;
                          dependencies: PROC) ;
```

(continues on next page)

(continued from previous page)

```
(*
   RequestDependant - used to specify that modulename is dependant upon
                      module dependantmodule.
*)

RequestDependant
PROCEDURE RequestDependant (modulename, dependantmodule: ADDRESS) ;

(*
   ExecuteTerminationProcedures - calls each installed termination
                                  procedure in reverse order.
*)

ExecuteTerminationProcedures
PROCEDURE ExecuteTerminationProcedures ;

(*
   InstallTerminationProcedure - installs a procedure, p, which will
                                 be called when the procedure
                                 ExecuteTerminationProcedures
                                 is invoked.  It returns TRUE is the
                                 procedure is installed.
*)

InstallTerminationProcedure
PROCEDURE InstallTerminationProcedure (p: PROC) : BOOLEAN ;

(*
   ExecuteInitialProcedures - executes the initial procedures installed
                              by InstallInitialProcedure.
*)

ExecuteInitialProcedures
PROCEDURE ExecuteInitialProcedures ;

(*
   InstallInitialProcedure - installs a procedure to be executed just
                             before the BEGIN code section of the main
                             program module.
*)

InstallInitialProcedure
PROCEDURE InstallInitialProcedure (p: PROC) : BOOLEAN ;

(*
   HALT - terminate the current program.  The procedure
          ExecuteTerminationProcedures
          is called before the program is stopped.  The parameter
          exitcode is optional.  If the parameter is not supplied
          HALT will call libc 'abort', otherwise it will exit with
          the code supplied.  Supplying a parameter to HALT has the
```

(continues on next page)

(continued from previous page)

```
          same effect as calling ExitOnHalt with the same code and
          then calling HALT with no parameter.
*)

HALT
PROCEDURE HALT ([exitcode: INTEGER = -1]) ;

(*
   Halt - provides a more user friendly version of HALT, which takes
          four parameters to aid debugging.
*)

Halt
PROCEDURE Halt (file: ARRAY OF CHAR; line: CARDINAL;
                function: ARRAY OF CHAR; description: ARRAY OF CHAR) ;

(*
   ExitOnHalt - if HALT is executed then call exit with the exit code, e.
*)

ExitOnHalt
PROCEDURE ExitOnHalt (e: INTEGER) ;

(*
   ErrorMessage - emits an error message to stderr and then calls exit (1).
*)

ErrorMessage
PROCEDURE ErrorMessage (message: ARRAY OF CHAR;
                        file: ARRAY OF CHAR;
                        line: CARDINAL;
                        function: ARRAY OF CHAR) ;

(*
   IsTerminating - Returns true if any coroutine has started program termination
                   and false otherwise.
*)

IsTerminating
PROCEDURE IsTerminating () : BOOLEAN ;

(*
   HasHalted - Returns true if a call to HALT has been made and false
               otherwise.
*)

HasHalted
PROCEDURE HasHalted () : BOOLEAN ;

(*
   Length - returns the length of a string, a. This is called whenever
```

(continues on next page)

(continued from previous page)

```
            the user calls LENGTH and the parameter cannot be calculated
            at compile time.
*)

Length
PROCEDURE Length (a: ARRAY OF CHAR) : CARDINAL ;

(*
   The following are the runtime exception handler routines.
*)

AssignmentException
PROCEDURE AssignmentException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ReturnException
PROCEDURE ReturnException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
IncException
PROCEDURE IncException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
DecException
PROCEDURE DecException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
InclException
PROCEDURE InclException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
ExclException
PROCEDURE ExclException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
ShiftException
PROCEDURE ShiftException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
RotateException
PROCEDURE RotateException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
StaticArraySubscriptException
PROCEDURE StaticArraySubscriptException (filename: ADDRESS; line, column: CARDINAL; scope,␣
↪message: ADDRESS) ;
DynamicArraySubscriptException
PROCEDURE DynamicArraySubscriptException (filename: ADDRESS; line, column: CARDINAL; scope,␣
↪message: ADDRESS) ;
ForLoopBeginException
PROCEDURE ForLoopBeginException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ForLoopToException
PROCEDURE ForLoopToException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ForLoopEndException
PROCEDURE ForLoopEndException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
PointerNilException
PROCEDURE PointerNilException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
NoReturnException
PROCEDURE NoReturnException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS)␣
↪;
CaseException
PROCEDURE CaseException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;
```

(continues on next page)

(continued from previous page)

```
WholeNonPosDivException
PROCEDURE WholeNonPosDivException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeNonPosModException
PROCEDURE WholeNonPosModException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeZeroDivException
PROCEDURE WholeZeroDivException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeZeroRemException
PROCEDURE WholeZeroRemException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
WholeValueException
PROCEDURE WholeValueException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
RealValueException
PROCEDURE RealValueException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
ParameterException
PROCEDURE ParameterException (filename: ADDRESS; line, column: CARDINAL; scope, message:␣
↪ADDRESS) ;
NoException
PROCEDURE NoException (filename: ADDRESS; line, column: CARDINAL; scope, message: ADDRESS) ;

END M2RTS.
```

18.3.27 gm2-libs-iso/MemStream

```
DEFINITION MODULE MemStream ;

(*
    Description: provides an ISO module which can write to a memory
                 buffer or read from a memory buffer.
*)

FROM IOChan IMPORT ChanId ;
FROM ChanConsts IMPORT FlagSet, OpenResults ;
FROM SYSTEM IMPORT ADDRESS, LOC ;

(*
   Attempts to obtain and open a channel connected to a contigeous
   buffer in memory.  The write flag is implied; without the raw
   flag, text is implied.  If successful, assigns to cid the identity of
   the opened channel, assigns the value opened to res.
   If a channel cannot be opened as required,
   the value of res indicates the reason, and cid identifies the
   invalid channel.

   The parameters, buffer, length and used maybe updated as
   data is written.  The buffer maybe reallocated
```

(continues on next page)

(continued from previous page)

```
   and its address might alter, however the parameters will
   always reflect the current active buffer.  When this
   channel is closed the buffer is deallocated and
   buffer will be set to NIL, length and used will be set to
   zero.
*)

OpenWrite
PROCEDURE OpenWrite (VAR cid: ChanId; flags: FlagSet;
                     VAR res: OpenResults;
                     VAR buffer: ADDRESS;
                     VAR length: CARDINAL;
                     VAR used: CARDINAL;
                     deallocOnClose: BOOLEAN) ;

(*
   Attempts to obtain and open a channel connected to a contigeous
   buffer in memory.  The read and old flags are implied; without
   the raw flag, text is implied.  If successful, assigns to cid the
   identity of the opened channel, assigns the value opened to res, and
   selects input mode, with the read position corresponding to the start
   of the buffer.  If a channel cannot be opened as required, the value of
   res indicates the reason, and cid identifies the invalid channel.
*)

OpenRead
PROCEDURE OpenRead (VAR cid: ChanId; flags: FlagSet;
                    VAR res: OpenResults;
                    buffer: ADDRESS; length: CARDINAL;
                    deallocOnClose: BOOLEAN) ;

(*
   Close - if the channel identified by cid is not open to
           a memory stream, the exception wrongDevice is
           raised; otherwise closes the channel, and assigns
           the value identifying the invalid channel to cid.
*)

Close
PROCEDURE Close (VAR cid: ChanId) ;

(*
   Rewrite - assigns the buffer index to zero.  Subsequent
             writes will overwrite the previous buffer contents.
*)

Rewrite
PROCEDURE Rewrite (cid: ChanId) ;

(*
   Reread - assigns the buffer index to zero.  Subsequent
```

(continues on next page)

(continued from previous page)

```
            reads will read the previous buffer contents.
*)

Reread
PROCEDURE Reread (cid: ChanId) ;

(*
   IsMem - tests if the channel identified by cid is open as
           a memory stream.
*)

IsMem
PROCEDURE IsMem (cid: ChanId) : BOOLEAN ;

END MemStream.
```

18.3.28 gm2-libs-iso/Preemptive

```
DEFINITION MODULE Preemptive ;

(*
   initPreemptive - if microsecs > 0 then turn on preemptive scheduling.
                    if microsecs = 0 then preemptive scheduling is turned off.
*)

initPreemptive
PROCEDURE initPreemptive (seconds, microsecs: CARDINAL) ;

END Preemptive.
```

18.3.29 gm2-libs-iso/Processes

```
DEFINITION MODULE Processes;

  (* This module allows concurrent algorithms to be expressed using
     processes. A process is a unit of a program that has the
     potential to run in parallel with other processes.
  *)

IMPORT SYSTEM;

TYPE
  ProcessId;                    (* Used to identify processes *)
Parameter (type)
  Parameter     = SYSTEM.ADDRESS; (* Used to pass data between processes *)
Body (type)
  Body          = PROC;         (* Used as the type of a process body *)
```

(continues on next page)

(continued from previous page)

```
Urgency (type)
  Urgency       = INTEGER;        (* Used by the internal scheduler *)
Sources (type)
  Sources       = CARDINAL;       (* Used to identify event sources *)
ProcessesExceptions (type)
  ProcessesExceptions =           (* Exceptions raised by this module *)
    (passiveProgram, processError);

(* The following procedures create processes and switch control between
   them. *)

Create
PROCEDURE Create (procBody: Body; extraSpace: CARDINAL; procUrg: Urgency;
                  procParams: Parameter; VAR procId: ProcessId);
  (* Creates a new process with procBody as its body, and with urgency
     and parameters given by procUrg and procParams.  At least as
     much workspace (in units of SYSTEM.LOC) as is specified by
     extraSpace is allocated to the process.
     An identity for the new process is returned in procId.
     The process is created in the passive state; it will not run
     until activated.
  *)

Start
PROCEDURE Start (procBody: Body; extraSpace: CARDINAL; procUrg: Urgency;
                 procParams: Parameter; VAR procId: ProcessId);
  (* Creates a new process, with parameters as for Create.
     The process is created in the ready state; it is eligible to
     run immediately.
  *)

StopMe
PROCEDURE StopMe ();
  (* Terminates the calling process.
     The process must not be associated with a source of events.
  *)

SuspendMe
PROCEDURE SuspendMe ();
  (* Causes the calling process to enter the passive state.  The
     procedure only returns when the calling process is again
     activated by another process.
  *)

Activate
PROCEDURE Activate (procId: ProcessId);
  (* Causes the process identified by procId to enter the ready
     state, and thus to become eligible to run again.
  *)

SuspendMeAndActivate
```

(continues on next page)

(continued from previous page)

```
PROCEDURE SuspendMeAndActivate (procId: ProcessId);
  (* Executes an atomic sequence of SuspendMe() and
     Activate(procId). *)

Switch
PROCEDURE Switch (procId: ProcessId; VAR info: Parameter);
  (* Causes the calling process to enter the passive state; the
     process identified by procId becomes the currently executing
     process.  info is used to pass parameter information from the
     calling to the activated process.  On return, info will
     contain information from the process that chooses to switch
     back to this one (or will be NIL if Activate or
     SuspendMeAndActivate are used instead of Switch).
  *)

Wait
PROCEDURE Wait ();
  (* Causes the calling process to enter the waiting state.
     The procedure will return when the calling process is
     activated by another process, or when one of its associated
     eventSources has generated an event.
  *)

(* The following procedures allow the association of processes
   with sources of external events.
*)

Attach
PROCEDURE Attach (eventSource: Sources);
  (* Associates the specified eventSource with the calling
     process. *)

Detach
PROCEDURE Detach (eventSource: Sources);
  (* Dissociates the specified eventSource from the program. *)

IsAttached
PROCEDURE IsAttached (eventSource: Sources): BOOLEAN;
  (* Returns TRUE if and only if the specified eventSource is
     currently associated with one of the processes of the
     program.
  *)

Handler
PROCEDURE Handler (eventSource: Sources): ProcessId;
  (* Returns the identity of the process, if any, that is
     associated with the specified eventSource.
  *)

(* The following procedures allow processes to obtain their
   identity, parameters, and urgency.
```

(continues on next page)

(continued from previous page)

```
*)

Me
PROCEDURE Me (): ProcessId;
  (* Returns the identity of the calling process (as assigned
     when the process was first created).
  *)

MyParam
PROCEDURE MyParam (): Parameter;
  (* Returns the value specified as procParams when the calling
     process was created. *)

UrgencyOf
PROCEDURE UrgencyOf (procId: ProcessId): Urgency;
  (* Returns the urgency established when the process identified
     by procId was first created.
  *)

(* The following procedure provides facilities for exception
   handlers. *)

ProcessesException
PROCEDURE ProcessesException (): ProcessesExceptions;
  (* If the current coroutine is in the exceptional execution state
     because of the raising of a language exception, returns the
     corresponding enumeration value, and otherwise raises an
     exception.
  *)

IsProcessesException
PROCEDURE IsProcessesException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception in
     a routine from this module; otherwise returns FALSE.
  *)

(*
   Reschedule - rotates the ready queue and transfers to the process
                with the highest run priority.
*)

Reschedule
PROCEDURE Reschedule ;

(*
   displayProcesses -
*)

displayProcesses
PROCEDURE displayProcesses (message: ARRAY OF CHAR) ;
```

(continues on next page)

(continued from previous page)

```
END Processes.
```

18.3.30 gm2-libs-iso/ProgramArgs

```
DEFINITION MODULE ProgramArgs;

  (* Access to program arguments *)

IMPORT IOChan;

TYPE
ChanId (type)
  ChanId = IOChan.ChanId;

ArgChan
PROCEDURE ArgChan (): ChanId;
  (* Returns a value that identifies a channel for reading
     program arguments *)

IsArgPresent
PROCEDURE IsArgPresent (): BOOLEAN;
  (* Tests if there is a current argument to read from.  If not,
     read <= IOChan.CurrentFlags() will be FALSE, and attempting
     to read from the argument channel will raise the exception
     notAvailable.
  *)

NextArg
PROCEDURE NextArg ();
  (* If there is another argument, causes subsequent input from the
     argument device to come from the start of the next argument.
     Otherwise there is no argument to read from, and a call of
     IsArgPresent will return FALSE.
  *)

END ProgramArgs.
```

18.3.31 gm2-libs-iso/RTco

```
DEFINITION MODULE RTco ;

FROM SYSTEM IMPORT ADDRESS ;

(* init initializes the module and allows the application to lazily invoke threads.  *)

init
```

(continues on next page)

(continued from previous page)

```
PROCEDURE init () : INTEGER ;

initThread
PROCEDURE initThread (p: PROC; stackSize: CARDINAL; interruptLevel: CARDINAL) : INTEGER ;

initSemaphore
PROCEDURE initSemaphore (value: CARDINAL) : INTEGER ;

wait
PROCEDURE wait (semaphore: INTEGER) ;

signal
PROCEDURE signal (semaphore: INTEGER) ;

transfer
PROCEDURE transfer (VAR p1: INTEGER; p2: INTEGER) ;

waitThread
PROCEDURE waitThread (tid: INTEGER) ;

signalThread
PROCEDURE signalThread (tid: INTEGER) ;

currentThread
PROCEDURE currentThread () : INTEGER ;

(* currentInterruptLevel returns the interrupt level of the current thread.  *)

currentInterruptLevel
PROCEDURE currentInterruptLevel () : CARDINAL ;

(* turninterrupts returns the old interrupt level and assigns the interrupt level
   to newLevel.  *)

turnInterrupts
PROCEDURE turnInterrupts (newLevel: CARDINAL) : CARDINAL ;

(*
   select access to the select system call which will be thread safe.
   This is typically called from the idle process to wait for an interrupt.
*)

select
PROCEDURE select (p1: INTEGER;
                  p2: ADDRESS;
                  p3: ADDRESS;
                  p4: ADDRESS;
                  p5: ADDRESS) : INTEGER ;

END RTco.
```

18.3.32 gm2-libs-iso/RTdata

```
DEFINITION MODULE RTdata ;

(*
    Description: provides a mechanism whereby devices can store
                 data attached to a device.
*)

FROM SYSTEM IMPORT ADDRESS ;
FROM IOLink IMPORT DeviceTablePtr ;

TYPE
ModuleId (type)
   ModuleId ;
FreeProcedure (type)
   FreeProcedure = PROCEDURE (ADDRESS) ;

(*
   MakeModuleId - creates a unique module Id.
*)

MakeModuleId
PROCEDURE MakeModuleId (VAR m: ModuleId) ;

(*
   InitData - adds, datum, to the device, d.  The datum
              is associated with ModuleID, m.
*)

InitData
PROCEDURE InitData (d: DeviceTablePtr; m: ModuleId;
                    datum: ADDRESS; f: FreeProcedure) ;

(*
   GetData - returns the datum assocated with ModuleId, m.
*)

GetData
PROCEDURE GetData (d: DeviceTablePtr; m: ModuleId) : ADDRESS ;

(*
   KillData - destroys the datum associated with ModuleId, m,
              in device, d.  It invokes the free procedure
              given during InitData.
*)

KillData
PROCEDURE KillData (d: DeviceTablePtr; m: ModuleId) ;

END RTdata.
```

18.3.33 gm2-libs-iso/RTentity

```
DEFINITION MODULE RTentity ;

(*
    Description: provides a set of routines for maintaining an
                 efficient mechanism to group opaque (or pointer)
                 data structures together.  Internally the
                 entities are grouped together using a binary
                 tree.  It does not use Storage - and instead
                 uses malloc, free from libc as Storage uses the
                 module to detect erroneous deallocations.
*)

IMPORT SYSTEM ;

TYPE
Group (type)
   Group ;

InitGroup
PROCEDURE InitGroup () : Group ;
KillGroup
PROCEDURE KillGroup (g: Group) : Group ;
GetKey
PROCEDURE GetKey (g: Group; a: SYSTEM.ADDRESS) : CARDINAL ;
PutKey
PROCEDURE PutKey (g: Group; a: SYSTEM.ADDRESS; key: CARDINAL) ;
DelKey
PROCEDURE DelKey (g: Group; a: SYSTEM.ADDRESS) ;
IsIn
PROCEDURE IsIn (g: Group; a: SYSTEM.ADDRESS) : BOOLEAN ;

END RTentity.
```

18.3.34 gm2-libs-iso/RTfio

```
DEFINITION MODULE RTfio ;

(*
    Description: provides default FIO based methods for the RTgenif
                 procedures.  These will be used by StreamFile,
                 SeqFile, StdChans, TermFile and RndFile.
*)

FROM SYSTEM IMPORT ADDRESS ;
FROM IOLink IMPORT DeviceTablePtr;
FROM RTgenif IMPORT GenDevIF ;

(*
```

(continues on next page)

(continued from previous page)

```
   doreadchar - returns a CHAR from the file associated with, g.
*)

doreadchar
PROCEDURE doreadchar (g: GenDevIF; d: DeviceTablePtr) : CHAR ;

(*
   dounreadchar - pushes a CHAR back onto the file associated
                  with, g.
*)

dounreadchar
PROCEDURE dounreadchar (g: GenDevIF; d: DeviceTablePtr; ch: CHAR) : CHAR ;

(*
   dogeterrno - returns the errno relating to the generic device.
*)

dogeterrno
PROCEDURE dogeterrno (g: GenDevIF; d: DeviceTablePtr) : INTEGER ;

(*
   dorbytes - reads upto, max, bytes setting, actual, and
              returning FALSE if an error (not due to eof)
              occurred.
*)

dorbytes
PROCEDURE dorbytes (g: GenDevIF;
                    d: DeviceTablePtr;
                    to: ADDRESS;
                    max: CARDINAL;
                    VAR actual: CARDINAL) : BOOLEAN ;

(*
   dowbytes - writes up to, nBytes.  It returns FALSE
              if an error occurred and it sets actual
              to the amount of data written.
*)

dowbytes
PROCEDURE dowbytes (g: GenDevIF;
                    d: DeviceTablePtr;
                    from: ADDRESS;
                    nBytes: CARDINAL;
                    VAR actual: CARDINAL) : BOOLEAN ;

(*
   dowriteln - attempt to write an end of line marker to the
               file and returns TRUE if successful.
*)
```

(continues on next page)

(continued from previous page)

```
dowriteln
PROCEDURE dowriteln (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   iseof - returns TRUE if end of file has been seen.
*)

iseof
PROCEDURE iseof (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   iseoln - returns TRUE if end of line has been seen.
*)

iseoln
PROCEDURE iseoln (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   iserror - returns TRUE if an error was seen on the device.
             Note that reaching EOF is not classified as an
             error.
*)

iserror
PROCEDURE iserror (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

END RTfio.
```

18.3.35 gm2-libs-iso/RTgen

```
DEFINITION MODULE RTgen ;

(*
    Description: provides a generic device interface between
                 ISO channels and the underlying PIM style
                 FIO procedure calls.
*)

FROM RTgenif IMPORT GenDevIF ;
FROM IOLink IMPORT DeviceId, DeviceTablePtr;
FROM IOConsts IMPORT ReadResults ;
FROM SYSTEM IMPORT ADDRESS ;

TYPE
ChanDev (type)
   ChanDev ;
DeviceType (type)
   DeviceType = (seqfile, streamfile, programargs, stdchans, term, socket, rndfile) ;
```

(continues on next page)

(continued from previous page)

```
(*
   InitChanDev - initialize and return a ChanDev.
*)

InitChanDev
PROCEDURE InitChanDev (t: DeviceType; d: DeviceId; g: GenDevIF) : ChanDev ;

(*
   KillChanDev - deallocates, g.
*)

KillChanDev
PROCEDURE KillChanDev (g: GenDevIF) : GenDevIF ;

(*
   RaiseEOFinLook - returns TRUE if the Look procedure
                    should raise an exception if it
                    sees end of file.
*)

RaiseEOFinLook
PROCEDURE RaiseEOFinLook (g: ChanDev) : BOOLEAN ;

(*
   RaiseEOFinSkip - returns TRUE if the Skip procedure
                    should raise an exception if it
                    sees end of file.
*)

RaiseEOFinSkip
PROCEDURE RaiseEOFinSkip (g: ChanDev) : BOOLEAN ;

doLook
PROCEDURE doLook (g: ChanDev;
                  d: DeviceTablePtr;
                  VAR ch: CHAR;
                  VAR r: ReadResults) ;

doSkip
PROCEDURE doSkip (g: ChanDev;
                  d: DeviceTablePtr) ;

doSkipLook
PROCEDURE doSkipLook (g: ChanDev;
                      d: DeviceTablePtr;
                      VAR ch: CHAR;
                      VAR r: ReadResults) ;

doWriteLn
PROCEDURE doWriteLn (g: ChanDev;
                     d: DeviceTablePtr) ;
```

(continues on next page)

(continued from previous page)

```
doReadText
PROCEDURE doReadText (g: ChanDev;
                      d: DeviceTablePtr;
                      to: ADDRESS;
                      maxChars: CARDINAL;
                      VAR charsRead: CARDINAL) ;

doWriteText
PROCEDURE doWriteText (g: ChanDev;
                       d: DeviceTablePtr;
                       from: ADDRESS;
                       charsToWrite: CARDINAL) ;

doReadLocs
PROCEDURE doReadLocs (g: ChanDev;
                      d: DeviceTablePtr;
                      to: ADDRESS;
                      maxLocs: CARDINAL;
                      VAR locsRead: CARDINAL) ;

doWriteLocs
PROCEDURE doWriteLocs (g: ChanDev;
                       d: DeviceTablePtr;
                       from: ADDRESS;
                       locsToWrite: CARDINAL) ;

(*
   checkErrno - checks a number of errno conditions and raises
                appropriate ISO exceptions if they occur.
*)

checkErrno
PROCEDURE checkErrno (g: ChanDev; d: DeviceTablePtr) ;

END RTgen.
```

18.3.36 gm2-libs-iso/RTgenif

```
DEFINITION MODULE RTgenif ;

(*
    Description: provides a generic interface mechanism used
                 by RTgen.  This is not an ISO module but rather
                 a runtime support module.
*)

FROM SYSTEM IMPORT ADDRESS ;
FROM IOLink IMPORT DeviceId, DeviceTablePtr ;
```

(continues on next page)

(continued from previous page)

```
TYPE
GenDevIF (type)
   GenDevIF ;
readchar (type)
   readchar   = PROCEDURE (GenDevIF, DeviceTablePtr) : CHAR ;
unreadchar (type)
   unreadchar = PROCEDURE (GenDevIF, DeviceTablePtr, CHAR) : CHAR ;
geterrno (type)
   geterrno   = PROCEDURE (GenDevIF, DeviceTablePtr) : INTEGER ;
readbytes (type)
   readbytes  = PROCEDURE (GenDevIF, DeviceTablePtr, ADDRESS, CARDINAL, VAR CARDINAL) : BOOLEAN ;
writebytes (type)
   writebytes = PROCEDURE (GenDevIF, DeviceTablePtr, ADDRESS, CARDINAL, VAR CARDINAL) : BOOLEAN ;
writeln (type)
   writeln    = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;
iseof (type)
   iseof      = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;
iseoln (type)
   iseoln     = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;
iserror (type)
   iserror    = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;

(*
   InitGenDevIF - initializes a generic device.
*)

InitGenDevIF
PROCEDURE InitGenDevIF (d     : DeviceId;
                        rc    : readchar;
                        urc   : unreadchar;
                        geterr: geterrno;
                        rbytes: readbytes;
                        wbytes: writebytes;
                        wl    : writeln;
                        eof   : iseof;
                        eoln  : iseoln;
                        iserr : iserror) : GenDevIF ;

(*
   getDID - returns the device id this generic interface.
*)

getDID
PROCEDURE getDID (g: GenDevIF) : DeviceId ;

(*
   doReadChar - returns the next character from the generic
                device.
*)

doReadChar
```

(continues on next page)

(continued from previous page)

```
PROCEDURE doReadChar (g: GenDevIF; d: DeviceTablePtr) : CHAR ;

(*
   doUnReadChar - pushes back a character to the generic device.
*)

doUnReadChar
PROCEDURE doUnReadChar (g: GenDevIF; d: DeviceTablePtr; ch: CHAR) : CHAR ;

(*
   doGetErrno - returns the errno relating to the generic device.
*)

doGetErrno
PROCEDURE doGetErrno (g: GenDevIF; d: DeviceTablePtr) : INTEGER ;

(*
   doRBytes - attempts to read, n, bytes from the generic device.
              It set the actual amount read and returns a boolean
              to determine whether an error occurred.
*)

doRBytes
PROCEDURE doRBytes (g: GenDevIF; d: DeviceTablePtr;
                    to: ADDRESS; max: CARDINAL;
                    VAR actual: CARDINAL) : BOOLEAN ;

(*
   doWBytes - attempts to write, n, bytes to the generic device.
              It sets the actual amount written and returns a
              boolean to determine whether an error occurred.
*)

doWBytes
PROCEDURE doWBytes (g: GenDevIF; d: DeviceTablePtr;
                    from: ADDRESS; max: CARDINAL;
                    VAR actual: CARDINAL) : BOOLEAN ;

(*
   doWrLn - writes an end of line marker and returns
            TRUE if successful.
*)

doWrLn
PROCEDURE doWrLn (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   isEOF - returns true if the end of file was reached.
*)

isEOF
```

(continues on next page)

(continued from previous page)

```
PROCEDURE isEOF (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   isEOLN - returns true if the end of line was reached.
*)

isEOLN
PROCEDURE isEOLN (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   isError - returns true if an error was seen in the device.
*)

isError
PROCEDURE isError (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;

(*
   KillGenDevIF - deallocates a generic device.
*)

KillGenDevIF
PROCEDURE KillGenDevIF (g: GenDevIF) : GenDevIF ;

END RTgenif.
```

18.3.37 gm2-libs-iso/RTio

```
DEFINITION MODULE RTio ;

(*
    Description: provides low level routines for creating and destroying
                 ChanIds.  This is necessary to allow multiple modules
                 to create, ChanId values, where ChanId is an opaque
                 type.
*)

IMPORT FIO, IOLink ;

TYPE
ChanId (type)
   ChanId ;

(*
   InitChanId - return a new ChanId.
*)

InitChanId
PROCEDURE InitChanId () : ChanId ;

(*
```

(continues on next page)

(continued from previous page)

```
   KillChanId - deallocate a ChanId.
*)

KillChanId
PROCEDURE KillChanId (c: ChanId) : ChanId ;

(*
   NilChanId - return a NIL pointer.
*)

NilChanId
PROCEDURE NilChanId () : ChanId ;

(*
   GetDeviceId - returns the device id, from, c.
*)

GetDeviceId
PROCEDURE GetDeviceId (c: ChanId) : IOLink.DeviceId ;

(*
   SetDeviceId - sets the device id in, c.
*)

SetDeviceId
PROCEDURE SetDeviceId (c: ChanId; d: IOLink.DeviceId) ;

(*
   GetDevicePtr - returns the device table ptr, from, c.
*)

GetDevicePtr
PROCEDURE GetDevicePtr (c: ChanId) : IOLink.DeviceTablePtr ;

(*
   SetDevicePtr - sets the device table ptr in, c.
*)

SetDevicePtr
PROCEDURE SetDevicePtr (c: ChanId; p: IOLink.DeviceTablePtr) ;

(*
   GetFile - returns the file field from, c.
*)

GetFile
PROCEDURE GetFile (c: ChanId) : FIO.File ;

(*
   SetFile - sets the file field in, c.
*)
```

(continues on next page)

(continued from previous page)

```
SetFile
PROCEDURE SetFile (c: ChanId; f: FIO.File) ;

END RTio.
```

18.3.38 gm2-libs-iso/RandomNumber

```
DEFINITION MODULE RandomNumber ;

(*
    Description: provides primitives for obtaining random numbers on
                 pervasive data types.
*)

FROM SYSTEM IMPORT BYTE ;
EXPORT QUALIFIED Randomize, RandomInit, RandomBytes,
                 RandomCard, RandomShortCard, RandomLongCard,
                 RandomInt, RandomShortInt, RandomLongInt,
                 RandomReal, RandomLongReal, RandomShortReal ;

(*
   Randomize - initialize the random number generator with a seed
               based on the microseconds.
*)

Randomize
PROCEDURE Randomize ;

(*
   RandomInit - initialize the random number generator with value, seed.
*)

RandomInit
PROCEDURE RandomInit (seed: CARDINAL) ;

(*
   RandomBytes - fills in an array with random values.
*)

RandomBytes
PROCEDURE RandomBytes (VAR a: ARRAY OF BYTE) ;

(*
   RandomInt - return an INTEGER in the range [low .. high].
*)

RandomInt
PROCEDURE RandomInt (low, high: INTEGER) : INTEGER ;
```

(continues on next page)

(continued from previous page)

```
(*
   RandomShortInt - return an SHORTINT in the range [low..high].
*)

RandomShortInt
PROCEDURE RandomShortInt (low, high: SHORTINT) : SHORTINT ;

(*
   RandomLongInt - return an LONGINT in the range [low..high].
*)

RandomLongInt
PROCEDURE RandomLongInt (low, high: LONGINT) : LONGINT ;

(*
   RandomShortCard - return a SHORTCARD in the range [low..high].
*)

RandomShortCard
PROCEDURE RandomShortCard (low, high: CARDINAL) : CARDINAL ;

(*
   RandomCard - return a CARDINAL in the range [low..high].
*)

RandomCard
PROCEDURE RandomCard (low, high: CARDINAL) : CARDINAL ;

(*
   RandomLongCard - return an LONGCARD in the range [low..high].
*)

RandomLongCard
PROCEDURE RandomLongCard (low, high: LONGCARD) : LONGCARD ;

(*
   RandomReal - return a REAL number in the range 0.0..1.0
*)

RandomReal
PROCEDURE RandomReal () : REAL ;

(*
   RandomShortReal - return a SHORTREAL number in the range 0.0..1.0
*)

RandomShortReal
PROCEDURE RandomShortReal () : SHORTREAL ;

(*
   RandomLongReal - return a LONGREAL number in the range 0.0..1.0
```

(continues on next page)

(continued from previous page)

```
*)

RandomLongReal
PROCEDURE RandomLongReal () : LONGREAL ;

END RandomNumber.
```

18.3.39 gm2-libs-iso/RawIO

```
DEFINITION MODULE RawIO;

  (* Reading and writing data over specified channels using raw
     operations, that is, with no conversion or interpretation.
     The read result is of the type IOConsts.ReadResults.
  *)

IMPORT IOChan, SYSTEM;

Read
PROCEDURE Read (cid: IOChan.ChanId; VAR to: ARRAY OF SYSTEM.LOC);
  (* Reads storage units from cid, and assigns them to
     successive components of to. The read result is set
     to the value allRight, wrongFormat, or endOfInput.
  *)

Write
PROCEDURE Write (cid: IOChan.ChanId; from: ARRAY OF SYSTEM.LOC);
  (* Writes storage units to cid from successive components
     of from. *)

END RawIO.
```

18.3.40 gm2-libs-iso/RealConv

```
DEFINITION MODULE RealConv;

  (* Low-level REAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)
ConvResults (type)
  ConvResults = ConvTypes.ConvResults;

ScanReal
```

(continues on next page)

(continued from previous page)

```
PROCEDURE ScanReal (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                    VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for real
     numbers - assigns class of inputCh to chClass and a procedure
     representing the next state to nextState.
   *)

FormatReal
PROCEDURE FormatReal (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to REAL. *)

ValueReal
PROCEDURE ValueReal (str: ARRAY OF CHAR): REAL;
  (* Returns the value corresponding to the real number string value
     str if str is well-formed; otherwise raises the RealConv
     exception.
  *)

LengthFloatReal
PROCEDURE LengthFloatReal (real: REAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point string
     representation of real with sigFigs significant figures.
  *)

LengthEngReal
PROCEDURE LengthEngReal (real: REAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point engineering
     string representation of real with sigFigs significant figures.
  *)

LengthFixedReal
PROCEDURE LengthFixedReal (real: REAL; place: INTEGER): CARDINAL;
  (* Returns the number of characters in the fixed-point string
     representation of real rounded to the given place relative to the
     decimal point.
  *)

IsRConvException
PROCEDURE IsRConvException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception in a
     routine from this module; otherwise returns FALSE.
  *)

END RealConv.
```

18.3.41 gm2-libs-iso/RealIO

```
DEFINITION MODULE RealIO;

  (* Input and output of real numbers in decimal text form
     over specified channels.  The read result is of the
     type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E", ["+" | "-"], decimal digit, {decimal digit}
  *)

ReadReal
PROCEDURE ReadReal (cid: IOChan.ChanId; VAR real: REAL);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of a signed fixed or floating
     point number.  The value of this number is assigned to real.
     The read result is set to the value allRight, outOfRange,
     wrongFormat, endOfLine, or endOfInput.
  *)

WriteFloat
PROCEDURE WriteFloat (cid: IOChan.ChanId; real: REAL;
                      sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to cid in floating-point text form,
     with sigFigs significant figures, in a field of the given
     minimum width.
  *)

WriteEng
PROCEDURE WriteEng (cid: IOChan.ChanId; real: REAL;
                    sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with
     one to three digits in the whole number part, and with an
     exponent that is a multiple of three.
  *)

WriteFixed
PROCEDURE WriteFixed (cid: IOChan.ChanId; real: REAL;
                      place: INTEGER; width: CARDINAL);
  (* Writes the value of real to cid in fixed-point text form,
     rounded to the given place relative to the decimal point,
     in a field of the given minimum width.
  *)
```

(continues on next page)

(continued from previous page)

```
WriteReal
PROCEDURE WriteReal (cid: IOChan.ChanId;
                     real: REAL; width: CARDINAL);
  (* Writes the value of real to cid, as WriteFixed if the sign
     and magnitude can be shown in the given width, or otherwise
     as WriteFloat.  The number of places or significant digits
     depends on the given width.
  *)

END RealIO.
```

18.3.42 gm2-libs-iso/RealMath

```
DEFINITION MODULE RealMath;

  (* Mathematical functions for the type REAL *)

CONST
pi    (const)
  pi   = 3.14159265358979323846264338327950288419 72;
exp1  (const)
  exp1 = 2.71828182845904523536028747135266249775 72;

sqrt
PROCEDURE __BUILTIN__ sqrt (x: REAL): REAL;
  (* Returns the positive square root of x *)

exp
PROCEDURE __BUILTIN__ exp (x: REAL): REAL;
  (* Returns the exponential of x *)

ln
PROCEDURE __BUILTIN__ ln (x: REAL): REAL;
  (* Returns the natural logarithm of x *)

  (* The angle in all trigonometric functions is measured in radians *)

sin
PROCEDURE __BUILTIN__ sin (x: REAL): REAL;
  (* Returns the sine of x *)

cos
PROCEDURE __BUILTIN__ cos (x: REAL): REAL;
  (* Returns the cosine of x *)

tan
PROCEDURE tan (x: REAL): REAL;
  (* Returns the tangent of x *)
```

(continues on next page)

(continued from previous page)

```
arcsin
PROCEDURE arcsin (x: REAL): REAL;
  (* Returns the arcsine of x *)

arccos
PROCEDURE arccos (x: REAL): REAL;
  (* Returns the arccosine of x *)

arctan
PROCEDURE arctan (x: REAL): REAL;
  (* Returns the arctangent of x *)

power
PROCEDURE power (base, exponent: REAL) : REAL;
  (* Returns the value of the number base raised to the power exponent *)

round
PROCEDURE round (x: REAL) : INTEGER;
  (* Returns the value of x rounded to the nearest integer *)

IsRMathException
PROCEDURE IsRMathException () : BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
     returns FALSE.
  *)

END RealMath.
```

18.3.43 gm2-libs-iso/RealStr

```
DEFINITION MODULE RealStr;

  (* REAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)
ConvResults (type)
  ConvResults = ConvTypes.ConvResults;

(* the string form of a signed fixed-point real number is
     ["+" | "-"], decimal digit, {decimal digit}, [".",
     {decimal digit}]
*)

(* the string form of a signed floating-point real number is
     signed fixed-point real number, "E", ["+" | "-"],
```

(continues on next page)

(continued from previous page)

```
     decimal digit, {decimal digit}
*)

StrToReal
PROCEDURE StrToReal (str: ARRAY OF CHAR; VAR real: REAL;
                     VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent characters
     in str are in the format of a signed real number, assigns a
     corresponding value to real.  Assigns a value indicating the
     format of str to res.
  *)

RealToFloat
PROCEDURE RealToFloat (real: REAL; sigFigs: CARDINAL;
                       VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with
     sigFigs significant figures, and copies the possibly truncated
     result to str.
  *)

RealToEng
PROCEDURE RealToEng (real: REAL; sigFigs: CARDINAL;
                     VAR str: ARRAY OF CHAR);
  (* Converts the value of real to floating-point string form, with
     sigFigs significant figures, and copies the possibly truncated
     result to str.  The number is scaled with one to three digits
     in the whole number part and with an exponent that is a multiple
     of three.
  *)

RealToFixed
PROCEDURE RealToFixed (real: REAL; place: INTEGER;
                       VAR str: ARRAY OF CHAR);
  (* Converts the value of real to fixed-point string form, rounded
     to the given place relative to the decimal point, and copies
     the possibly truncated result to str.
  *)

RealToStr
PROCEDURE RealToStr (real: REAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of real as RealToFixed if the sign and
     magnitude can be shown within the capacity of str, or
     otherwise as RealToFloat, and copies the possibly truncated
     result to str. The number of places or significant digits are
     implementation-defined.
  *)

END RealStr.
```

18.3.44 gm2-libs-iso/RndFile

```
DEFINITION MODULE RndFile;

  (* Random access files *)

IMPORT IOChan, ChanConsts, SYSTEM;

TYPE
ChanId (type)
   ChanId = IOChan.ChanId;
FlagSet (type)
   FlagSet = ChanConsts.FlagSet;
OpenResults (type)
   OpenResults = ChanConsts.OpenResults;

   (* Accepted singleton values of FlagSet *)

CONST
   (* input operations are requested/available *)
read  (const)
   read = FlagSet{ChanConsts.readFlag};
   (* output operations are requested/available *)
write  (const)
   write = FlagSet{ChanConsts.writeFlag};
   (* a file may/must/did exist before the channel is opened *)
old  (const)
   old = FlagSet{ChanConsts.oldFlag};
   (* text operations are requested/available *)
text  (const)
   text = FlagSet{ChanConsts.textFlag};
   (* raw operations are requested/available *)
raw  (const)
   raw = FlagSet{ChanConsts.rawFlag};

OpenOld
PROCEDURE OpenOld (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                   VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored random
     access file of the given name.
     The old flag is implied; without the write flag, read is implied;
     without the text flag, raw is implied.
     If successful, assigns to cid the identity of the opened channel,
     assigns the value opened to res, and sets the read/write position
     to the start of the file.
     If a channel cannot be opened as required, the value of res indicates
     the reason, and cid identifies the invalid channel.
  *)

OpenClean
PROCEDURE OpenClean (VAR cid: ChanId; name: ARRAY OF CHAR; flags: FlagSet;
                     VAR res: OpenResults);
```

(continues on next page)

(continued from previous page)

```
  (* Attempts to obtain and open a channel connected to a stored random
     access file of the given name.
     The write flag is implied; without the text flag, raw is implied.
     If successful, assigns to cid the identity of the opened channel,
     assigns the value opened to res, and truncates the file to zero length.
     If a channel cannot be opened as required, the value of res indicates
     the reason, and cid identifies the invalid channel.
  *)

IsRndFile
PROCEDURE IsRndFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a random access file. *)

IsRndFileException
PROCEDURE IsRndFileException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution
     state because of the raising of a RndFile exception; otherwise returns
     FALSE.
  *)

CONST
FilePosSize  (const)
   FilePosSize = SIZE(LONGINT) ;
   (* <implementation-defined whole number greater than zero>; *)

TYPE
FilePos (type)
   FilePos = LONGINT ;  (* ARRAY [1 .. FilePosSize] OF SYSTEM.LOC; *)

StartPos
PROCEDURE StartPos (cid: ChanId): FilePos;
  (* If the channel identified by cid is not open to a random access file,
     the exception wrongDevice is raised; otherwise returns the position of
     the start of the file.
  *)

CurrentPos
PROCEDURE CurrentPos (cid: ChanId): FilePos;
  (* If the channel identified by cid is not open to a random access file,
     the exception wrongDevice is raised; otherwise returns the position
     of the current read/write position.
  *)

EndPos
PROCEDURE EndPos (cid: ChanId): FilePos;
  (* If the channel identified by cid is not open to a random access file,
     the exception wrongDevice is raised; otherwise returns the first
     position after which there have been no writes.
  *)

NewPos
```

(continues on next page)

(continued from previous page)

```
PROCEDURE NewPos (cid: ChanId; chunks: INTEGER; chunkSize: CARDINAL;
                  from: FilePos): FilePos;
  (* If the channel identified by cid is not open to a random access file,
     the exception wrongDevice is raised; otherwise returns the position
     (chunks * chunkSize) relative to the position given by from, or
     raises the exception posRange if the required position cannot be
     represented as a value of type FilePos.
  *)

SetPos
PROCEDURE SetPos (cid: ChanId; pos: FilePos);
  (* If the channel identified by cid is not open to a random access file,
     the exception wrongDevice is raised; otherwise sets the read/write
     position to the value given by pos.
  *)

Close
PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a random access file,
     the exception wrongDevice is raised; otherwise closes the channel,
     and assigns the value identifying the invalid channel to cid.
  *)

END RndFile.
```

18.3.45 gm2-libs-iso/SIOResult

```
DEFINITION MODULE SIOResult;

  (* Read results for the default input channel *)

IMPORT IOConsts;

TYPE
ReadResults (type)
  ReadResults = IOConsts.ReadResults;

  (*
ReadResults (type)
    ReadResults =   (* This type is used to classify the result of an input operation *)
    (
      notKnown,     (* no read result is set *)
      allRight,     (* data is as expected or as required *)
      outOfRange,   (* data cannot be represented *)
      wrongFormat,  (* data not in expected format *)
      endOfLine,    (* end of line seen before expected data *)
      endOfInput    (* end of input seen before expected data *)
    );
  *)
```

(continues on next page)

(continued from previous page)

```
ReadResult
PROCEDURE ReadResult (): ReadResults;
  (* Returns the result for the last read operation on the default input channel. *)

END SIOResult.
```

18.3.46 gm2-libs-iso/SLongIO

```
DEFINITION MODULE SLongIO;

  (* Input and output of long real numbers in decimal text form
     using default channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E", ["+" | "-"], decimal digit, {decimal digit}
  *)

ReadReal
PROCEDURE ReadReal (VAR real: LONGREAL);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of a signed
     fixed or floating point number. The value of this number
     is assigned to real.  The read result is set to the value
     allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteFloat
PROCEDURE WriteFloat (real: LONGREAL; sigFigs: CARDINAL;
                      width: CARDINAL);
  (* Writes the value of real to the default output channel in
     floating-point text form, with sigFigs significant figures,
     in a field of the given minimum width.
  *)

WriteEng
PROCEDURE WriteEng (real: LONGREAL; sigFigs: CARDINAL;
                    width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with
     one to three digits in the whole number part, and with an
     exponent that is a multiple of three.
  *)

WriteFixed
```

(continues on next page)

(continued from previous page)

```
PROCEDURE WriteFixed (real: LONGREAL; place: INTEGER;
                      width: CARDINAL);
  (* Writes the value of real to the default output channel in
     fixed-point text form, rounded to the given place relative
     to the decimal point, in a field of the given minimum width.
  *)

WriteReal
PROCEDURE WriteReal (real: LONGREAL; width: CARDINAL);
  (* Writes the value of real to the default output channel, as
     WriteFixed if the sign and magnitude can be shown in the
     given width, or otherwise as WriteFloat. The number of
     places or significant digits depends on the given width.
  *)

END SLongIO.
```

18.3.47 gm2-libs-iso/SLongWholeIO

```
DEFINITION MODULE SLongWholeIO;

  (* Input and output of whole numbers in decimal text form over
     default channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

ReadInt
PROCEDURE ReadInt (VAR int: LONGINT);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of a signed
     whole number.  The value of this number is assigned
     to int.  The read result is set to the value allRight,
     outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteInt
PROCEDURE WriteInt (int: LONGINT; width: CARDINAL);
  (* Writes the value of int to the default output channel in
     text form, in a field of the given minimum width.
  *)

ReadCard
PROCEDURE ReadCard (VAR card: LONGCARD);
```

(continues on next page)

(continued from previous page)

```
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of an
     unsigned whole number.  The value of this number is
     assigned to card.  The read result is set to the value
     allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteCard
PROCEDURE WriteCard (card: LONGCARD; width: CARDINAL);
  (* Writes the value of card to the default output channel in
     text form, in a field of the given minimum width.
  *)

END SLongWholeIO.
```

18.3.48 gm2-libs-iso/SRawIO

```
DEFINITION MODULE SRawIO;

  (* Reading and writing data over default channels using raw operations, that is, with no
     conversion or interpretation. The read result is of the type IOConsts.ReadResults.
  *)

IMPORT SYSTEM;

Read
PROCEDURE Read (VAR to: ARRAY OF SYSTEM.LOC);
  (* Reads storage units from the default input channel, and assigns them to successive
     components of to.  The read result is set to the value allRight, wrongFormat, or
     endOfInput.
  *)

Write
PROCEDURE Write (from: ARRAY OF SYSTEM.LOC);
  (* Writes storage units to the default output channel from successive components of from.
  *)

END SRawIO.
```

18.3.49 gm2-libs-iso/SRealIO

```
DEFINITION MODULE SRealIO;

  (* Input and output of real numbers in decimal text form over
     default channels.  The read result is of the type
     IOConsts.ReadResults.
  *)
```

(continues on next page)

(continued from previous page)

```
  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E", ["+" | "-"], decimal digit, {decimal digit}
  *)

ReadReal
PROCEDURE ReadReal (VAR real: REAL);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of a signed
     fixed or floating point number. The value of this number
     is assigned to real.  The read result is set to the value
     allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteFloat
PROCEDURE WriteFloat (real: REAL; sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to the default output channel in
     floating-point text form, with sigFigs significant figures,
     in a field of the given minimum width.
  *)

WriteEng
PROCEDURE WriteEng (real: REAL; sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with one to
     three digits in the whole number part, and with an exponent that
     is a multiple of three.
  *)

WriteFixed
PROCEDURE WriteFixed (real: REAL; place: INTEGER; width: CARDINAL);
  (* Writes the value of real to the default output channel in
     fixed-point text form, rounded to the given place relative
     to the decimal point, in a field of the given minimum width.
  *)

WriteReal
PROCEDURE WriteReal (real: REAL; width: CARDINAL);
  (* Writes the value of real to the default output channel, as
     WriteFixed if the sign and magnitude can be shown in the
     given width, or otherwise as WriteFloat. The number of
     places or significant digits depends on the given width.
  *)

END SRealIO.
```

18.3.50 gm2-libs-iso/SShortIO

```
DEFINITION MODULE SShortIO;

  (* Input and output of short real numbers in decimal text form
     using default channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit},
       [".", {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E", ["+" | "-"], decimal digit, {decimal digit}
  *)

ReadReal
PROCEDURE ReadReal (VAR real: SHORTREAL);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of a signed
     fixed or floating point number. The value of this number
     is assigned to real.  The read result is set to the value
     allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteFloat
PROCEDURE WriteFloat (real: SHORTREAL; sigFigs: CARDINAL;
                      width: CARDINAL);
  (* Writes the value of real to the default output channel in
     floating-point text form, with sigFigs significant figures,
     in a field of the given minimum width.
  *)

WriteEng
PROCEDURE WriteEng (real: SHORTREAL; sigFigs: CARDINAL;
                    width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with
     one to three digits in the whole number part, and with an
     exponent that is a multiple of three.
  *)

WriteFixed
PROCEDURE WriteFixed (real: SHORTREAL; place: INTEGER;
                      width: CARDINAL);
  (* Writes the value of real to the default output channel in
     fixed-point text form, rounded to the given place relative
     to the decimal point, in a field of the given minimum width.
  *)

WriteReal
```

(continues on next page)

(continued from previous page)

```
PROCEDURE WriteReal (real: SHORTREAL; width: CARDINAL);
  (* Writes the value of real to the default output channel, as
     WriteFixed if the sign and magnitude can be shown in the
     given width, or otherwise as WriteFloat. The number of
     places or significant digits depends on the given width.
  *)

END SShortIO.
```

18.3.51 gm2-libs-iso/SShortWholeIO

```
DEFINITION MODULE SShortWholeIO;

  (* Input and output of whole numbers in decimal text form over
     default channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

ReadInt
PROCEDURE ReadInt (VAR int: SHORTINT);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of a signed
     whole number.  The value of this number is assigned
     to int.  The read result is set to the value allRight,
     outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteInt
PROCEDURE WriteInt (int: SHORTINT; width: CARDINAL);
  (* Writes the value of int to the default output channel in
     text form, in a field of the given minimum width.
  *)

ReadCard
PROCEDURE ReadCard (VAR card: SHORTCARD);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of an
     unsigned whole number.  The value of this number is
     assigned to card.  The read result is set to the value
     allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteCard
```

(continues on next page)

(continued from previous page)

```
PROCEDURE WriteCard (card: SHORTCARD; width: CARDINAL);
  (* Writes the value of card to the default output channel in
     text form, in a field of the given minimum width.
  *)

END SShortWholeIO.
```

18.3.52 gm2-libs-iso/STextIO

```
DEFINITION MODULE STextIO;

  (* Input and output of character and string types over default channels. The read result
     is of the type IOConsts.ReadResults.
  *)

  (* The following procedures do not read past line marks *)

ReadChar
PROCEDURE ReadChar (VAR ch: CHAR);
  (* If possible, removes a character from the default input stream, and assigns the
     corresponding value to ch.  The read result is set to allRight, endOfLine or
     endOfInput.
  *)

ReadRestLine
PROCEDURE ReadRestLine (VAR s: ARRAY OF CHAR);
  (* Removes any remaining characters from the default input stream before the next line
     mark, copying to s as many as can be accommodated as a string value.  The read result
     is set to the value allRight, outOfRange, endOfLine, or endOfInput.
  *)

ReadString
PROCEDURE ReadString (VAR s: ARRAY OF CHAR);
  (* Removes only those characters from the default input stream before the next line mark
     that can be accommodated in s as a string value, and copies them to s. The read result
     is set to the value allRight, endOfLine, or endOfInput.
  *)

ReadToken
PROCEDURE ReadToken (VAR s: ARRAY OF CHAR);
  (* Skips leading spaces, and then removes characters from the default input stream before
     the next space or line mark, copying to s as many as can be accommodated as a string
     value.  The read result is set to the value allRight, outOfRange, endOfLine, or
     endOfInput.
  *)

  (* The following procedure reads past the next line mark *)

SkipLine
PROCEDURE SkipLine;
```

(continues on next page)

(continued from previous page)

```
  (* Removes successive items from the default input stream up to and including the next
     line mark or until the end of input is reached. The read result is set to the value
     allRight, or endOfInput.
  *)

  (* Output procedures *)

WriteChar
PROCEDURE WriteChar (ch: CHAR);
  (* Writes the value of ch to the default output stream. *)

WriteLn
PROCEDURE WriteLn;
  (* Writes a line mark to the default output stream. *)

WriteString
PROCEDURE WriteString (s: ARRAY OF CHAR);
  (* Writes the string value of s to the default output stream. *)

END STextIO.
```

18.3.53 gm2-libs-iso/SWholeIO

```
DEFINITION MODULE SWholeIO;

  (* Input and output of whole numbers in decimal text form over
     default channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

ReadInt
PROCEDURE ReadInt (VAR int: INTEGER);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of a signed
     whole number.  The value of this number is assigned
     to int.  The read result is set to the value allRight,
     outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteInt
PROCEDURE WriteInt (int: INTEGER; width: CARDINAL);
  (* Writes the value of int to the default output channel in
     text form, in a field of the given minimum width.
```

(continues on next page)

(continued from previous page)

```
  *)

ReadCard
PROCEDURE ReadCard (VAR card: CARDINAL);
  (* Skips leading spaces, and removes any remaining characters
     from the default input channel that form part of an
     unsigned whole number.  The value of this number is
     assigned to card.  The read result is set to the value
     allRight, outOfRange, wrongFormat, endOfLine, or endOfInput.
  *)

WriteCard
PROCEDURE WriteCard (card: CARDINAL; width: CARDINAL);
  (* Writes the value of card to the default output channel in
     text form, in a field of the given minimum width.
  *)

END SWholeIO.
```

18.3.54 gm2-libs-iso/SYSTEM

```
DEFINITION MODULE SYSTEM;

  (* Gives access to system programming facilities that are probably
     non portable. *)

  (* The constants and types define underlying properties of storage *)

EXPORT QUALIFIED BITSPERLOC, LOCSPERWORD,
                 LOC, ADDRESS, BYTE, WORD, INTEGER8,
                 INTEGER16, INTEGER32, INTEGER64, CARDINAL8,
                 CARDINAL16, CARDINAL32, CARDINAL64, WORD16,
                 WORD32, WORD64, BITSET8, BITSET16,
                 BITSET32, REAL32, REAL64, REAL128,
                 COMPLEX32, COMPLEX64, COMPLEX128, CSIZE_T,
                 CSSIZE_T,
                 ADDADR, SUBADR, DIFADR, MAKEADR, ADR, ROTATE,
                 SHIFT, CAST, TSIZE,

                 (* Internal GM2 compiler functions *)
                 ShiftVal, ShiftLeft, ShiftRight,
                 RotateVal, RotateLeft, RotateRight,
                 THROW, TBITSIZE ;

CONST
                  (* <implementation-defined constant> ; *)
BITSPERLOC     (const)
  BITSPERLOC    = __ATTRIBUTE__ __BUILTIN__ ((BITS_PER_UNIT)) ;
                  (* <implementation-defined constant> ; *)
LOCSPERWORD    (const)
```

(continues on next page)

(continued from previous page)

```
  LOCSPERWORD   = __ATTRIBUTE__ __BUILTIN__ ((UNITS_PER_WORD)) ;
                  (* <implementation-defined constant> ; *)
LOCSPERBYTE  (const)
  LOCSPERBYTE = 8 DIV BITSPERLOC ;

(*
   all the objects below are declared internally to gm2
   ====================================================

TYPE
LOC (type)
   LOC ;
ADDRESS (type)
   ADDRESS ;
BYTE (type)
   BYTE ;
WORD (type)
   WORD ;
INTEGER8 (type)
   INTEGER8 ;
INTEGER16 (type)
   INTEGER16 ;
INTEGER32 (type)
   INTEGER32 ;
INTEGER64 (type)
   INTEGER64 ;
CARDINAL8 (type)
   CARDINAL8 ;
CARDINAL16 (type)
   CARDINAL16 ;
CARDINAL32 (type)
   CARDINAL32 ;
CARDINAL64 (type)
   CARDINAL64 ;
WORD16 (type)
   WORD16 ;
WORD32 (type)
   WORD32 ;
WORD64 (type)
   WORD64 ;
BITSET8 (type)
   BITSET8 ;
BITSET16 (type)
   BITSET16 ;
BITSET32 (type)
   BITSET32 ;
REAL32 (type)
   REAL32 ;
REAL64 (type)
   REAL64 ;
REAL128 (type)
```

(continues on next page)

(continued from previous page)

```
   REAL128 ;
COMPLEX32 (type)
   COMPLEX32 ;
COMPLEX64 (type)
   COMPLEX64 ;
COMPLEX128 (type)
   COMPLEX128 ;
CSIZE_T (type)
   CSIZE_T ;
CSSIZE_T (type)
   CSSIZE_T ;

TYPE
  LOC; (* A system basic type. Values are the uninterpreted
          contents of the smallest addressable unit of storage *)
ADDRESS (type)
  ADDRESS = POINTER TO LOC;
WORD (type)
  WORD = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

  (* BYTE and LOCSPERBYTE are provided if appropriate for machine *)

TYPE
BYTE (type)
  BYTE = ARRAY [0 .. LOCSPERBYTE-1] OF LOC;

ADDADR
PROCEDURE ADDADR (addr: ADDRESS; offset: CARDINAL): ADDRESS;
  (* Returns address given by (addr + offset), or may raise
     an exception if this address is not valid.
  *)

SUBADR
PROCEDURE SUBADR (addr: ADDRESS; offset: CARDINAL): ADDRESS;
  (* Returns address given by (addr - offset), or may raise an
     exception if this address is not valid.
  *)

DIFADR
PROCEDURE DIFADR (addr1, addr2: ADDRESS): INTEGER;
  (* Returns the difference between addresses (addr1 - addr2),
     or may raise an exception if the arguments are invalid
     or address space is non-contiguous.
  *)

MAKEADR
PROCEDURE MAKEADR (high: <some type>; ...): ADDRESS;
  (* Returns an address constructed from a list of values whose
     types are implementation-defined, or may raise an
     exception if this address is not valid.
```

(continues on next page)

(continued from previous page)

```
    In GNU Modula-2, MAKEADR can take any number of arguments
    which are mapped onto the type ADDRESS. The first parameter
    maps onto the high address bits and subsequent parameters map
    onto lower address bits. For example:

    a := MAKEADR(BYTE(0FEH), BYTE(0DCH), BYTE(0BAH), BYTE(098H),
                 BYTE(076H), BYTE(054H), BYTE(032H), BYTE(010H)) ;

    then the value of, a, on a 64 bit machine is: 0FEDCBA9876543210H

    The parameters do not have to be the same type, but constants
    _must_ be typed.
  *)

ADR
PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
  (* Returns the address of variable v. *)

ROTATE
PROCEDURE ROTATE (val: <a packedset type>;
                  num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by rotating up/right
     or down/right by the absolute value of num.  The direction is
     down/right if the sign of num is negative, otherwise the direction
     is up/left.
  *)

SHIFT
PROCEDURE SHIFT (val: <a packedset type>;
                 num: INTEGER): <type of first parameter>;
  (* Returns a bit sequence obtained from val by shifting up/left
     or down/right by the absolute value of num, introducing
     zeros as necessary.  The direction is down/right if the sign of
     num is negative, otherwise the direction is up/left.
  *)

CAST
PROCEDURE CAST (<targettype>; val: <anytype>): <targettype>;
  (* CAST is a type transfer function.  Given the expression
     denoted by val, it returns a value of the type <targettype>.
     An invalid value for the target value or a
     physical address alignment problem may raise an exception.
  *)

TSIZE
PROCEDURE TSIZE (<type>; ... ): CARDINAL;
  (* Returns the number of LOCS used to store a value of the
     specified <type>.   The extra parameters, if present,
     are used to distinguish variants in a variant record.
  *)
```

(continues on next page)

(continued from previous page)

```
THROW
PROCEDURE THROW (i: INTEGER) ;
  (*
     THROW is a GNU extension and was not part of the PIM or ISO
     standards.  It throws an exception which will be caught by the
     EXCEPT block (assuming it exists).  This is a compiler builtin
     function which interfaces to the GCC exception handling runtime
     system.
     GCC uses the term throw, hence the naming distinction between
     the GCC builtin and the Modula-2 runtime library procedure Raise.
     The later library procedure Raise will call SYSTEM.THROW after
     performing various housekeeping activities.
  *)

TBITSIZE
PROCEDURE TBITSIZE (<type>) : CARDINAL ;
  (* Returns the minimum number of bits necessary to represent
     <type>.  This procedure function is only useful for determining
     the number of bits used for any type field within a packed RECORD.
     It is not particularly useful elsewhere since <type> might be
     optimized for speed, for example a BOOLEAN could occupy a WORD.
  *)
*)

(* The following procedures are invoked by GNU Modula-2 to
   shift non word set types. They are not part of ISO Modula-2
   but are used by GNU Modula-2 to implement the SHIFT procedure
   defined above. *)

(*
   ShiftVal - is a runtime procedure whose job is to implement
              the SHIFT procedure of ISO SYSTEM. GNU Modula-2 will
              inline a SHIFT of a single WORD sized set and will only
              call this routine for larger sets.
*)

ShiftVal
PROCEDURE ShiftVal (VAR s, d: ARRAY OF BITSET;
                    SetSizeInBits: CARDINAL;
                    ShiftCount: INTEGER) ;

(*
   ShiftLeft - performs the shift left for a multi word set.
               This procedure might be called by the back end of
               GNU Modula-2 depending whether amount is known at
               compile time.
*)

ShiftLeft
PROCEDURE ShiftLeft (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
```

(continues on next page)

(continued from previous page)

```
                    ShiftCount: CARDINAL) ;

(*
   ShiftRight - performs the shift left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

ShiftRight
PROCEDURE ShiftRight (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      ShiftCount: CARDINAL) ;

(*
   RotateVal - is a runtime procedure whose job is to implement
               the ROTATE procedure of ISO SYSTEM. GNU Modula-2 will
               inline a ROTATE of a single WORD (or less)
               sized set and will only call this routine for larger
               sets.
*)

RotateVal
PROCEDURE RotateVal (VAR s, d: ARRAY OF BITSET;
                     SetSizeInBits: CARDINAL;
                     RotateCount: INTEGER) ;

(*
   RotateLeft - performs the rotate left for a multi word set.
                This procedure might be called by the back end of
                GNU Modula-2 depending whether amount is known at
                compile time.
*)

RotateLeft
PROCEDURE RotateLeft (VAR s, d: ARRAY OF BITSET;
                      SetSizeInBits: CARDINAL;
                      RotateCount: CARDINAL) ;

(*
   RotateRight - performs the rotate right for a multi word set.
                 This procedure might be called by the back end of
                 GNU Modula-2 depending whether amount is known at
                 compile time.
*)

RotateRight
PROCEDURE RotateRight (VAR s, d: ARRAY OF BITSET;
                       SetSizeInBits: CARDINAL;
                       RotateCount: CARDINAL) ;

END SYSTEM.
```

18.3.55 gm2-libs-iso/Semaphores

```
DEFINITION MODULE Semaphores;

  (* Provides mutual exclusion facilities for use by processes. *)

TYPE
  SEMAPHORE;

Create
PROCEDURE Create (VAR s: SEMAPHORE; initialCount: CARDINAL );
  (* Creates and returns s as the identity of a new semaphore that
     has its associated count initialized to initialCount, and has
     no processes yet waiting on it.
  *)

Destroy
PROCEDURE Destroy (VAR s: SEMAPHORE);
  (* Recovers the resources used to implement the semaphore s,
     provided that no process is waiting for s to become free.
  *)

Claim
PROCEDURE Claim (s: SEMAPHORE);
  (* If the count associated with the semaphore s is non-zero,
     decrements this count and allows the calling process to
     continue; otherwise suspends the calling process until
     s is released.
  *)

Release
PROCEDURE Release (s: SEMAPHORE);
  (* If there are any processes waiting on the semaphore s,
     allows one of them to enter the ready state; otherwise
     increments the count associated with s.
  *)

CondClaim
PROCEDURE CondClaim (s: SEMAPHORE): BOOLEAN;
  (* Returns FALSE if the call Claim(s) would cause the calling
     process to be suspended; in this case the count associated
     with s is not changed. Otherwise returns TRUE and the
     associated count is decremented.
  *)

END Semaphores.
```

18.3.56 gm2-libs-iso/SeqFile

```
DEFINITION MODULE SeqFile;

  (* Rewindable sequential files *)

IMPORT IOChan, ChanConsts;

TYPE
ChanId (type)
  ChanId = IOChan.ChanId;
FlagSet (type)
  FlagSet = ChanConsts.FlagSet;
OpenResults (type)
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  (* input operations are requested/available *)
read  (const)
  read = FlagSet{ChanConsts.readFlag};

  (* output operations are requested/available *)
write  (const)
  write = FlagSet{ChanConsts.writeFlag};

  (* a file may/must/did exist before the channel is opened *)
old  (const)
  old = FlagSet{ChanConsts.oldFlag};

  (* text operations are requested/available *)
text  (const)
  text = FlagSet{ChanConsts.textFlag};

  (* raw operations are requested/available *)
raw  (const)
  raw = FlagSet{ChanConsts.rawFlag};

OpenWrite
PROCEDURE OpenWrite (VAR cid: ChanId; name: ARRAY OF CHAR;
                     flags: FlagSet; VAR res: OpenResults);
  (*
     Attempts to obtain and open a channel connected to a stored
     rewindable file of the given name.
     The write flag is implied; without the raw flag, text is
     implied.  If successful, assigns to cid the identity of
     the opened channel, assigns the value opened to res, and
     selects output mode, with the write position at the start
     of the file (i.e. the file is of zero length).
     If a channel cannot be opened as required, the value of
     res indicates the reason, and cid identifies the invalid
```

(continues on next page)

(continued from previous page)

```
     channel.
  *)

OpenAppend
PROCEDURE OpenAppend (VAR cid: ChanId; name: ARRAY OF CHAR;
                      flags: FlagSet; VAR res: OpenResults);
  (*
     Attempts to obtain and open a channel connected to a stored
     rewindable file of the given name.  The write and old flags
     are implied; without the raw flag, text is implied.  If
     successful, assigns to cid the identity of the opened channel,
     assigns the value opened to res, and selects output mode,
     with the write position corresponding to the length of the
     file.  If a channel cannot be opened as required, the value
     of res indicates the reason, and cid identifies the invalid
     channel.
  *)

OpenRead
PROCEDURE OpenRead (VAR cid: ChanId; name: ARRAY OF CHAR;
                    flags: FlagSet; VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a stored
     rewindable file of the given name.
     The read and old flags are implied; without the raw flag,
     text is implied.  If successful, assigns to cid the
     identity of the opened channel, assigns the value opened to
     res, and selects input mode, with the read position
     corresponding to the start of the file.
     If a channel cannot be opened as required, the value of
     res indicates the reason, and cid identifies the invalid
     channel.
  *)

IsSeqFile
PROCEDURE IsSeqFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a
     rewindable sequential file. *)

Reread
PROCEDURE Reread (cid: ChanId);
  (* If the channel identified by cid is not open to a rewindable
     sequential file, the exception wrongDevice is raised;
     otherwise attempts to set the read position to the
     start of the file, and to select input mode.
     If the operation cannot be performed (perhaps because of
     insufficient permissions) neither input mode nor output
     mode is selected.
  *)

Rewrite
PROCEDURE Rewrite (cid: ChanId);
```

(continues on next page)

(continued from previous page)

```
  (* If the channel identified by cid is not open to a
     rewindable sequential file, the exception wrongDevice is
     raised; otherwise, attempts to truncate the file to zero
     length, and to select output mode.  If the operation
     cannot be performed (perhaps because of insufficient
     permissions) neither input mode nor output mode is selected.
  *)

Close
PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a rewindable
     sequential file, the exception wrongDevice is raised;
     otherwise closes the channel, and assigns the value
     identifying the invalid channel to cid.
  *)

END SeqFile.
```

18.3.57 gm2-libs-iso/ServerSocket

```
DEFINITION MODULE ServerSocket ;

(*
    Description: provides a mechanism to open a server socket
                 as an ISO Modula-2 channel.
*)

FROM IOChan IMPORT ChanId ;
FROM ChanConsts IMPORT FlagSet, OpenResults ;

(*
   OpenSocketBindListen - opens a TCP server socket.  The socket
                          is bound to, port, and will allow, listen,
                          pending connections.  The result of these
                          combined operations is returned in, res.
*)

OpenSocketBindListen
PROCEDURE OpenSocketBindListen (VAR socketid: ChanId;
                                port: CARDINAL; listen: CARDINAL;
                                VAR res: OpenResults) ;

(*
   OpenAccept - attempts to open a new channel whose
                input/output capability is determined by,
                flags.  The result of this attempt is returned
                in res.
*)

OpenAccept
```

(continues on next page)

(continued from previous page)

```
PROCEDURE OpenAccept (VAR cid: ChanId; socketid: ChanId;
                      flags: FlagSet; VAR res: OpenResults) ;

(*
   Close - if the channel identified by cid was not opened as
           a server socket stream, the exception wrongDevice is
           raised; otherwise closes the channel, and assigns
           the value identifying the invalid channel to cid.
*)

Close
PROCEDURE Close (VAR cid: ChanId) ;

(*
   IsSocket - tests if the channel identified by cid is open as
              a server socket stream.
*)

IsSocket
PROCEDURE IsSocket (cid: ChanId) : BOOLEAN ;

END ServerSocket.
```

18.3.58 gm2-libs-iso/ShortComplexMath

```
DEFINITION MODULE ShortComplexMath;

  (* Mathematical functions for the type SHORTCOMPLEX *)

CONST
i  (const)
  i =    CMPLX (0.0, 1.0);
one  (const)
  one =  CMPLX (1.0, 0.0);
zero  (const)
  zero = CMPLX (0.0, 0.0);

abs
PROCEDURE abs (z: SHORTCOMPLEX): SHORTREAL;
  (* Returns the length of z *)

arg
PROCEDURE arg (z: SHORTCOMPLEX): SHORTREAL;
  (* Returns the angle that z subtends to the positive real axis *)

conj
PROCEDURE conj (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the complex conjugate of z *)

power
```

(continues on next page)

(continued from previous page)

```
PROCEDURE power (base: SHORTCOMPLEX; exponent: SHORTREAL): SHORTCOMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

sqrt
PROCEDURE sqrt (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the principal square root of z *)

exp
PROCEDURE exp (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the complex exponential of z *)

ln
PROCEDURE ln (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

sin
PROCEDURE sin (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the sine of z *)

cos
PROCEDURE cos (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the cosine of z *)

tan
PROCEDURE tan (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the tangent of z *)

arcsin
PROCEDURE arcsin (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the arcsine of z *)

arccos
PROCEDURE arccos (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the arccosine of z *)

arctan
PROCEDURE arctan (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the arctangent of z *)

polarToComplex
PROCEDURE polarToComplex (abs, arg: SHORTREAL): SHORTCOMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

scalarMult
PROCEDURE scalarMult (scalar: SHORTREAL; z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the scalar product of scalar with z *)

IsCMathException
PROCEDURE IsCMathException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution state
     because of the raising of an exception in a routine from this module; otherwise
```

(continues on next page)

(continued from previous page)

```
     returns FALSE.
  *)

END ShortComplexMath.
```

18.3.59 gm2-libs-iso/ShortIO

```
DEFINITION MODULE ShortIO;

  (* Input and output of short real numbers in decimal text form
     over specified channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed fixed-point real number is
       ["+" | "-"], decimal digit, {decimal digit}, [".",
       {decimal digit}]

     The text form of a signed floating-point real number is
       signed fixed-point real number,
       "E", ["+" | "-"], decimal digit, {decimal digit}
  *)

ReadReal
PROCEDURE ReadReal (cid: IOChan.ChanId; VAR real: SHORTREAL);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of a signed fixed or floating
     point number.  The value of this number is assigned to real.
     The read result is set to the value allRight, outOfRange,
     wrongFormat, endOfLine, or endOfInput.
  *)

WriteFloat
PROCEDURE WriteFloat (cid: IOChan.ChanId; real: SHORTREAL;
                      sigFigs: CARDINAL; width: CARDINAL);
  (* Writes the value of real to cid in floating-point text form,
     with sigFigs significant figures, in a field of the given
     minimum width.
  *)

WriteEng
PROCEDURE WriteEng (cid: IOChan.ChanId; real: SHORTREAL;
                    sigFigs: CARDINAL; width: CARDINAL);
  (* As for WriteFloat, except that the number is scaled with
     one to three digits in the whole number part, and with an
     exponent that is a multiple of three.
  *)
```

(continues on next page)

(continued from previous page)

```
WriteFixed
PROCEDURE WriteFixed (cid: IOChan.ChanId; real: SHORTREAL;
                      place: INTEGER; width: CARDINAL);
  (* Writes the value of real to cid in fixed-point text form,
     rounded to the given place relative to the decimal point,
     in a field of the given minimum width.
  *)

WriteReal
PROCEDURE WriteReal (cid: IOChan.ChanId; real: SHORTREAL;
                     width: CARDINAL);
  (* Writes the value of real to cid, as WriteFixed if the
     sign and magnitude can be shown in the given width, or
     otherwise as WriteFloat.  The number of places or
     significant digits depends on the given width.
  *)

END ShortIO.
```

18.3.60 gm2-libs-iso/ShortWholeIO

```
DEFINITION MODULE ShortWholeIO;

  (* Input and output of whole numbers in decimal text form
     over specified channels.  The read result is of the
     type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)

ReadInt
PROCEDURE ReadInt (cid: IOChan.ChanId; VAR int: SHORTINT);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of a signed whole number.  The
     value of this number is assigned to int.  The read result
     is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)

WriteInt
PROCEDURE WriteInt (cid: IOChan.ChanId; int: SHORTINT;
                    width: CARDINAL);
  (* Writes the value of int to cid in text form, in a field of
```

(continues on next page)

(continued from previous page)

```
     the given minimum width. *)

ReadCard
PROCEDURE ReadCard (cid: IOChan.ChanId; VAR card: SHORTCARD);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of an unsigned whole number.  The
     value of this number is assigned to card. The read result
     is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)

WriteCard
PROCEDURE WriteCard (cid: IOChan.ChanId; card: SHORTCARD;
                     width: CARDINAL);
  (* Writes the value of card to cid in text form, in a field
     of the given minimum width. *)

END ShortWholeIO.
```

18.3.61 gm2-libs-iso/SimpleCipher

```
DEFINITION MODULE SimpleCipher ;

(*
    Description: provides a simple Caesar cipher layer which
                 can be attached to any channel device.  This,
                 pedagogical, module is designed to show how
                 it is possible to add further layers underneath
                 the channel devices.
*)

FROM IOChan IMPORT ChanId ;

(*
   InsertCipherLayer - inserts a caesar cipher below channel, cid.
                       The encryption, key, is specified.
*)

InsertCipherLayer
PROCEDURE InsertCipherLayer (cid: ChanId; key: INTEGER) ;

(*
   RemoveCipherLayer - removes a Caesar cipher below channel, cid.
*)

RemoveCipherLayer
PROCEDURE RemoveCipherLayer (cid: ChanId) ;

END SimpleCipher.
```

18.3.62 gm2-libs-iso/StdChans

```
DEFINITION MODULE StdChans;

  (* Access to standard and default channels *)

IMPORT IOChan;

TYPE
ChanId (type)
  ChanId = IOChan.ChanId;
    (* Values of this type are used to identify channels *)

  (* The following functions return the standard channel values.
     These channels cannot be closed.
  *)

StdInChan
PROCEDURE StdInChan (): ChanId;
  (* Returns the identity of the implementation-defined standard source for
program
     input.
  *)

StdOutChan
PROCEDURE StdOutChan (): ChanId;
  (* Returns the identity of the implementation-defined standard source for program
     output.
  *)

StdErrChan
PROCEDURE StdErrChan (): ChanId;
  (* Returns the identity of the implementation-defined standard destination for program
     error messages.
  *)

NullChan
PROCEDURE NullChan (): ChanId;
  (* Returns the identity of a channel open to the null device. *)

  (* The following functions return the default channel values *)

InChan
PROCEDURE InChan (): ChanId;
  (* Returns the identity of the current default input channel. *)

OutChan
PROCEDURE OutChan (): ChanId;
  (* Returns the identity of the current default output channel. *)

ErrChan
PROCEDURE ErrChan (): ChanId;
```

(continues on next page)

(continued from previous page)

```
  (* Returns the identity of the current default error message channel. *)

  (* The following procedures allow for redirection of the default channels *)

SetInChan
PROCEDURE SetInChan (cid: ChanId);
  (* Sets the current default input channel to that identified by cid. *)

SetOutChan
PROCEDURE SetOutChan (cid: ChanId);
  (* Sets the current default output channel to that identified by cid. *)

SetErrChan
PROCEDURE SetErrChan (cid: ChanId);
  (* Sets the current default error channel to that identified by cid. *)

END StdChans.
```

18.3.63 gm2-libs-iso/Storage

```
DEFINITION MODULE Storage;

  (* Facilities for dynamically allocating and deallocating storage *)

IMPORT SYSTEM;

ALLOCATE
PROCEDURE ALLOCATE (VAR addr: SYSTEM.ADDRESS; amount: CARDINAL);
  (* Allocates storage for a variable of size amount and assigns
     the address of this variable to addr. If there is insufficient
     unallocated storage to do this, the value NIL is assigned to addr.
  *)

DEALLOCATE
PROCEDURE DEALLOCATE (VAR addr: SYSTEM.ADDRESS; amount: CARDINAL);
  (* Deallocates amount locations allocated by ALLOCATE for
     the storage of the variable addressed by addr and assigns
     the value NIL to addr.
  *)

REALLOCATE
PROCEDURE REALLOCATE (VAR addr: SYSTEM.ADDRESS; amount: CARDINAL);
  (* Attempts to reallocate, amount of storage.  Effectively it
     calls ALLOCATE, copies the amount of data pointed to by
     addr into the new space and DEALLOCATES the addr.
     This procedure is a GNU extension.
  *)

TYPE
StorageExceptions (type)
```

(continues on next page)

(continued from previous page)

```
  StorageExceptions = (
    nilDeallocation,             (* first argument to DEALLOCATE is NIL *)
    pointerToUnallocatedStorage, (* storage to deallocate not allocated by ALLOCATE *)
    wrongStorageToUnallocate     (* amount to deallocate is not amount allocated *)
  );

IsStorageException
PROCEDURE IsStorageException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional
     execution state because of the raising of an exception from
     StorageExceptions; otherwise returns FALSE.
  *)

StorageException
PROCEDURE StorageException (): StorageExceptions;
  (* If the current coroutine is in the exceptional execution
     state because of the raising of an exception from
     StorageExceptions, returns the corresponding
     enumeration value, and otherwise raises an exception.
  *)

END Storage.
```

18.3.64 gm2-libs-iso/StreamFile

```
DEFINITION MODULE StreamFile;

  (* Independent sequential data streams *)

IMPORT IOChan, ChanConsts;

TYPE
ChanId (type)
  ChanId = IOChan.ChanId;
FlagSet (type)
  FlagSet = ChanConsts.FlagSet;
OpenResults (type)
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
read  (const)
  read = FlagSet{ChanConsts.readFlag};   (* input operations are requested/available *)
write  (const)
  write = FlagSet{ChanConsts.writeFlag}; (* output operations are requested/available *)
old  (const)
  old = FlagSet{ChanConsts.oldFlag};     (* a file may/must/did exist before the channel is
                                            opened *)
text  (const)
```

(continues on next page)

(continued from previous page)

```
  text = FlagSet{ChanConsts.textFlag};   (* text operations are requested/available *)
raw  (const)
  raw = FlagSet{ChanConsts.rawFlag};     (* raw operations are requested/available *)

Open
PROCEDURE Open (VAR cid: ChanId; name: ARRAY OF CHAR;
                flags: FlagSet; VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to a
     sequential stream of the given name.
     The read flag implies old; without the raw flag, text is
     implied.  If successful, assigns to cid the identity of
     the opened channel, and assigns the value opened to res.
     If a channel cannot be opened as required, the value of
     res indicates the reason, and cid identifies the invalid
     channel.
  *)

IsStreamFile
PROCEDURE IsStreamFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a sequential stream. *)

Close
PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a sequential stream, the exception
     wrongDevice is raised; otherwise closes the channel, and assigns the value identifying
     the invalid channel to cid.
  *)

END StreamFile.
```

18.3.65 gm2-libs-iso/StringChan

```
DEFINITION MODULE StringChan ;

(*
    Description: provides a set of Channel and String
                 input and output procedures.
*)

FROM DynamicStrings IMPORT String ;
IMPORT IOChan;

(*
   writeString - writes a string, s, to ChanId, cid.
                 The string, s, is not destroyed.
*)

writeString
PROCEDURE writeString (cid: IOChan.ChanId; s: String) ;
```

(continues on next page)

(continued from previous page)

```
(*
   writeFieldWidth - writes a string, s, to ChanId, cid.
                     The string, s, is not destroyed and it
                     is prefixed by spaces so that at least,
                     width, characters are written.  If the
                     string, s, is longer than width then
                     no spaces are prefixed to the output
                     and the entire string is written.
*)

writeFieldWidth
PROCEDURE writeFieldWidth (cid: IOChan.ChanId;
                           s: String; width: CARDINAL) ;

END StringChan.
```

18.3.66 gm2-libs-iso/Strings

```
DEFINITION MODULE Strings;

  (* Facilities for manipulating strings *)

TYPE
String1 (type)
  String1 = ARRAY [0..0] OF CHAR;
    (* String1 is provided for constructing a value of a single-character string type from a
       single character value in order to pass CHAR values to ARRAY OF CHAR parameters.
    *)

Length
PROCEDURE Length (stringVal: ARRAY OF CHAR): CARDINAL;
  (* Returns the length of stringVal (the same value as would be returned by the
     pervasive function LENGTH).
  *)

(* The following seven procedures construct a string value, and attempt to assign it to a
   variable parameter.  They all have the property that if the length of the constructed string
   value exceeds the capacity of the variable parameter, a truncated value is assigned, while
   if the length of the constructed string value is less than the capacity of the variable
   parameter, a string terminator is appended before assignment is performed.
*)

Assign
PROCEDURE Assign (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Copies source to destination *)

Extract
PROCEDURE Extract (source: ARRAY OF CHAR; startIndex, numberToExtract: CARDINAL;
                   VAR destination: ARRAY OF CHAR);
  (* Copies at most numberToExtract characters from source to destination, starting at position
```

(continues on next page)

(continued from previous page)

```
     startIndex in source.
  *)

Delete
PROCEDURE Delete (VAR stringVar: ARRAY OF CHAR; startIndex, numberToDelete:
CARDINAL);
  (* Deletes at most numberToDelete characters from stringVar, starting at position
     startIndex.
  *)

Insert
PROCEDURE Insert (source: ARRAY OF CHAR; startIndex: CARDINAL;
                  VAR destination: ARRAY OF CHAR);
  (* Inserts source into destination at position startIndex *)

Replace
PROCEDURE Replace (source: ARRAY OF CHAR; startIndex: CARDINAL;
                   VAR destination: ARRAY OF CHAR);
  (* Copies source into destination, starting at position startIndex. Copying stops when
     all of source has been copied, or when the last character of the string value in
     destination has been replaced.
  *)

Append
PROCEDURE Append (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Appends source to destination. *)

Concat
PROCEDURE Concat (source1, source2: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Concatenates source2 onto source1 and copies the result into destination. *)

(* The following predicates provide for pre-testing of the operation-completion
   conditions for the procedures above.
*)

CanAssignAll
PROCEDURE CanAssignAll (sourceLength: CARDINAL; VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if a number of characters, indicated by sourceLength, will fit into
     destination; otherwise returns FALSE.
  *)

CanExtractAll
PROCEDURE CanExtractAll (sourceLength, startIndex, numberToExtract: CARDINAL;
                         VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there are numberToExtract characters starting at startIndex and
     within the sourceLength of some string, and if the capacity of destination is
     sufficient to hold numberToExtract characters; otherwise returns FALSE.
  *)

CanDeleteAll
PROCEDURE CanDeleteAll (stringLength, startIndex, numberToDelete: CARDINAL): BOOLEAN;
```

(continues on next page)

(continued from previous page)

```
  (* Returns TRUE if there are numberToDelete characters starting at startIndex and
     within the stringLength of some string; otherwise returns FALSE.
  *)

CanInsertAll
PROCEDURE CanInsertAll (sourceLength, startIndex: CARDINAL;
                        VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is room for the insertion of sourceLength characters from
     some string into destination starting at startIndex; otherwise returns FALSE.
  *)

CanReplaceAll
PROCEDURE CanReplaceAll (sourceLength, startIndex: CARDINAL;
                         VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is room for the replacement of sourceLength characters in
     destination starting at startIndex; otherwise returns FALSE.
  *)

CanAppendAll
PROCEDURE CanAppendAll (sourceLength: CARDINAL; VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is sufficient room in destination to append a string of
     length sourceLength to the string in destination; otherwise returns FALSE.
  *)

CanConcatAll
PROCEDURE CanConcatAll (source1Length, source2Length: CARDINAL;
                        VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is sufficient room in destination for a two strings of
     lengths source1Length and source2Length; otherwise returns FALSE.
  *)

(* The following type and procedures provide for the comparison of string values, and for the
   location of substrings within strings.
*)

TYPE
CompareResults (type)
  CompareResults = (less, equal, greater);

Compare
PROCEDURE Compare (stringVal1, stringVal2: ARRAY OF CHAR): CompareResults;
  (* Returns less, equal, or greater, according as stringVal1 is lexically less than,
     equal to, or greater than stringVal2.
  *)

Equal
PROCEDURE Equal (stringVal1, stringVal2: ARRAY OF CHAR): BOOLEAN;
  (* Returns Strings.Compare(stringVal1, stringVal2) = Strings.equal *)

FindNext
PROCEDURE FindNext (pattern, stringToSearch: ARRAY OF CHAR; startIndex: CARDINAL;
```

(continues on next page)

(continued from previous page)

```
                    VAR patternFound: BOOLEAN; VAR posOfPattern: CARDINAL);
  (* Looks forward for next occurrence of pattern in stringToSearch, starting the search at
     position startIndex. If startIndex < LENGTH(stringToSearch) and pattern is found,
     patternFound is returned as TRUE, and posOfPattern contains the start position in
     stringToSearch of pattern. Otherwise patternFound is returned as FALSE, and posOfPattern
     is unchanged.
  *)

FindPrev
PROCEDURE FindPrev (pattern, stringToSearch: ARRAY OF CHAR; startIndex: CARDINAL;
                    VAR patternFound: BOOLEAN; VAR posOfPattern: CARDINAL);
  (* Looks backward for the previous occurrence of pattern in stringToSearch and returns the
     position of the first character of the pattern if found. The search for the pattern
     begins at startIndex. If pattern is found, patternFound is returned as TRUE, and
     posOfPattern contains the start position in stringToSearch of pattern in the range
     [0..startIndex]. Otherwise patternFound is returned as FALSE, and posOfPattern is unchanged.
  *)

FindDiff
PROCEDURE FindDiff (stringVal1, stringVal2: ARRAY OF CHAR;
                    VAR differenceFound: BOOLEAN; VAR posOfDifference: CARDINAL);
  (* Compares the string values in stringVal1 and stringVal2 for differences. If they
     are equal, differenceFound is returned as FALSE, and TRUE otherwise. If
     differenceFound is TRUE, posOfDifference is set to the position of the first
     difference; otherwise posOfDifference is unchanged.
  *)

Capitalize
PROCEDURE Capitalize (VAR stringVar: ARRAY OF CHAR);
  (* Applies the function CAP to each character of the string value in stringVar. *)

END Strings.
```

18.3.67 gm2-libs-iso/SysClock

```
DEFINITION MODULE SysClock;

(* Facilities for accessing a system clock that records the date
   and time of day *)

CONST
maxSecondParts  (const)
  maxSecondParts = 1000000 ;

TYPE
Month (type)
  Month    = [1 .. 12];
Day (type)
  Day      = [1 .. 31];
Hour (type)
```

(continues on next page)

(continued from previous page)

```
  Hour     = [0 .. 23];
Min (type)
  Min      = [0 .. 59];
Sec (type)
  Sec      = [0 .. 59];
Fraction (type)
  Fraction = [0 .. maxSecondParts];
UTCDiff (type)
  UTCDiff  = [-780 .. 720];
DateTime (type)
  DateTime =
    RECORD
      year:      CARDINAL;
      month:     Month;
      day:       Day;
      hour:      Hour;
      minute:    Min;
      second:    Sec;
      fractions: Fraction;      (* parts of a second *)
      zone:      UTCDiff;       (* Time zone differential
                                   factor which is the number
                                   of minutes to add to local
                                   time to obtain UTC. *)
      summerTimeFlag: BOOLEAN;  (* Interpretation of flag
                                   depends on local usage. *)
    END;

CanGetClock
PROCEDURE CanGetClock(): BOOLEAN;
(* Tests if the clock can be read *)

CanSetClock
PROCEDURE CanSetClock(): BOOLEAN;
(* Tests if the clock can be set *)

IsValidDateTime
PROCEDURE IsValidDateTime(userData: DateTime): BOOLEAN;
(* Tests if the value of userData is a valid *)

GetClock
PROCEDURE GetClock(VAR userData: DateTime);
(* Assigns local date and time of the day to userData *)

SetClock
PROCEDURE SetClock(userData: DateTime);
(* Sets the system time clock to the given local date and
   time *)

END SysClock.
```

18.3.68 gm2-libs-iso/TERMINATION

```
DEFINITION MODULE TERMINATION;

(* Provides facilities for enquiries concerning the occurrence of termination events. *)

IsTerminating
PROCEDURE IsTerminating (): BOOLEAN ;
  (* Returns true if any coroutine has started  program termination and false otherwise. *)

HasHalted
PROCEDURE HasHalted (): BOOLEAN ;
  (* Returns true if a call to HALT has been made and false otherwise. *)

END TERMINATION.
```

18.3.69 gm2-libs-iso/TermFile

```
DEFINITION MODULE TermFile;

  (* Access to the terminal device *)

  (* Channels opened by this module are connected to a single
     terminal device; typed characters are distributed between
     channels according to the sequence of read requests.
  *)

IMPORT IOChan, ChanConsts;

TYPE
ChanId (type)
  ChanId = IOChan.ChanId;
FlagSet (type)
  FlagSet = ChanConsts.FlagSet;
OpenResults (type)
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
read  (const)
  read = FlagSet{ChanConsts.readFlag};
  (* input operations are requested/available *)
write  (const)
  write = FlagSet{ChanConsts.writeFlag};
  (* output operations are requested/available *)
text  (const)
  text = FlagSet{ChanConsts.textFlag};
  (* text operations are requested/available *)
raw  (const)
```

(continues on next page)

(continued from previous page)

```
  raw = FlagSet{ChanConsts.rawFlag};
  (* raw operations are requested/available *)
echo  (const)
  echo = FlagSet{ChanConsts.echoFlag};
  (* echoing by interactive device on reading of
     characters from input stream requested/applies
  *)

Open
PROCEDURE Open (VAR cid: ChanId; flagset: FlagSet; VAR res: OpenResults);
  (* Attempts to obtain and open a channel connected to
     the terminal.  Without the raw flag, text is implied.
     Without the echo flag, line mode is requested,
     otherwise single character mode is requested.
     If successful, assigns to cid the identity of
     the opened channel, and assigns the value opened to res.
     If a channel cannot be opened as required, the value of
     res indicates the reason, and cid identifies the
     invalid channel.
  *)

IsTermFile
PROCEDURE IsTermFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to
     the terminal. *)

Close
PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to the terminal,
     the exception wrongDevice is raised; otherwise closes the
     channel and assigns the value identifying the invalid channel
     to cid.
  *)

END TermFile.
```

18.3.70 gm2-libs-iso/TextIO

```
DEFINITION MODULE TextIO;

  (* Input and output of character and string types over
     specified channels.  The read result is of the type
     IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The following procedures do not read past line marks *)

ReadChar
```

(continues on next page)

(continued from previous page)

```
PROCEDURE ReadChar (cid: IOChan.ChanId; VAR ch: CHAR);
  (* If possible, removes a character from the input stream
     cid and assigns the corresponding value to ch.  The
     read result is set to the value allRight, endOfLine, or
     endOfInput.
  *)

ReadRestLine
PROCEDURE ReadRestLine (cid: IOChan.ChanId; VAR s: ARRAY OF CHAR);
  (* Removes any remaining characters from the input stream
     cid before the next line mark,  copying to s as many as
     can be accommodated as a string value.  The read result is
     set to the value allRight, outOfRange, endOfLine, or
     endOfInput.
  *)

ReadString
PROCEDURE ReadString (cid: IOChan.ChanId; VAR s: ARRAY OF CHAR);
  (* Removes only those characters from the input stream cid
     before the next line mark that can be accommodated in s
     as a string value, and copies them to s.  The read result
     is set to the value allRight, endOfLine, or endOfInput.
  *)

ReadToken
PROCEDURE ReadToken (cid: IOChan.ChanId; VAR s: ARRAY OF CHAR);
  (* Skips leading spaces, and then removes characters from
     the input stream cid before the next space or line mark,
     copying to s as many as can be accommodated as a string
     value.  The read result is set to the value allRight,
     outOfRange, endOfLine, or endOfInput.
  *)

  (* The following procedure reads past the next line mark *)

SkipLine
PROCEDURE SkipLine (cid: IOChan.ChanId);
  (* Removes successive items from the input stream cid up
     to and including the next line mark, or until the end
     of input is reached.  The read result is set to the
     value allRight, or endOfInput.
  *)

  (* Output procedures *)

WriteChar
PROCEDURE WriteChar (cid: IOChan.ChanId; ch: CHAR);
  (* Writes the value of ch to the output stream cid. *)

WriteLn
PROCEDURE WriteLn (cid: IOChan.ChanId);
```

(continues on next page)

(continued from previous page)

```
  (* Writes a line mark to the output stream cid. *)

WriteString
PROCEDURE WriteString (cid: IOChan.ChanId; s: ARRAY OF CHAR);
  (* Writes the string value in s to the output stream cid. *)

END TextIO.
```

18.3.71 gm2-libs-iso/WholeConv

```
DEFINITION MODULE WholeConv;

  (* Low-level whole-number/string conversions *)

IMPORT
  ConvTypes;

TYPE
ConvResults (type)
  ConvResults = ConvTypes.ConvResults;
        (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

ScanInt
PROCEDURE ScanInt (inputCh: CHAR;
                   VAR chClass: ConvTypes.ScanClass;
                   VAR nextState: ConvTypes.ScanState) ;
  (* Represents the start state of a finite state scanner for signed
     whole numbers - assigns class of inputCh to chClass and a
     procedure representing the next state to nextState.
  *)

FormatInt
PROCEDURE FormatInt (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to INTEGER. *)

ValueInt
PROCEDURE ValueInt (str: ARRAY OF CHAR): INTEGER;
  (* Returns the value corresponding to the signed whole number string
     value str if str is well-formed; otherwise raises the WholeConv
     exception.
   *)

LengthInt
PROCEDURE LengthInt (int: INTEGER): CARDINAL;
  (* Returns the number of characters in the string representation of
     int.
   *)

ScanCard
PROCEDURE ScanCard (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
```

(continues on next page)

(continued from previous page)

```
                    VAR nextState: ConvTypes.ScanState);
  (* Represents the start state of a finite state scanner for unsigned
     whole numbers - assigns class of inputCh to chClass and a procedure
     representing the next state to nextState.
   *)

FormatCard
PROCEDURE FormatCard (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to CARDINAL.
   *)

ValueCard
PROCEDURE ValueCard (str: ARRAY OF CHAR): CARDINAL;
  (* Returns the value corresponding to the unsigned whole number string
     value str if str is well-formed; otherwise raises the WholeConv
     exception.
   *)

LengthCard
PROCEDURE LengthCard (card: CARDINAL): CARDINAL;
  (* Returns the number of characters in the string representation of
     card.
   *)

IsWholeConvException
PROCEDURE IsWholeConvException (): BOOLEAN;
  (* Returns TRUE if the current coroutine is in the exceptional execution
     state because of the raising of an exception in a routine from this
     module; otherwise returns FALSE.
  *)

END WholeConv.
```

18.3.72 gm2-libs-iso/WholeIO

```
DEFINITION MODULE WholeIO;

  (* Input and output of whole numbers in decimal text form
     over specified channels.  The read result is of the
     type IOConsts.ReadResults.
  *)

IMPORT IOChan;

  (* The text form of a signed whole number is
       ["+" | "-"], decimal digit, {decimal digit}

     The text form of an unsigned whole number is
       decimal digit, {decimal digit}
  *)
```

(continues on next page)

(continued from previous page)

```
ReadInt
PROCEDURE ReadInt (cid: IOChan.ChanId; VAR int: INTEGER);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of a signed whole number.  The
     value of this number is assigned to int.  The read result
     is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)

WriteInt
PROCEDURE WriteInt (cid: IOChan.ChanId; int: INTEGER;
                    width: CARDINAL);
  (* Writes the value of int to cid in text form, in a field of
     the given minimum width. *)

ReadCard
PROCEDURE ReadCard (cid: IOChan.ChanId; VAR card: CARDINAL);
  (* Skips leading spaces, and removes any remaining characters
     from cid that form part of an unsigned whole number.  The
     value of this number is assigned to card. The read result
     is set to the value allRight, outOfRange, wrongFormat,
     endOfLine, or endOfInput.
  *)

WriteCard
PROCEDURE WriteCard (cid: IOChan.ChanId; card: CARDINAL;
                     width: CARDINAL);
  (* Writes the value of card to cid in text form, in a field
     of the given minimum width. *)

END WholeIO.
```

18.3.73 gm2-libs-iso/WholeStr

```
DEFINITION MODULE WholeStr;

  (* Whole-number/string conversions *)

IMPORT
  ConvTypes;

TYPE
ConvResults (type)
  ConvResults = ConvTypes.ConvResults;
  (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

(* the string form of a signed whole number is
     ["+" | "-"], decimal digit, {decimal digit}
*)
```

(continues on next page)

(continued from previous page)

```
StrToInt
PROCEDURE StrToInt (str: ARRAY OF CHAR; VAR int: INTEGER;
                    VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent
     characters in str are in the format of a signed whole
     number, assigns a corresponding value to int. Assigns
     a value indicating the format of str to res.
  *)

IntToStr
PROCEDURE IntToStr (int: INTEGER; VAR str: ARRAY OF CHAR);
  (* Converts the value of int to string form and copies the
     possibly truncated result to str. *)

(* the string form of an unsigned whole number is
     decimal digit, {decimal digit}
*)

StrToCard
PROCEDURE StrToCard (str: ARRAY OF CHAR;
                     VAR card: CARDINAL;
                     VAR res: ConvResults);
  (* Ignores any leading spaces in str. If the subsequent
     characters in str are in the format of an unsigned
     whole number, assigns a corresponding value to card.
     Assigns a value indicating the format of str to res.
  *)

CardToStr
PROCEDURE CardToStr (card: CARDINAL; VAR str: ARRAY OF CHAR);
  (* Converts the value of card to string form and copies the
     possibly truncated result to str. *)

END WholeStr.
```

18.3.74 gm2-libs-iso/wrapsock

```
DEFINITION MODULE wrapsock ;

(*
    Description: provides a set of wrappers to some client side
                 tcp socket primatives.
*)

FROM SYSTEM IMPORT ADDRESS ;
FROM ChanConsts IMPORT OpenResults ;

TYPE
clientInfo (type)
```

(continues on next page)

(continued from previous page)

```
   clientInfo = ADDRESS ;

(*
   clientOpen - returns an ISO Modula-2 OpenResult.
                It attempts to connect to:  hostname:portNo.
                If successful then the data structure, c,
                will have its fields initialized.
*)

clientOpen
PROCEDURE clientOpen (c: clientInfo;
                      hostname: ADDRESS;
                      length: CARDINAL;
                      portNo: CARDINAL) : OpenResults ;

(*
   clientOpenIP - returns an ISO Modula-2 OpenResult.
                  It attempts to connect to:  ipaddress:portNo.
                  If successful then the data structure, c,
                  will have its fields initialized.
*)

clientOpenIP
PROCEDURE clientOpenIP (c: clientInfo;
                        ip: CARDINAL;
                        portNo: CARDINAL) : OpenResults ;

(*
   getClientPortNo - returns the portNo from structure, c.
*)

getClientPortNo
PROCEDURE getClientPortNo (c: clientInfo) : CARDINAL ;

(*
   getClientHostname - fills in the hostname of the server
                       the to which the client is connecting.
*)

getClientHostname
PROCEDURE getClientHostname (c: clientInfo;
                             hostname: ADDRESS; high: CARDINAL) ;

(*
   getClientSocketFd - returns the sockFd from structure, c.
*)

getClientSocketFd
PROCEDURE getClientSocketFd (c: clientInfo) : INTEGER ;

(*
```

(continues on next page)

(continued from previous page)

```
   getClientIP - returns the sockFd from structure, s.
*)

getClientIP
PROCEDURE getClientIP (c: clientInfo) : CARDINAL ;

(*
   getPushBackChar - returns TRUE if a pushed back character
                     is available.
*)

getPushBackChar
PROCEDURE getPushBackChar (c: clientInfo; VAR ch: CHAR) : BOOLEAN ;

(*
   setPushBackChar - returns TRUE if it is able to push back a
                     character.
*)

setPushBackChar
PROCEDURE setPushBackChar (c: clientInfo; ch: CHAR) : BOOLEAN ;

(*
   getSizeOfClientInfo - returns the sizeof (opaque data type).
*)

getSizeOfClientInfo
PROCEDURE getSizeOfClientInfo () : CARDINAL ;

END wrapsock.
```

18.3.75 gm2-libs-iso/wraptime

```
DEFINITION MODULE wraptime ;

(*
    Description: provides an interface to various time related
                 entities on the underlying host operating system.
                 It provides access to the glibc/libc functions:
                 gettimeofday, settimeofday and localtime_r.
*)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
timeval (type)
   timeval  = ADDRESS ;
timezone (type)
   timezone = ADDRESS ;
tm (type)
```

(continues on next page)

(continued from previous page)

```
   tm       = ADDRESS ;

(*
   InitTimeval - returns a newly created opaque type.
*)

InitTimeval
PROCEDURE InitTimeval () : timeval ;

(*
   KillTimeval - deallocates the memory associated with an
                 opaque type.
*)

KillTimeval
PROCEDURE KillTimeval (tv: timeval) : timeval ;

(*
   InitTimezone - returns a newly created opaque type.
*)

InitTimezone
PROCEDURE InitTimezone () : timezone ;

(*
   KillTimezone - deallocates the memory associated with an
                  opaque type.
*)

KillTimezone
PROCEDURE KillTimezone (tv: timezone) : timezone ;

(*
   InitTM - returns a newly created opaque type.
*)

InitTM
PROCEDURE InitTM () : tm ;

(*
   KillTM - deallocates the memory associated with an
            opaque type.
*)

KillTM
PROCEDURE KillTM (tv: tm) : tm ;

(*
   gettimeofday - calls gettimeofday(2) with the same parameters, tv,
                  and, tz.  It returns 0 on success.
*)
```

(continues on next page)

(continued from previous page)

```
gettimeofday
PROCEDURE gettimeofday (tv: timeval; tz: timezone) : INTEGER ;

(*
   settimeofday - calls settimeofday(2) with the same parameters, tv,
                  and, tz.  It returns 0 on success.
*)

settimeofday
PROCEDURE settimeofday (tv: timeval; tz: timezone) : INTEGER ;

(*
   GetFractions - returns the tv_usec field inside the timeval structure
                  as a CARDINAL.
*)

GetFractions
PROCEDURE GetFractions (tv: timeval) : CARDINAL ;

(*
   localtime_r - returns the tm parameter, m, after it has been assigned with
                 appropriate contents determined by, tv.  Notice that
                 this procedure function expects, timeval, as its first
                 parameter and not a time_t (as expected by the posix
                 equivalent).  This avoids having to expose a time_t
                 system dependant definition.
*)

localtime_r
PROCEDURE localtime_r (tv: timeval; m: tm) : tm ;

(*
   GetYear - returns the year from the structure, m.
*)

GetYear
PROCEDURE GetYear (m: tm) : CARDINAL ;

(*
   GetMonth - returns the month from the structure, m.
*)

GetMonth
PROCEDURE GetMonth (m: tm) : CARDINAL ;

(*
   GetDay - returns the day of the month from the structure, m.
*)

GetDay
```

(continues on next page)

(continued from previous page)

```
PROCEDURE GetDay (m: tm) : CARDINAL ;

(*
   GetHour - returns the hour of the day from the structure, m.
*)

GetHour
PROCEDURE GetHour (m: tm) : CARDINAL ;

(*
   GetMinute - returns the minute within the hour from the structure, m.
*)

GetMinute
PROCEDURE GetMinute (m: tm) : CARDINAL ;

(*
   GetSecond - returns the seconds in the minute from the structure, m.
               The return value will always be in the range 0..59.
               A leap minute of value 60 will be truncated to 59.
*)

GetSecond
PROCEDURE GetSecond (m: tm) : CARDINAL ;

(*
   GetSummerTime - returns a boolean indicating whether summer time is
                   set.
*)

GetSummerTime
PROCEDURE GetSummerTime (tz: timezone) : BOOLEAN ;

(*
   GetDST - returns the number of minutes west of GMT.
*)

GetDST
PROCEDURE GetDST (tz: timezone) : INTEGER ;

(*
   SetTimeval - sets the fields in timeval, tv, with:
                second, minute, hour, day, month, year, fractions.
*)

SetTimeval
PROCEDURE SetTimeval (tv: timeval;
                      second, minute, hour, day,
                      month, year, yday, wday, isdst: CARDINAL) ;

(*
```

(continues on next page)

(continued from previous page)

```
   SetTimezone - set the timezone field inside timeval, tv.
*)

SetTimezone
PROCEDURE SetTimezone (tv: timeval;
                       zone: CARDINAL; minuteswest: INTEGER) ;

END wraptime.
```

18.4 PIM and Logitech 3.0 Compatible

These modules are provided to enable legacy Modula-2 applications to build with GNU Modula-2. It is advised that these module should not be used for new projects, maybe the ISO libraries or the native compiler PIM libraries (FIO) should be used instead.

Here is an outline of the module layering:

```
InOut  RealInOut  LongIO CardinalIO
    \     |       |      /
          Terminal
----------------------------------
             |
          Termbase
          /      \
    Keyboard      Display
```

Above the line are user level PIM [234] and Logitech 3.0 compatible modules. Below the line Logitech 3.0 advised that these modules should be considered part of the runtime system. The libraries do not provide all the features found in the Logitech libraries as a number of these features were MS-DOS related. Essentially the basic input/output, file system, string manipulation and conversion routines are provided. Access to DOSCALL, graphics, time and date are not as these were constrained by the limitations of MS-DOS.

The following libraries are contained within the base GNU Modula-2 libraries but are also Logitech-3.0 compatible: ASCII, Storage and MathLib0.

18.4.1 gm2-libs-pim/BitBlockOps

```
DEFINITION MODULE BitBlockOps ;

FROM SYSTEM IMPORT ADDRESS ;

(*
   BlockAnd - performs a bitwise AND on blocks
              [dest..dest+size-1] := [dest..dest+size-1] AND
                                     [src..src+size-1]
*)
```

(continues on next page)

(continued from previous page)

```
BlockAnd
PROCEDURE BlockAnd (dest, src: ADDRESS; size: CARDINAL) ;

(*
   BlockOr - performs a bitwise OR on blocks
             [dest..dest+size-1] := [dest..dest+size-1] OR
                                    [src..src+size-1]
*)

BlockOr
PROCEDURE BlockOr (dest, src: ADDRESS; size: CARDINAL) ;

(*
   BlockXor - performs a bitwise XOR on blocks
              [dest..dest+size-1] := [dest..dest+size-1] XOR
                                     [src..src+size-1]
*)

BlockXor
PROCEDURE BlockXor (dest, src: ADDRESS; size: CARDINAL) ;

(*
   BlockNot - performs a bitsize NOT on the block as defined
              by:  [dest..dest+size-1]
*)

BlockNot
PROCEDURE BlockNot (dest: ADDRESS; size: CARDINAL) ;

(*
   BlockShr - performs a block shift right of, count, bits.
              Where the block is defined as:
              [dest..dest+size-1].
              The block is considered to be an ARRAY OF BYTEs
              which is shifted, bit at a time over each byte in
              turn.  The left most byte is considered the byte
              located at the lowest address.
              If you require an endianness SHIFT use
              the SYSTEM.SHIFT procedure and declare the
              block as a POINTER TO set type.
*)

BlockShr
PROCEDURE BlockShr (dest: ADDRESS; size, count: CARDINAL) ;

(*
   BlockShl - performs a block shift left of, count, bits.
              Where the block is defined as:
              [dest..dest+size-1].
              The block is considered to be an ARRAY OF BYTEs
```

(continues on next page)

(continued from previous page)

```
              which is shifted, bit at a time over each byte in
              turn.  The left most byte is considered the byte
              located at the lowest address.
              If you require an endianness SHIFT use
              the SYSTEM.SHIFT procedure and declare the
              block as a POINTER TO set type.
*)

BlockShl
PROCEDURE BlockShl (dest: ADDRESS; size, count: CARDINAL) ;

(*
   BlockRor - performs a block rotate right of, count, bits.
              Where the block is defined as:
              [dest..dest+size-1].
              The block is considered to be an ARRAY OF BYTEs
              which is rotated, bit at a time over each byte in
              turn.  The left most byte is considered the byte
              located at the lowest address.
              If you require an endianness ROTATE use
              the SYSTEM.ROTATE procedure and declare the
              block as a POINTER TO set type.
*)

BlockRor
PROCEDURE BlockRor (dest: ADDRESS; size, count: CARDINAL) ;

(*
   BlockRol - performs a block rotate left of, count, bits.
              Where the block is defined as:
              [dest..dest+size-1].
              The block is considered to be an ARRAY OF BYTEs
              which is rotated, bit at a time over each byte in
              turn.  The left most byte is considered the byte
              located at the lowest address.
              If you require an endianness ROTATE use
              the SYSTEM.ROTATE procedure and declare the
              block as a POINTER TO set type.
*)

BlockRol
PROCEDURE BlockRol (dest: ADDRESS; size, count: CARDINAL) ;

END BitBlockOps.
```

18.4.2 gm2-libs-pim/BitByteOps

```
DEFINITION MODULE BitByteOps ;

FROM SYSTEM IMPORT BYTE ;

(*
   GetBits - returns the bits firstBit..lastBit from source.
             Bit 0 of byte maps onto the firstBit of source.
*)

GetBits
PROCEDURE GetBits (source: BYTE; firstBit, lastBit: CARDINAL) : BYTE ;

(*
   SetBits - sets bits in, byte, starting at, firstBit, and ending at,
             lastBit, with, pattern.  The bit zero of, pattern, will
             be placed into, byte, at position, firstBit.
*)

SetBits
PROCEDURE SetBits (VAR byte: BYTE; firstBit, lastBit: CARDINAL;
                   pattern: BYTE) ;

(*
   ByteAnd - returns a bitwise (left AND right)
*)

ByteAnd
PROCEDURE ByteAnd (left, right: BYTE) : BYTE ;

(*
   ByteOr - returns a bitwise (left OR right)
*)

ByteOr
PROCEDURE ByteOr (left, right: BYTE) : BYTE ;

(*
   ByteXor - returns a bitwise (left XOR right)
*)

ByteXor
PROCEDURE ByteXor (left, right: BYTE) : BYTE ;

(*
   ByteNot - returns a byte with all bits inverted.
*)

ByteNot
PROCEDURE ByteNot (byte: BYTE) : BYTE ;
```

(continues on next page)

(continued from previous page)

```
(*
   ByteShr - returns a, byte, which has been shifted, count
             bits to the right.
*)

ByteShr
PROCEDURE ByteShr (byte: BYTE; count: CARDINAL) : BYTE ;

(*
   ByteShl - returns a, byte, which has been shifted, count
             bits to the left.
*)

ByteShl
PROCEDURE ByteShl (byte: BYTE; count: CARDINAL) : BYTE ;

(*
   ByteSar - shift byte arthemetic right.  Preserves the top
             end bit and as the value is shifted right.
*)

ByteSar
PROCEDURE ByteSar (byte: BYTE; count: CARDINAL) : BYTE ;

(*
   ByteRor - returns a, byte, which has been rotated, count
             bits to the right.
*)

ByteRor
PROCEDURE ByteRor (byte: BYTE; count: CARDINAL) : BYTE ;

(*
   ByteRol - returns a, byte, which has been rotated, count
             bits to the left.
*)

ByteRol
PROCEDURE ByteRol (byte: BYTE; count: CARDINAL) : BYTE ;

(*
   HighNibble - returns the top nibble only from, byte.
                The top nibble of, byte, is extracted and
                returned in the bottom nibble of the return
                value.
*)

HighNibble
PROCEDURE HighNibble (byte: BYTE) : BYTE ;

(*
```

(continues on next page)

(continued from previous page)

```
   LowNibble - returns the low nibble only from, byte.
               The top nibble is replaced by zeros.
*)

LowNibble
PROCEDURE LowNibble (byte: BYTE) : BYTE ;

(*
   Swap - swaps the low and high nibbles in the, byte.
*)

Swap
PROCEDURE Swap (byte: BYTE) : BYTE ;

END BitByteOps.
```

18.4.3 gm2-libs-pim/BitWordOps

```
DEFINITION MODULE BitWordOps ;

FROM SYSTEM IMPORT WORD ;

(*
   GetBits - returns the bits firstBit..lastBit from source.
             Bit 0 of word maps onto the firstBit of source.
*)

GetBits
PROCEDURE GetBits (source: WORD; firstBit, lastBit: CARDINAL) : WORD ;

(*
   SetBits - sets bits in, word, starting at, firstBit, and ending at,
             lastBit, with, pattern.  The bit zero of, pattern, will
             be placed into, word, at position, firstBit.
*)

SetBits
PROCEDURE SetBits (VAR word: WORD; firstBit, lastBit: CARDINAL;
                   pattern: WORD) ;

(*
   WordAnd - returns a bitwise (left AND right)
*)

WordAnd
PROCEDURE WordAnd (left, right: WORD) : WORD ;

(*
   WordOr - returns a bitwise (left OR right)
*)
```

(continues on next page)

(continued from previous page)

```
WordOr
PROCEDURE WordOr (left, right: WORD) : WORD ;

(*
   WordXor - returns a bitwise (left XOR right)
*)

WordXor
PROCEDURE WordXor (left, right: WORD) : WORD ;

(*
   WordNot - returns a word with all bits inverted.
*)

WordNot
PROCEDURE WordNot (word: WORD) : WORD ;

(*
   WordShr - returns a, word, which has been shifted, count
             bits to the right.
*)

WordShr
PROCEDURE WordShr (word: WORD; count: CARDINAL) : WORD ;

(*
   WordShl - returns a, word, which has been shifted, count
             bits to the left.
*)

WordShl
PROCEDURE WordShl (word: WORD; count: CARDINAL) : WORD ;

(*
   WordSar - shift word arthemetic right.  Preserves the top
             end bit and as the value is shifted right.
*)

WordSar
PROCEDURE WordSar (word: WORD; count: CARDINAL) : WORD ;

(*
   WordRor - returns a, word, which has been rotated, count
             bits to the right.
*)

WordRor
PROCEDURE WordRor (word: WORD; count: CARDINAL) : WORD ;

(*
```

(continues on next page)

(continued from previous page)

```
   WordRol - returns a, word, which has been rotated, count
             bits to the left.
*)

WordRol
PROCEDURE WordRol (word: WORD; count: CARDINAL) : WORD ;

(*
   HighByte - returns the top byte only from, word.
              The byte is returned in the bottom byte
              in the return value.
*)

HighByte
PROCEDURE HighByte (word: WORD) : WORD ;

(*
   LowByte - returns the low byte only from, word.
             The byte is returned in the bottom byte
             in the return value.
*)

LowByte
PROCEDURE LowByte (word: WORD) : WORD ;

(*
   Swap - byte flips the contents of word.
*)

Swap
PROCEDURE Swap (word: WORD) : WORD ;

END BitWordOps.
```

18.4.4 gm2-libs-pim/BlockOps

```
DEFINITION MODULE BlockOps ;

FROM SYSTEM IMPORT ADDRESS ;

(*
   MoveBlockForward - moves, n, bytes from, src, to, dest.
                      Starts copying from src and keep copying
                      until, n, bytes have been copied.
*)

BlockMoveForward
PROCEDURE BlockMoveForward (dest, src: ADDRESS; n: CARDINAL) ;

(*
```

(continues on next page)

(continued from previous page)

```
   MoveBlockBackward - moves, n, bytes from, src, to, dest.
                       Starts copying from src+n and keeps copying
                       until, n, bytes have been copied.
                       The last datum to be copied will be the byte
                       at address, src.
*)

BlockMoveBackward
PROCEDURE BlockMoveBackward (dest, src: ADDRESS; n: CARDINAL) ;

(*
   BlockClear - fills, block..block+n-1, with zero's.
*)

BlockClear
PROCEDURE BlockClear (block: ADDRESS; n: CARDINAL) ;

(*
   BlockSet - fills, n, bytes starting at, block, with a pattern
              defined at address pattern..pattern+patternSize-1.
*)

BlockSet
PROCEDURE BlockSet (block: ADDRESS; n: CARDINAL;
                    pattern: ADDRESS; patternSize: CARDINAL) ;

(*
   BlockEqual - returns TRUE if the blocks defined, a..a+n-1, and,
                b..b+n-1 contain the same bytes.
*)

BlockEqual
PROCEDURE BlockEqual (a, b: ADDRESS; n: CARDINAL) : BOOLEAN ;

(*
   BlockPosition - searches for a pattern as defined by
                   pattern..patternSize-1 in the block,
                   block..block+blockSize-1.  It returns
                   the offset from block indicating the
                   first occurence of, pattern.
                   MAX(CARDINAL) is returned if no match
                   is detected.
*)

BlockPosition
PROCEDURE BlockPosition (block: ADDRESS; blockSize: CARDINAL;
                         pattern: ADDRESS; patternSize: CARDINAL) : CARDINAL ;

END BlockOps.
```

18.4.5 gm2-libs-pim/Break

```
DEFINITION MODULE Break ;

EXPORT QUALIFIED EnableBreak, DisableBreak, InstallBreak, UnInstallBreak ;

(*
   EnableBreak - enable the current break handler.
*)

EnableBreak
PROCEDURE EnableBreak ;

(*
   DisableBreak - disable the current break handler (and all
                  installed handlers).
*)

DisableBreak
PROCEDURE DisableBreak ;

(*
   InstallBreak - installs a procedure, p, to be invoked when
                  a ctrl-c is caught. Any number of these
                  procedures may be stacked. Only the top
                  procedure is run when ctrl-c is caught.
*)

InstallBreak
PROCEDURE InstallBreak (p: PROC) ;

(*
   UnInstallBreak - pops the break handler stack.
*)

UnInstallBreak
PROCEDURE UnInstallBreak ;

END Break.
```

18.4.6 gm2-libs-pim/CardinalIO

```
DEFINITION MODULE CardinalIO ;

EXPORT QUALIFIED Done,
                 ReadCardinal, WriteCardinal, ReadHex, WriteHex,
                 ReadLongCardinal, WriteLongCardinal, ReadLongHex,
                 WriteLongHex,
                 ReadShortCardinal, WriteShortCardinal, ReadShortHex,
                 WriteShortHex ;
```

(continues on next page)

(continued from previous page)

```
VAR
Done (var)
   Done: BOOLEAN ;

(*
   ReadCardinal - read an unsigned decimal number from the terminal.
                  The read continues until a space, newline, esc or
                  end of file is reached.
*)

ReadCardinal
PROCEDURE ReadCardinal (VAR c: CARDINAL) ;

(*
   WriteCardinal - writes the value, c, to the terminal and ensures
                   that at least, n, characters are written. The number
                   will be padded out by preceeding spaces if necessary.
*)

WriteCardinal
PROCEDURE WriteCardinal (c: CARDINAL; n: CARDINAL) ;

(*
   ReadHex - reads in an unsigned hexadecimal number from the terminal.
             The read continues until a space, newline, esc or
             end of file is reached.
*)

ReadHex
PROCEDURE ReadHex (VAR c: CARDINAL) ;

(*
   WriteHex - writes out a CARDINAL, c, in hexadecimal format padding
              with, n, characters (leading with '0')
*)

WriteHex
PROCEDURE WriteHex (c: CARDINAL; n: CARDINAL) ;

(*
   ReadLongCardinal - read an unsigned decimal number from the terminal.
                      The read continues until a space, newline, esc or
                      end of file is reached.
*)

ReadLongCardinal
PROCEDURE ReadLongCardinal (VAR c: LONGCARD) ;

(*
   WriteLongCardinal - writes the value, c, to the terminal and ensures
```

(continues on next page)

(continued from previous page)

```
                      that at least, n, characters are written. The number
                      will be padded out by preceeding spaces if necessary.
*)

WriteLongCardinal
PROCEDURE WriteLongCardinal (c: LONGCARD; n: CARDINAL) ;

(*
   ReadLongHex - reads in an unsigned hexadecimal number from the terminal.
                 The read continues until a space, newline, esc or
                 end of file is reached.
*)

ReadLongHex
PROCEDURE ReadLongHex (VAR c: LONGCARD) ;

(*
   WriteLongHex - writes out a LONGCARD, c, in hexadecimal format padding
                  with, n, characters (leading with '0')
*)

WriteLongHex
PROCEDURE WriteLongHex (c: LONGCARD; n: CARDINAL) ;

(*
   WriteShortCardinal - writes the value, c, to the terminal and ensures
                        that at least, n, characters are written. The number
                        will be padded out by preceeding spaces if necessary.
*)

WriteShortCardinal
PROCEDURE WriteShortCardinal (c: SHORTCARD; n: CARDINAL) ;

(*
   ReadShortCardinal - read an unsigned decimal number from the terminal.
                       The read continues until a space, newline, esc or
                       end of file is reached.
*)

ReadShortCardinal
PROCEDURE ReadShortCardinal (VAR c: SHORTCARD) ;

(*
   ReadShortHex - reads in an unsigned hexadecimal number from the terminal.
                  The read continues until a space, newline, esc or
                  end of file is reached.
*)

ReadShortHex
PROCEDURE ReadShortHex (VAR c: SHORTCARD) ;
```

(continues on next page)

(continued from previous page)

```
(*
   WriteShortHex - writes out a SHORTCARD, c, in hexadecimal format padding
                   with, n, characters (leading with '0')
*)

WriteShortHex
PROCEDURE WriteShortHex (c: SHORTCARD; n: CARDINAL) ;

END CardinalIO.
```

18.4.7 gm2-libs-pim/Conversions

```
DEFINITION MODULE Conversions ;

EXPORT QUALIFIED ConvertOctal, ConvertHex, ConvertCardinal,
                 ConvertInteger, ConvertLongInt, ConvertShortInt ;

(*
   ConvertOctal - converts a CARDINAL, num, into an octal/hex/decimal
                  string and right justifies the string. It adds
                  spaces rather than '0' to pad out the string
                  to len characters.

                  If the length of str is < num then the number is
                  truncated on the right.
*)

ConvertOctal
PROCEDURE ConvertOctal    (num, len: CARDINAL; VAR str: ARRAY OF CHAR) ;
ConvertHex
PROCEDURE ConvertHex      (num, len: CARDINAL; VAR str: ARRAY OF CHAR) ;
ConvertCardinal
PROCEDURE ConvertCardinal (num, len: CARDINAL; VAR str: ARRAY OF CHAR) ;

(*
   The INTEGER counterparts will add a '-' if, num, is <0
*)

ConvertInteger
PROCEDURE ConvertInteger  (num: INTEGER; len: CARDINAL; VAR str: ARRAY OF CHAR) ;
ConvertLongInt
PROCEDURE ConvertLongInt  (num: LONGINT; len: CARDINAL; VAR str: ARRAY OF CHAR) ;
ConvertShortInt
PROCEDURE ConvertShortInt (num: SHORTINT; len: CARDINAL; VAR str: ARRAY OF CHAR) ;

END Conversions.
```

18.4.8 gm2-libs-pim/DebugPMD

```
DEFINITION MODULE DebugPMD ;

END DebugPMD.
```

18.4.9 gm2-libs-pim/DebugTrace

```
DEFINITION MODULE DebugTrace ;

END DebugTrace.
```

18.4.10 gm2-libs-pim/Delay

```
DEFINITION MODULE Delay ;

EXPORT QUALIFIED Delay ;

(*
   milliSec - delays the program by approximately, milliSec, milliseconds.
*)

Delay
PROCEDURE Delay (milliSec: INTEGER) ;

END Delay.
```

18.4.11 gm2-libs-pim/Display

```
DEFINITION MODULE Display ;

EXPORT QUALIFIED Write ;

(*
   Write - display a character to the stdout.
           ASCII.EOL moves to the beginning of the next line.
           ASCII.del erases the character to the left of the cursor.
*)

Write
PROCEDURE Write (ch: CHAR) ;

END Display.
```

18.4.12 gm2-libs-pim/ErrorCode

```
DEFINITION MODULE ErrorCode ;

EXPORT QUALIFIED SetErrorCode, GetErrorCode, ExitToOS ;

(*
   SetErrorCode - sets the exit value which will be used if
                  the application terminates normally.
*)

SetErrorCode
PROCEDURE SetErrorCode (value: INTEGER) ;

(*
   GetErrorCode - returns the current value to be used upon
                  application termination.
*)

GetErrorCode
PROCEDURE GetErrorCode (VAR value: INTEGER) ;

(*
   ExitToOS - terminate the application and exit returning
              the last value set by SetErrorCode to the OS.
*)

ExitToOS
PROCEDURE ExitToOS ;

END ErrorCode.
```

18.4.13 gm2-libs-pim/FileSystem

```
DEFINITION MODULE FileSystem ;

(*  Use this module sparingly, FIO or the ISO file modules have a
    much cleaner interface.  *)

FROM SYSTEM IMPORT WORD, BYTE, ADDRESS ;
IMPORT FIO ;
FROM DynamicStrings IMPORT String ;

EXPORT QUALIFIED File, Response, Flag, FlagSet,

                 Create, Close, Lookup, Rename, Delete,
                 SetRead, SetWrite, SetModify, SetOpen,
                 Doio, SetPos, GetPos, Length, Reset,

                 ReadWord, ReadChar, ReadByte, ReadNBytes,
```

(continues on next page)

(continued from previous page)

```
                WriteWord, WriteChar, WriteByte, WriteNBytes ;

TYPE
File (type)
   File = RECORD
             res     : Response ;
             flags   : FlagSet ;
             eof     : BOOLEAN ;
             lastWord: WORD ;
             lastByte: BYTE ;
             fio     : FIO.File ;
             highpos,
             lowpos  : CARDINAL ;
             name    : String ;
END (type)
          END ;

Flag (type)
   Flag = (
            read,        (* read access mode *)
            write,       (* write access mode *)
            modify,
            truncate,    (* truncate file when closed *)
            again,       (* reread the last character *)
            temporary,   (* file is temporary *)
            opened       (* file has been opened *)
           );

FlagSet (type)
   FlagSet = SET OF Flag;

Response (type)
   Response = (done, notdone, notsupported, callerror,
               unknownfile, paramerror, toomanyfiles,
userdeverror) (type)
               userdeverror) ;

Command (type)
   Command = (create, close, lookup, rename, delete,
              setread, setwrite, setmodify, setopen,
              doio, setpos, getpos, length) ;

(*
   Create - creates a temporary file. To make the file perminant
            the file must be renamed.
*)

Create
PROCEDURE Create (VAR f: File) ;

(*
```

(continues on next page)

(continued from previous page)

```
   Close - closes an open file.
*)

Close
PROCEDURE Close (f: File) ;

(*
   Lookup - looks for a file, filename. If the file is found
            then, f, is opened. If it is not found and, newFile,
            is TRUE then a new file is created and attached to, f.
            If, newFile, is FALSE and no file was found then f.res
            is set to notdone.
*)

Lookup
PROCEDURE Lookup (VAR f: File; filename: ARRAY OF CHAR; newFile: BOOLEAN) ;

(*
   Rename - rename a file and change a temporary file to a permanent
            file. f.res is set appropriately.
*)

Rename
PROCEDURE Rename (VAR f: File; newname: ARRAY OF CHAR) ;

(*
   Delete - deletes a file, name, and sets the f.res field.
            f.res is set appropriately.
*)

Delete
PROCEDURE Delete (name: ARRAY OF CHAR; VAR f: File) ;

(*
   ReadWord - reads a WORD, w, from file, f.
              f.res is set appropriately.
*)

ReadWord
PROCEDURE ReadWord (VAR f: File; VAR w: WORD) ;

(*
   WriteWord - writes one word to a file, f.
               f.res is set appropriately.
*)

WriteWord
PROCEDURE WriteWord (VAR f: File; w: WORD) ;

(*
   ReadChar - reads one character from a file, f.
```

(continues on next page)

(continued from previous page)

```
*)

ReadChar
PROCEDURE ReadChar (VAR f: File; VAR ch: CHAR) ;

(*
   WriteChar - writes a character, ch, to a file, f.
               f.res is set appropriately.
*)

WriteChar
PROCEDURE WriteChar (VAR f: File; ch: CHAR) ;

(*
   ReadByte - reads a BYTE, b, from file, f.
              f.res is set appropriately.
*)

ReadByte
PROCEDURE ReadByte (VAR f: File; VAR b: BYTE) ;

(*
   WriteByte - writes one BYTE, b, to a file, f.
               f.res is set appropriately.
*)

WriteByte
PROCEDURE WriteByte (VAR f: File; b: BYTE) ;

(*
   ReadNBytes - reads a sequence of bytes from a file, f.
*)

ReadNBytes
PROCEDURE ReadNBytes (VAR f: File; a: ADDRESS; amount: CARDINAL;
                      VAR actuallyRead: CARDINAL) ;

(*
   WriteNBytes - writes a sequence of bytes to file, f.
*)

WriteNBytes
PROCEDURE WriteNBytes (VAR f: File; a: ADDRESS; amount: CARDINAL;
                       VAR actuallyWritten: CARDINAL) ;

(*
   Again - returns the last character read to the internal buffer
           so that it can be read again.
*)

Again
```

(continues on next page)

(continued from previous page)

```
PROCEDURE Again (VAR f: File) ;

(*
   SetRead - puts the file, f, into the read state.
             The file position is unchanged.
*)

SetRead
PROCEDURE SetRead (VAR f: File) ;

(*
   SetWrite - puts the file, f, into the write state.
              The file position is unchanged.
*)

SetWrite
PROCEDURE SetWrite (VAR f: File) ;

(*
   SetModify - puts the file, f, into the modify state.
               The file position is unchanged but the file can be
               read and written.
*)

SetModify
PROCEDURE SetModify (VAR f: File) ;

(*
   SetOpen - places a file, f, into the open state. The file may
             have been in the read/write/modify state before and
             in which case the previous buffer contents are flushed
             and the file state is reset to open. The position is
             unaltered.
*)

SetOpen
PROCEDURE SetOpen (VAR f: File) ;

(*
   Reset - places a file, f, into the open state and reset the
           position to the start of the file.
*)

Reset
PROCEDURE Reset (VAR f: File) ;

(*
   SetPos - lseek to a position within a file.
*)

SetPos
```

(continues on next page)

(continued from previous page)

```
PROCEDURE SetPos (VAR f: File; high, low: CARDINAL) ;

(*
   GetPos - return the position within a file.
*)

GetPos
PROCEDURE GetPos (VAR f: File; VAR high, low: CARDINAL) ;

(*
   Length - returns the length of file, in, high, and, low.
*)

Length
PROCEDURE Length (VAR f: File; VAR high, low: CARDINAL) ;

(*
   Doio - effectively flushes a file in write mode, rereads the
          current buffer from disk if in read mode and writes
          and rereads the buffer if in modify mode.
*)

Doio
PROCEDURE Doio (VAR f: File) ;

(*
   FileNameChar - checks to see whether the character, ch, is
                  legal in a filename. nul is returned if the
                  character was illegal.
*)

FileNameChar
PROCEDURE FileNameChar (ch: CHAR) ;

END FileSystem.
```

18.4.14 gm2-libs-pim/FloatingUtilities

```
DEFINITION MODULE FloatingUtilities ;

EXPORT QUALIFIED Frac, Round, Float, Trunc,
                 Fracl, Roundl, Floatl, Truncl ;

(*
   Frac - returns the fractional component of, r.
*)

Frac
PROCEDURE Frac (r: REAL) : REAL ;
```

(continues on next page)

(continued from previous page)

```
(*
   Int - returns the integer part of r. It rounds the value towards zero.
*)

Int
PROCEDURE Int (r: REAL) : INTEGER ;

(*
   Round - returns the number rounded to the nearest integer.
*)

Round
PROCEDURE Round (r: REAL) : INTEGER ;

(*
   Float - returns a REAL value corresponding to, i.
*)

Float
PROCEDURE Float (i: INTEGER) : REAL ;

(*
   Trunc - round to the nearest integer not larger in absolute
           value.
*)

Trunc
PROCEDURE Trunc (r: REAL) : INTEGER ;

(*
   Fracl - returns the fractional component of, r.
*)

Fracl
PROCEDURE Fracl (r: LONGREAL) : LONGREAL ;

(*
   Intl - returns the integer part of r. It rounds the value towards zero.
*)

Intl
PROCEDURE Intl (r: LONGREAL) : LONGINT ;

(*
   Roundl - returns the number rounded to the nearest integer.
*)

Roundl
PROCEDURE Roundl (r: LONGREAL) : LONGINT ;

(*
```

(continues on next page)

(continued from previous page)

```
   Floatl - returns a REAL value corresponding to, i.
*)

Floatl
PROCEDURE Floatl (i: INTEGER) : LONGREAL ;

(*
   Truncl - round to the nearest integer not larger in absolute
            value.
*)

Truncl
PROCEDURE Truncl (r: LONGREAL) : LONGINT ;

END FloatingUtilities.
```

18.4.15 gm2-libs-pim/InOut

```
DEFINITION MODULE InOut ;

IMPORT ASCII ;
FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED EOL, Done, termCH, OpenInput, OpenOutput,
                 CloseInput, CloseOutput,
                 Read, ReadString, ReadInt, ReadCard,
                 Write, WriteLn, WriteString, WriteInt, WriteCard,
                 WriteOct, WriteHex,
                 ReadS, WriteS ;

CONST
EOL  (const)
   EOL = ASCII.EOL ;

VAR
Done   (var)
   Done  : BOOLEAN ;
termCH (var)
   termCH: CHAR ;

(*
   OpenInput - reads a string from stdin as the filename for reading.
               If the filename ends with `.' then it appends the defext
               extension. The global variable Done is set if all
               was successful.
*)

OpenInput
PROCEDURE OpenInput (defext: ARRAY OF CHAR) ;

(*
```

(continues on next page)

(continued from previous page)

```
   CloseInput - closes an opened input file and returns input back to
                StdIn.
*)

CloseInput
PROCEDURE CloseInput ;

(*
   OpenOutput - reads a string from stdin as the filename for writing.
                If the filename ends with `.' then it appends the defext
                extension. The global variable Done is set if all
                was successful.
*)

OpenOutput
PROCEDURE OpenOutput (defext: ARRAY OF CHAR) ;

(*
   CloseOutput - closes an opened output file and returns output back to
                 StdOut.
*)

CloseOutput
PROCEDURE CloseOutput ;

(*
   Read - reads a single character from the current input file.
          Done is set to FALSE if end of file is reached or an
          error occurs.
*)

Read
PROCEDURE Read (VAR ch: CHAR) ;

(*
   ReadString - reads a sequence of characters. Leading white space
                is ignored and the string is terminated with a character
                <= ' '
*)

ReadString
PROCEDURE ReadString (VAR s: ARRAY OF CHAR) ;

(*
   WriteString - writes a string to the output file.
*)

WriteString
PROCEDURE WriteString (s: ARRAY OF CHAR) ;

(*
```

(continues on next page)

(continued from previous page)

```
   Write - writes out a single character, ch, to the current output file.
*)

Write
PROCEDURE Write (ch: CHAR) ;

(*
   WriteLn - writes a newline to the output file.
*)

WriteLn
PROCEDURE WriteLn ;

(*
   ReadInt - reads a string and converts it into an INTEGER, x.
             Done is set if an INTEGER is read.
*)

ReadInt
PROCEDURE ReadInt (VAR x: INTEGER) ;

(*
   ReadInt - reads a string and converts it into an INTEGER, x.
             Done is set if an INTEGER is read.
*)

ReadCard
PROCEDURE ReadCard (VAR x: CARDINAL) ;

(*
   WriteCard - writes the CARDINAL, x, to the output file. It ensures
               that the number occupies, n, characters. Leading spaces
               are added if required.
*)

WriteCard
PROCEDURE WriteCard (x, n: CARDINAL) ;

(*
   WriteInt - writes the INTEGER, x, to the output file. It ensures
              that the number occupies, n, characters. Leading spaces
              are added if required.
*)

WriteInt
PROCEDURE WriteInt (x: INTEGER; n: CARDINAL) ;

(*
   WriteOct - writes the CARDINAL, x, to the output file in octal.
              It ensures that the number occupies, n, characters.
              Leading spaces are added if required.
```

(continues on next page)

(continued from previous page)

```
*)

WriteOct
PROCEDURE WriteOct (x, n: CARDINAL) ;

(*
   WriteHex - writes the CARDINAL, x, to the output file in hexadecimal.
              It ensures that the number occupies, n, characters.
              Leading spaces are added if required.
*)

WriteHex
PROCEDURE WriteHex (x, n: CARDINAL) ;

(*
   ReadS - returns a string which has is a sequence of characters.
           Leading white space is ignored and string is terminated
           with a character <= ' '.
*)

ReadS
PROCEDURE ReadS () : String ;

(*
   WriteS - writes a String to the output device.
            It returns the string, s.
*)

WriteS
PROCEDURE WriteS (s: String) : String ;

END InOut.
```

18.4.16 gm2-libs-pim/Keyboard

```
DEFINITION MODULE Keyboard ;

EXPORT QUALIFIED Read, KeyPressed ;

(*
   Read - reads a character from StdIn. If necessary it will wait
          for a key to become present on StdIn.
*)

Read
PROCEDURE Read (VAR ch: CHAR) ;

(*
   KeyPressed - returns TRUE if a character can be read from StdIn
                without blocking the caller.
```

(continues on next page)

(continued from previous page)

```
*)

KeyPressed
PROCEDURE KeyPressed () : BOOLEAN ;

END Keyboard.
```

18.4.17 gm2-libs-pim/LongIO

```
DEFINITION MODULE LongIO ;

EXPORT QUALIFIED Done, ReadLongInt, WriteLongInt ;

VAR
Done (var)
   Done: BOOLEAN ;

ReadLongInt
PROCEDURE ReadLongInt (VAR i: LONGINT) ;
WriteLongInt
PROCEDURE WriteLongInt (i: LONGINT; n: CARDINAL) ;

END LongIO.
```

18.4.18 gm2-libs-pim/NumberConversion

```
DEFINITION MODULE NumberConversion ;

(* --fixme-- finish this.  *)

END NumberConversion.
```

18.4.19 gm2-libs-pim/Random

```
DEFINITION MODULE Random ;

FROM SYSTEM IMPORT BYTE ;
EXPORT QUALIFIED Randomize, RandomInit, RandomBytes, RandomCard, RandomInt, RandomReal,␣
↪RandomLongReal ;

(*
   Randomize - initialize the random number generator with a seed
               based on the microseconds.
*)
```

(continues on next page)

(continued from previous page)

```
Randomize
PROCEDURE Randomize ;

(*
   RandomInit - initialize the random number generator with value, seed.
*)

RandomInit
PROCEDURE RandomInit (seed: CARDINAL) ;

(*
   RandomBytes - fills in an array with random values.
*)

RandomBytes
PROCEDURE RandomBytes (VAR a: ARRAY OF BYTE) ;

(*
   RandomInt - return an INTEGER in the range 0..bound-1
*)

RandomInt
PROCEDURE RandomInt (bound: INTEGER) : INTEGER ;

(*
   RandomCard - return a CARDINAL in the range 0..bound-1
*)

RandomCard
PROCEDURE RandomCard (bound: CARDINAL) : CARDINAL ;

(*
   RandomReal - return a REAL number in the range 0.0..1.0
*)

RandomReal
PROCEDURE RandomReal () : REAL ;

(*
   RandomLongReal - return a LONGREAL number in the range 0.0..1.0
*)

RandomLongReal
PROCEDURE RandomLongReal () : LONGREAL ;

END Random.
```

18.4.20 gm2-libs-pim/RealConversions

```
DEFINITION MODULE RealConversions ;

EXPORT QUALIFIED SetNoOfExponentDigits,
                 RealToString, StringToReal,
                 LongRealToString, StringToLongReal ;

(*
   SetNoOfExponentDigits - sets the number of exponent digits to be
                           used during future calls of LongRealToString
                           and RealToString providing that the width
                           is sufficient.
                           If this value is set to 0 (the default) then
                           the number digits used is the minimum necessary.
*)

SetNoOfExponentDigits
PROCEDURE SetNoOfExponentDigits (places: CARDINAL) ;

(*
   RealToString - converts a real, r, into a right justified string, str.
                  The number of digits to the right of the decimal point
                  is given in, digits.  The value, width, represents the
                  maximum number of characters to be used in the string,
                  str.

                  If digits is negative then exponent notation is used
                  whereas if digits is positive then fixed point notation
                  is used.

                  If, r, is less than 0.0 then a '-' preceeds the value,
                  str.  However, if, r, is >= 0.0 a '+' is not added.

                  If the conversion of, r, to a string requires more
                  than, width, characters then the string, str, is set
                  to a nul string and, ok is assigned FALSE.

                  For fixed point notation the minimum width required is
                  ABS(width)+8

                  For exponent notation the minimum width required is
                  ABS(digits)+2+log10(magnitude).

                  if r is a NaN then the string 'nan' is returned formatted and
                  ok will be FALSE.
*)

RealToString
PROCEDURE RealToString (r: REAL; digits, width: INTEGER;
                        VAR str: ARRAY OF CHAR; VAR ok: BOOLEAN) ;
```

(continues on next page)

(continued from previous page)

```
(*
   LongRealToString - converts a real, r, into a right justified string, str.
                      The number of digits to the right of the decimal point
                      is given in, digits. The value, width, represents the
                      maximum number of characters to be used in the string,
                      str.

                      If digits is negative then exponent notation is used
                      whereas if digits is positive then fixed point notation
                      is used.

                      If, r, is less than 0.0 then a '-' preceeds the value,
                      str. However, if, r, is >= 0.0 a '+' is not added.

                      If the conversion of, r, to a string requires more
                      than, width, characters then the string, str, is set
                      to a nul string and, ok is assigned FALSE.

                      For fixed point notation the minimum width required is
                      ABS(width)+8

                      For exponent notation the minimum width required is
                      ABS(digits)+2+log10(magnitude).

                      Examples:
                      RealToString(100.0, 10, 10, a, ok)       ->  '100.000000'
                      RealToString(100.0, -5, 12, a, ok)       ->  '  1.00000E+2'

                      RealToString(123.456789, 10, 10, a, ok)  ->  '123.456789'
                      RealToString(123.456789, -5, 13, a, ok)  ->  '    1.23456E+2'

                      RealToString(123.456789, -2, 15, a, ok)  ->  '          1.23E+2'

                      if r is a NaN then the string 'nan' is returned formatted and
                      ok will be FALSE.
*)

LongRealToString
PROCEDURE LongRealToString (r: LONGREAL; digits, width: INTEGER;
                            VAR str: ARRAY OF CHAR; VAR ok: BOOLEAN) ;

(*
   StringToReal - converts, str, into a REAL, r. The parameter, ok, is
                  set to TRUE if the conversion was successful.
*)

StringToReal
PROCEDURE StringToReal (str: ARRAY OF CHAR; VAR r: REAL; VAR ok: BOOLEAN) ;

(*
   StringToLongReal - converts, str, into a LONGREAL, r. The parameter, ok, is
```

(continues on next page)

(continued from previous page)

```
                    set to TRUE if the conversion was successful.
*)

StringToLongReal
PROCEDURE StringToLongReal (str: ARRAY OF CHAR; VAR r: LONGREAL; VAR ok: BOOLEAN) ;

END RealConversions.
```

18.4.21 gm2-libs-pim/RealInOut

```
DEFINITION MODULE RealInOut ;

EXPORT QUALIFIED SetNoOfDecimalPlaces,
                 ReadReal, WriteReal, WriteRealOct,
                 ReadLongReal, WriteLongReal, WriteLongRealOct,
                 ReadShortReal, WriteShortReal, WriteShortRealOct,
                 Done ;

CONST
DefaultDecimalPlaces  (const)
   DefaultDecimalPlaces = 6 ;

VAR
Done (var)
   Done: BOOLEAN ;

(*
   SetNoOfDecimalPlaces - number of decimal places WriteReal and
                          WriteLongReal should emit.  This procedure
                          can be used to override the default
                          DefaultDecimalPlaces constant.
*)

SetNoOfDecimalPlaces
PROCEDURE SetNoOfDecimalPlaces (places: CARDINAL) ;

(*
   ReadReal - reads a real number, legal syntaxes include:
              100, 100.0, 100e0, 100E0, 100E-1, E2, +1E+2, 1e+2
*)

ReadReal
PROCEDURE ReadReal (VAR x: REAL) ;

(*
   WriteReal - writes a real to the terminal. The real number
               is right justified and, n, is the minimum field
               width.
*)
```

(continues on next page)

(continued from previous page)

```
WriteReal
PROCEDURE WriteReal (x: REAL; n: CARDINAL) ;

(*
   WriteRealOct - writes the real to terminal in octal words.
*)

WriteRealOct
PROCEDURE WriteRealOct (x: REAL) ;

(*
   ReadLongReal - reads a LONGREAL number, legal syntaxes include:
                  100, 100.0, 100e0, 100E0, 100E-1, E2, +1E+2, 1e+2
*)

ReadLongReal
PROCEDURE ReadLongReal (VAR x: LONGREAL) ;

(*
   WriteLongReal - writes a LONGREAL to the terminal. The real number
                   is right justified and, n, is the minimum field
                   width.
*)

WriteLongReal
PROCEDURE WriteLongReal (x: LONGREAL; n: CARDINAL) ;

(*
   WriteLongRealOct - writes the LONGREAL to terminal in octal words.
*)

WriteLongRealOct
PROCEDURE WriteLongRealOct (x: LONGREAL) ;

(*
   ReadShortReal - reads a SHORTREAL number, legal syntaxes include:
                   100, 100.0, 100e0, 100E0, 100E-1, E2, +1E+2, 1e+2
*)

ReadShortReal
PROCEDURE ReadShortReal (VAR x: SHORTREAL) ;

(*
   WriteShortReal - writes a SHORTREAL to the terminal. The real number
                    is right justified and, n, is the minimum field
                    width.
*)

WriteShortReal
PROCEDURE WriteShortReal (x: SHORTREAL; n: CARDINAL) ;
```

(continues on next page)

(continued from previous page)

```
(*
   WriteShortRealOct - writes the SHORTREAL to terminal in octal words.
*)

WriteShortRealOct
PROCEDURE WriteShortRealOct (x: SHORTREAL) ;

END RealInOut.
```

18.4.22 gm2-libs-pim/Strings

```
DEFINITION MODULE Strings ;

EXPORT QUALIFIED Assign, Insert, Delete, Pos, Copy, ConCat, Length,
                 CompareStr ;

(*
   Assign - dest := source.
*)

Assign
PROCEDURE Assign (VAR dest: ARRAY OF CHAR; source: ARRAY OF CHAR) ;

(*
   Insert - insert the string, substr, into str at position, index.
            substr, is added to the end of, str, if, index >= length(str)
*)

Insert
PROCEDURE Insert (substr: ARRAY OF CHAR; VAR str: ARRAY OF CHAR;
                  index: CARDINAL) ;

(*
   Delete - delete len characters from, str, starting at, index.
*)

Delete
PROCEDURE Delete (VAR str: ARRAY OF CHAR; index: CARDINAL; length: CARDINAL) ;

(*
   Pos - return the first position of, substr, in, str.
*)

Pos
PROCEDURE Pos (substr, str: ARRAY OF CHAR) : CARDINAL ;

(*
   Copy - copy at most, length, characters in, substr, to, str,
          starting at position, index.
*)
```

(continues on next page)

(continued from previous page)

```
Copy
PROCEDURE Copy (str: ARRAY OF CHAR;
                index, length: CARDINAL; VAR result: ARRAY OF CHAR) ;

(*
   ConCat - concatenates two strings, s1, and, s2
            and places the result into, dest.
*)

ConCat
PROCEDURE ConCat (s1, s2: ARRAY OF CHAR; VAR dest: ARRAY OF CHAR) ;

(*
   Length - return the length of string, s.
*)

Length
PROCEDURE Length (s: ARRAY OF CHAR) : CARDINAL ;

(*
   CompareStr - compare two strings, left, and, right.
*)

CompareStr
PROCEDURE CompareStr (left, right: ARRAY OF CHAR) : INTEGER ;

END Strings.
```

18.4.23 gm2-libs-pim/Termbase

```
DEFINITION MODULE Termbase ;

(*
   Initially the read routines from Keyboard and the
   write routine from Display is assigned to the Read,
   KeyPressed and Write procedures.
*)

EXPORT QUALIFIED ReadProcedure, StatusProcedure, WriteProcedure,
                 AssignRead, AssignWrite, UnAssignRead, UnAssignWrite,
                 Read, KeyPressed, Write ;

TYPE
ReadProcedure (type)
   ReadProcedure = PROCEDURE (VAR CHAR) ;
WriteProcedure (type)
   WriteProcedure = PROCEDURE (CHAR) ;
StatusProcedure (type)
   StatusProcedure = PROCEDURE () : BOOLEAN ;
```

(continues on next page)

(continued from previous page)

```
(*
   AssignRead - assigns a read procedure and status procedure for terminal
                input. Done is set to TRUE if successful. Subsequent
                Read and KeyPressed calls are mapped onto the user supplied
                procedures. The previous read and status procedures are
                uncovered and reused after UnAssignRead is called.
*)

AssignRead
PROCEDURE AssignRead (rp: ReadProcedure; sp: StatusProcedure;
                      VAR Done: BOOLEAN) ;

(*
   UnAssignRead - undo the last call to AssignRead and set Done to TRUE
                  on success.
*)

UnAssignRead
PROCEDURE UnAssignRead (VAR Done: BOOLEAN) ;

(*
   Read - reads a single character using the currently active read
          procedure.
*)

Read
PROCEDURE Read (VAR ch: CHAR) ;

(*
   KeyPressed - returns TRUE if a character is available to be read.
*)

KeyPressed
PROCEDURE KeyPressed () : BOOLEAN ;

(*
   AssignWrite - assigns a write procedure for terminal output.
                 Done is set to TRUE if successful. Subsequent
                 Write calls are mapped onto the user supplied
                 procedure. The previous write procedure is
                 uncovered and reused after UnAssignWrite is called.
*)

AssignWrite
PROCEDURE AssignWrite (wp: WriteProcedure; VAR Done: BOOLEAN) ;

(*
   UnAssignWrite - undo the last call to AssignWrite and set Done to TRUE
                   on success.
*)
```

(continues on next page)

(continued from previous page)

```
UnAssignWrite
PROCEDURE UnAssignWrite (VAR Done: BOOLEAN) ;

(*
   Write - writes a single character using the currently active write
           procedure.
*)

Write
PROCEDURE Write (VAR ch: CHAR) ;

END Termbase.
```

18.4.24 gm2-libs-pim/Terminal

```
DEFINITION MODULE Terminal ;

(*
   It provides simple terminal input output
   routines which all utilize the TermBase module.
*)

EXPORT QUALIFIED Read, KeyPressed, ReadAgain, ReadString, Write,
                 WriteString, WriteLn ;

(*
   Read - reads a single character.
*)

Read
PROCEDURE Read (VAR ch: CHAR) ;

(*
   KeyPressed - returns TRUE if a character can be read without blocking
                the caller.
*)

KeyPressed
PROCEDURE KeyPressed () : BOOLEAN ;

(*
   ReadString - reads a sequence of characters.
                Tabs are expanded into 8 spaces and <cr> or <lf> terminates
                the string.
*)

ReadString
PROCEDURE ReadString (VAR s: ARRAY OF CHAR) ;
```

(continues on next page)

(continued from previous page)

```
(*
   ReadAgain - makes the last character readable again.
*)

ReadAgain
PROCEDURE ReadAgain ;

(*
   Write - writes a single character to the Termbase module.
*)

Write
PROCEDURE Write (ch: CHAR) ;

(*
   WriteString - writes out a string which is terminated by a <nul>
                 character or the end of string HIGH(s).
*)

WriteString
PROCEDURE WriteString (s: ARRAY OF CHAR) ;

(*
   WriteLn - writes a lf character.
*)

WriteLn
PROCEDURE WriteLn ;

END Terminal.
```

18.4.25 gm2-libs-pim/TimeDate

```
DEFINITION MODULE TimeDate ;

(*
   Legacy compatibility - you are advised to use cleaner
   designed modules based on 'man 3 strtime'
   and friends for new projects as the day value here is ugly.
   [it was mapped onto MSDOS pre 2000].
*)

EXPORT QUALIFIED Time, GetTime, SetTime, CompareTime, TimeToZero,
                 TimeToString ;

TYPE
(*
   day holds:  bits 0..4 = day of month (1..31)
                    5..8 = month of year (1..12)
                    9..  = year - 1900
```

(continues on next page)

(continued from previous page)

```
   minute holds:    hours * 60 + minutes
   millisec holds:  seconds * 1000 + millisec
                    which is reset to 0 every minute
*)

   Time = RECORD
             day, minute, millisec: CARDINAL ;
          END ;

(*
   GetTime - returns the current date and time.
*)

GetTime
PROCEDURE GetTime (VAR curTime: Time) ;

(*
   SetTime - does nothing, but provides compatibility with
             the Logitech-3.0 library.
*)

SetTime
PROCEDURE SetTime (curTime: Time) ;

(*
   CompareTime - compare two dates and time which returns:

                 -1  if t1 < t2
                  0  if t1 = t2
                  1  if t1 > t2
*)

CompareTime
PROCEDURE CompareTime (t1, t2: Time) : INTEGER ;

(*
   TimeToZero - initializes, t, to zero.
*)

TimeToZero
PROCEDURE TimeToZero (VAR t: Time) ;

(*
   TimeToString - convert time, t, to a string.
                  The string, s, should be at least 19 characters
                  long and the returned string will be

                  yyyy-mm-dd hh:mm:ss
*)

TimeToString
```

(continues on next page)

(continued from previous page)

```
PROCEDURE TimeToString (t: Time; VAR s: ARRAY OF CHAR) ;

END TimeDate.
```

18.5 Indices