makeinfo texinfo-format-buffer

The GNU Modula-2 front end to GCC

gm2-1.0

Gaius Mulley

Last updated Thursday 25 November 2010

"And then one day you find ten years have got behind you", Pink Floyd

1 Using GNU Modula-2

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 '`gm2-1.0`' or '`gm2`' for short. This corresponds to GCC version 4.1.2 and GNU Modula-2 version 1.0.

1.1 What is GNU Modula-2

GNU Modula-2 is a front end `http://gcc.gnu.org/frontends.html` for GCC. GCC is a retargetable C compiler which has been ported to a large number of architectures and operating systems. GNU Modula-2 utilizes the back end of GCC and replaces the C language front end with a Modula-2 front end. Consequently GNU Modula-2 has been built on i[3456]86 GNU/Linux, i[3456]86 BSD, Opteron LP64 GNU/Linux and sparc GNU/Linux systems. It has also been built as a cross compiler for MinGW, StrongARM GNU/Linux and ATMega8 microprocessor.

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).

There are currently three sets of libraries. The 'Programming in Modula-2' libraries, the 'University of ULM libraries' and the ISO libraries. The ISO libraries are still being written, however all definition modules for the three library sets are contained within this document.

1.2 Why use GNU Modula-2

This section is not designed to generate a language war, but rather map out some of the advantages of using GNU Modula-2 rather than translate Modula-2 sources into another language.

It is expected that the primary purpose of GNU Modula-2 will be to compile legacy code. Currently there are only a few commercial Modula-2 compilers being actively maintained. Code which was written ten or fifteen years ago may still be compiled by older commercial (possibly unmaintained) compilers. While the 32 bit x86 remains these compilers presumably can be run in compatibility mode (some compilers only produced 16 bit code). Time is running out as the computing industry is switching to 64 microprocessors. While x86 emulation or 16 bit backwards compatibility is always possible it has some serious drawbacks. In order for the older source to run natively the source code will either have to be translated into another high level language or alternatively a Modula-2 compiler which can target these new generation of microprocessors will have to be acquired. GNU Modula-2 builds and passes all its regression tests on Debian Pure 64 (LP64 architecture), 64 bit Solaris, 32 bit x86 GNU/Linux (Suse, Debian, stable and unstable) and 32 bit x86 FreeBSD. GNU Modula-2 has also been configured as a cross compiler for embedded microprocessors such as the ATMega8 and StrongArm.

GNU Modula-2 also has the advantage of being closely tied to GCC. Not only does this produce excellent code and excellent architectural and operating system coverage but it also utilises many of the GCC features. For example GNU Modula-2 can invoke the C preprocessor to manage conditional compilation; in-lining of `SYSTEM` procedures, intrinsic functions, memory copying routines are also exploited; access to assembly language using GCC syntax is also provided. GNU Modula-2 also support sets of any ordinal type (memory permitting). The compiler can also easily generate shared library modules and it has an automatic method of producing a swig interface file from a definition module. This means that after a few command line invocations the compiler can generate Python, Perl or TCL shared library modules providing they use simple data types.

GNU Modula-2 was based on a Modula-2 front end which performed a substantial amount of static analysis of the source code (see '`-Wpedantic`', '`-Wpedantic-param-names`', '`-Wstudents`' and '`-Wpedentic-cast`'.

Finally runtime checking has been implemented and can check to ensure that ranges of subranges are not exceeded, array indices are within range, functions execute a `RETURN` statement, a pointer does not dereference a `NIL` pointer value and that the expression within a `CASE` statement is correctly matched.

1.3 Release map

This section attempts to give an idea of which releases points are likely in the future. Clearly this is a fairly fluid release map but hopefully it is more helpful than omitting it altogether. Please note that this is not set in stone and if you wish to see something different please email `gaius@gnu.org` with your ideas. Also please note that the actual release numbers do not have any correlation to the estimated time of release. For example please do not misunderstand that GM2-1.0 will take twice as long as GM2-0.5 to appear. It is worthing noting that some of the later points in the release life have already been addressed (in part) but are not yet complete.

`0.50` compatible with gcc-3.3.6. GNU Modula-2 is stable and passes all regression tests on LP64 Opteron and 32 bit x86 GNU/Linux. The compiler is PIM-234 compatible (use '`-fpim2`', '`-fpim3`' and '`-fpim4`' to force mutually exclusive PIM features).

It is also able to compile the University of Ulm libraries which are now distributed as part of GNU Modula-2. To reference these libraries use the '`-flibs=ulm`' compiler switch.

`0.60` many Logitech compatible libraries will be provided, which will be available when invoked by '`-fpim`'.

`0.70` ISO exception handling working. Core exception handling ISO modules completed. Basic ISO IO libraries implemented. Multi-dimensional dynamic arrays complete. COMPLEX types implemented.

`0.80` a full set of ISO libraries will have been implemented. This release point marks the bug fixing of libraries and porting and fixing bugs.

`0.90` This release marks the porting and bug fixing and rapid releasing count down towards..

`1.0` GNU Modula-2 will be fully ISO compliant.

There will be releases inbetween those outlined above and these releases may occur when GNU Modula-2 builds using a different GCC source tree. It is a goal that backward compatibility to gcc-3.3.6 will be provided as far as it is possible. Releases will also occur if a key component of ISO Modula-2 is implemented (for example exception handling, complex types or '`FINALLY`' is implemented).

1.4 Compiler options

This section describes the compiler options specific to GNU Modula-2 for generic flags details see See ⟨undefined⟩ [Invoking GCC], page ⟨undefined⟩.

`-d` a GCC option See ⟨undefined⟩ [Invoking GCC], page ⟨undefined⟩. It allows users to specify how the rtl passes should dump their internal state.

`-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.

`-fobject-path=`
used to specify the path for objects during the linking stage. An example is: `gm2 -g -fobject-path=.:../../libs/O2 -I.:../../libs foo.mod`. The combination of `-I` and `-fobject-path=` allows projects to keep various styles of objects separate from their source counterparts. For example it would be possible to compile implementation modules with different levels of optimization and with/without debugging and keep them in separate directories. If the `-fobject-path=` option is not specified then it is set internally by using the path as specified by the `-I` option. If the `-I` was also not specified then it uses the current directory. In all cases the appropriate language dialect library directories are appended to the end of the path.

`-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.

`-fmakeinit`
generate the start up C++ code for the module, a file '`_m2_modulename.cpp`' is created. This is an internal command line option.

`-fmakeall`
generate a temporary makefile and build all dependent modules and link.

`-fclean` removes all objects before building the application. This option should be used with `-fmakeall`.

`-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 dependancy. 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 initialisation 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 -fcase`.

`-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.

`-fstatistics`
: generates quadruple information: number of quadruples generated, number of quadruples remaining after optimisation.

`-fmakelist`
: this option is only applicable when linking a program module. The compiler will generate a '`modulename.lst`' file which contains a list indicating the initialisation order of all modules which are to be linked. The actual link does not occur. The GNU Modula-2 linker scans all `IMPORT`s, generates a list of dependencies and produces an ordered list for initialisation. It will probably get the order wrong if your project has cyclic dependencies, but the '`.lst`' file is plain text and can be modified if required. Once the '`.lst`' file is created it can be used by the compiler to link your project via the '`-fuselist`' option. It has no effect if the '`-c`' option is present.

`-fuselist`
: providing '`gm2`' has been told to link the program module this option uses the file '`modulename.lst`' for the initialisation order of modules.

`-fcpp`
: preprocess the source with '`cpp -lang-asm -traditional-cpp`' For further details about these options see See ⟨undefined⟩ [Invocation], page ⟨undefined⟩. 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 See Section 1.9 [Dialect], page 15.

`-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 See Section 1.9 [Dialect], page 15.

`-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 See Section 1.9 [Dialect], page 15.

`-fpim3`
: turn on PIM-3 standard features. Currently this only effects the behaviour of `DIV` and `MOD` operators. See See Section 1.9 [Dialect], page 15.

`-fpim4`
: turn on PIM-4 standard features. Currently this only effects the behaviour of `DIV` and `MOD` operators. See See Section 1.9 [Dialect], page 15.

`-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 See Section 1.9 [Dialect], page 15.

`-flibs=ulm`
: modifies the default library search path so that the University of Ulm libraries are searched before the other PIM libraries.

`-flibs=pim`
: modifies the default library search path so that the PIM libraries are searched before any others (the default).

`-flibs=min`
: modifies the default library search path so that the absolute minimum of Modula-2 runtime libraries are loaded (a stripped down M2RTS, SYSTEM and libc).

`-flibs=iso`
: modifies the default library search path so that the ISO libraries are searched before any others (not needed if '`-fiso`' was specified).

`-flibs=logitech`
: modifies the default library search path so that the Logitech compatible libraries are searched before the base PIM libraries.

`-flibs=pim-coroutine`
: modifies the default libraries search path so that the PIM `SYSTEM` module providing coroutine support is searched before the base PIM libraries. This directory also includes many coroutine related libraries.

`-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.

`-fmodules`
: displays the path to each modules object file. This option is used at link time to check the correct object file is being linked.

`-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. Both this option and `-fdef=` also work with the `-fmakeall` option.

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

`-fonlylink`
: only link the modula-2 application, do not compile the program module beforehand.

`-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.

`-Wstudents`
checks for bad 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, but it can be extremely useful, as long as code is intentionally written in this way.

`-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.

1.5 Example compile and link

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 Section 1.4 [Compiler options], page 3. 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 should be able to compile and link it by: '`gm2 -g hello.mod`'. The result should 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 hello.mod`' and the link via: '`gm2 -g -fonlylink hello.mod`'.

To compile larger projects consisting of many modules you might find the option `-fmakeall` useful. For example to compile this tiny example using this option you can use the following command line:

'`gm2 -g -I. -fmakeall hello.mod`'

which will read the file '`hello.mod`' work out the dependancies and proceed to compile any dependant module whose source is in the directory determined by '`-I.`'.

[1]

1.6 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:../project/unix foo.mod
$ gm2 -fonlylink -g -I. -I../project:../project/unix \
  -fobject-path=../project/obj:../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.

Lastly there is the `GM2_ROOT` environment variable which determines where the compiler subcomponents reside in the filesystem. This environment variable overrides the compiler time configure option `--prefix=`. For example suppose the compiler was built to reside

[1] 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. -fmakeall hello.mod`'

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

in '/usr/local' and the system administrator decided to move the entire compiler tree to '/architecture/i386/usr'. Once the tree is moved then a system wide environment variable (GM2_ROOT) could be set to:

```
$ export GM2_ROOT=/architecture/i386/usr
```

The system administrator needs to ensure that the front end binary 'gm2' can be seen by the users path. At that point a user can invoke gm2 -g -c -I. hello.mod from the command line and all subcomponents will be picked up from '/architecture/i386/usr'. This allows users to try out different GNU Modula-2 releases and also allows system administrators to install compiler binaries at different locations to where they were initially configured to reside.

The environment variable GM2_ROOT has no effect if either the LIBRARY_PATH or COMPILE_PATH is set. The last two environment variables are used by gcc. However if by mistake GM2_ROOT and either LIBRARY_PATH or COMPILE_PATH is set then an error message is issued.

1.7 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
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 Section 1.21 [The PIM system module], page 39. See Section 1.22 [The ISO system module], page 43.

1.8 Permanently accessible base procedures.

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

1.8.1 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.
*)

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.
*)

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).
*)

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


(*
   DISPOSE - the procedure DISPOSE is replaced by:
             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.
*)

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.
*)

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

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

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.
*)

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

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

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

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

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

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

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
*)

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.
*)

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

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

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.
*)

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.
*)
```

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

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

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


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

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^.
*)

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

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

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


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

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.
*)

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.
*)

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.
*)

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.
*)

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.
*)

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

1.8.2 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.
*)

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).
*)

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


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

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.
*)

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.
*)

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

1.9 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 agressively 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.

1.9.1 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
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.

1.10 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) 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`'.

1.11 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 Section 1.4 [Compiler options], page 3.

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 "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 column of the first token on the line.

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.

1.11.1 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`.

```
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 Section 1.21 [The PIM system module], page 39. See Section 1.22 [The ISO system module], page 43.

1.12 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
SHORTINT                  pervasive
CARDINAL                  pervasive
LONGCARD                  pervasive
SHORTCARD                 pervasive
BOOLEAN                   pervasive
REAL                      pervasive
LONGREAL                  pervasive
```

```
SHORTREAL                pervasive
CHAR                     pervasive
SHORTCOMPLEX             pervasive
COMPLEX                  pervasive
LONGCOMPLEX              pervasive

BITSET                   SYSTEM
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 compiler categorises compatibility between all these types into three components: assignment, parameter and expression.

1.12.1 Assignment compatibility

This section discusses the assignment issues surrounding assignment compatibility of fundamental types. Obviously compatibility exists between the same sized 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.

1.12.2 Expression compatibility

According to the various Modula-2 standards `INTEGER` and `CARDINAL` types are not expression compatible (`http://freepages.modula2.org/report4/modula-2.html` and ISO Modula-2). This is rule is also extended across the fixed sized data types.

1.12.3 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` and `WORD` and sized their derivitives are assignment and parameter compatible with any data type of the same size.

1.13 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 `trees`, 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 `trees`, 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 implementation 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 `structure` 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.

1.14 Building a shared library

This section describes building a tiny shared library implemented entirely in Modula-2. 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:

```
gm2 -fiso -g -c -fPIC a.mod
gm2 -fiso -g -c -fPIC b.mod
gm2 -fiso -g -c -fPIC c.mod
```

The second step is to perform the link. In a simple project such as this you can use the single command:

```
gm2 -fiso -shared -fshared -fPIC -g c.mod -o c.so
```

At this point the shared library '`c.so`' will have been created. All procedures which are exported from the definition modules are also visible to the user of the shared library.

The `BEGIN` `END` code at module scope in '`a.mod`', '`b.mod`', '`c.mod`' and all dependent library modules will be initialized when the shared library is loaded. Likewise the `FINISH` `END`

code at all module scopes is executed when the library is unloaded. Both the initialization and termination is implemented transparently using the GNU/Linux mechanism:

```
void __attribute__ ((constructor)) initcode (void);
void __attribute__ ((destructor)) fincode (void);
```

The linking command above utilizes a number of subcomponents which determine the runtime initialization order, create the C++ initialization and finalization function and links all required modules together. You can inspect and modify the initialization and finalization order by using the following commands:

```
gm2 -c -fmakelist -fiso -fPIC -g c.mod
```

The command above will create a file '`c.lst`' which is a plain ascii file which contains the modules in initialization order. The `#` is a comment which continues to the next newline. It is worth noting that the first five or so modules which are listed before the commented lines are the critical runtime modules which have a default order. You can override these using the command line option `-fruntime-modules=`. However you should only do this if you are building an application for an embedded system or your own set of Modula-2 low level libraries.

You can modify the order of the modules in this file, although it is wise to leave the initial five modules alone. It is much more useful to order your own modules which will be near the end of the file. Once you are happy with the order you can conclude the link with the command:

```
gm2 -fonlylink -fuselist -fiso -fPIC -g c.mod -o c.so
```

which creates '`c.so`' and '`c_m2.cpp`'. The initialization and finalization code is contained within '`c_m2.cpp`' which is compiled and linked with all the modules required to satisfy all the program module '`c.mod`' dependencies.

1.15 How to produce swig interface files

This section describes how your 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 your implementation module.

Here is an example of how you might call upon the services of the Modula-2 library module `NumberIO` from Python. This example can be found in the directory '`gm2/examples/swig/full-strlib`' and can be run using the commands:

```
$ cd gcc-4.1.2/gcc/gm2/examples/swig/full-strlib
$ make numberio
```

If you wanted to do this step by step without the '`Makefile`' then firstly you should compile the `NumberIO` module as a shared library. This can be achieved by using the following commands:

```
$ cd gcc-4.1.2/gcc/gm2/examples/swig/full-strlib
$ gm2 -fshared -I. -c -fPIC -g -fswig -I../../../gm2-libs \
    ../../../gm2-libs/NumberIO.mod
```

The example assumes that the source code for '`NumberIO.mod`' can be found in directory '`../../../gm2-libs`'. The first command produces two files: '`NumberIO.i`' and

'NumberIO.o'. The file 'NumberIO.o' is a position independant code object file whereas the file 'NumberIO.i' is a swig interface file and contains a swig interpretation of the 'NumberIO.def'. GNU Modula-2 uses the same mechanism for handling exceptions as GNU C++. The file 'NumberIO.i' contains exception handling information therefore we need to ask swig to generate C++ wrappers for 'NumberIO.mod'. This is achieved by:

```
$ swig -c++ -python NumberIO.i
$ gcc -c -fPIC NumberIO_wrap.cxx -I/usr/include/python2.4
```

The swig command line generates the necessary Python and C++ interface files using the interface file. The C++ interface file is also compiled into position independant code. Finally the module NumberIO is linked with all its dependants and 'NumberIO_wrap.o'.

```
$ gm2 -fonlylink -shared -fshared -fPIC -g \
     ../../../gm2-libs/NumberIO.mod NumberIO_wrap.o -o _NumberIO.so
```

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:

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

1.15.1 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.

1.16 How to produce a Python module

This section descibes how it is possible to produce a Python module from your 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, maybe users of your code are familiar with Python and can use it to configure your application.

The examples given here can be found in the source tree '`gcc-`*`version`*`/gcc/gm2/examples/gravity/`'. The example used here is a pedagogical two dimensional gravity next event simulation. The Python module needs to have a clear API, this also needs to be placed in a single definition module. In the gravity simulation example this is found in the file: '`gcc-`*`version`*`/gcc/gm2/examples/gravity/twoDsim.def`'. Furthermore the API should only use fundamental pervasive data types and strings. Below is '`twoDsim.def`':

```
DEFINITION MODULE twoDsim ;

EXPORT UNQUALIFIED gravity, box, poly3, poly5, poly6, mass,
                   fix, circle, pivot, velocity, accel, fps,
                   replayRate, simulateFor, addDebugging ;
(*
   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.
*)

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 ;


(*
   pivot - pivot an object at position, (x0,y0).
*)

PROCEDURE pivot (x0, y0: REAL; id1: CARDINAL) : 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) ;


(*
   addDebugging - add a debugging event at time, t, which colours objects,
                  a, and, b, blue.
*)

PROCEDURE addDebugging (t: REAL; a, b: CARDINAL) ;


END twoDsim.
```

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

```
#!/usr/bin/python

from twoDsim import *

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 -fPIC -fiso -c deviceGnuPic.mod
$ gm2 -g -fPIC -fiso -c roots.mod
$ gm2 -g -fPIC -fiso -c -fswig twoDsim.mod
```

Notice that the last command both compiled and produced a swig interface file '`swig.i`'. We now use `swig` and `gcc` to produce and compile the interface wrappers:

```
$ swig -c++ -python twoDsim.i
$ gcc -c -fPIC twoDsim_wrap.cxx -I/usr/include/python2.5
```

Finally the application is linked into a shared library:

```
$ gm2 -fonlylink -fiso -fPIC -fshared -shared \
      twoDsim.mod twoDsim_wrap.o -o _twoDsim.so
```

The library name must start with a `_` to comply with the Python naming scheme.

1.17 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 which can be found in the source tree '`gcc-version/gcc/gm2/examples/callingC/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 the `[ 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 -fmakeall -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
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)`.

1.18 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.

```
PROCEDURE Example (foo, bar: CARDINAL) : CARDINAL ;
VAR
   myout: CARDINAL ;
BEGIN
   ASM VOLATILE ("movl %1,%%eax; addl %2,%%eax; movl %%eax,%0"
      : "=g" (myout)           (* outputs *)
      : "g" (foo), "g" (bar)   (* inputs  *)
      : "eax") ;               (* we trash *)
   RETURN( myout )
END Example ;
```

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

See ⟨undefined⟩ [Extensions to the C Language Family], page ⟨undefined⟩.

1.19 Data type alignment

GNU Modula-2 allows you to specify alignment for types and variables. The syntax is similar to GCC and uses the `__ATTRIBUTE__` mechanism. The ebnf of the alignment production is:

```
Alignment := [ "__ATTRIBUTE__" "(" "(" ALIGNED "(" 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 __ATTRIBUTE__ ((ALIGNED(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 __ATTRIBUTE__ ((ALIGNED(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
   x  : CHAR ;
   z  : POINTER TO INTEGER __ATTRIBUTE__ ((ALIGNED(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  __ATTRIBUTE__ ((ALIGNED(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 __ATTRIBUTE__ ((ALIGNED(1024))) ;

VAR
   x  : CHAR ;
   z  : foo ;
BEGIN
   IF ADR(z) MOD 1024=0
   THEN
      exit(0)
   ELSE
      exit(1)
   END
END align6.
```

1.20 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 ;

(*
    Description: provides a convenient place to list all the GNU Modula-2
                 built-in functions. These functions should be copied into
                 more generic modules.

                 For example the mathematical functions can be applied to
                 gm2-iso/LongMath. But each built-in function is here for
                 reference.
*)

FROM SYSTEM IMPORT ADDRESS ;

(* floating point intrinsic procedure functions *)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
PROCEDURE __BUILTIN__ scalbl (x, n: LONGREAL) : LONGREAL ;

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

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

(* complex arithmetic intrincic procedure functions *)

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

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

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

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

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

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

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

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

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

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

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

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

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

(* memory and string intrincic procedure functions *)

PROCEDURE __BUILTIN__ alloca (i: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ memcpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ index (s: ADDRESS; c: INTEGER) : ADDRESS ;
PROCEDURE __BUILTIN__ rindex (s: ADDRESS; c: INTEGER) : ADDRESS ;
PROCEDURE __BUILTIN__ memcmp (s1, s2: ADDRESS; n: CARDINAL) : INTEGER ;
PROCEDURE __BUILTIN__ memset (s: ADDRESS; c: INTEGER; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ memmove (s1, s2: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ strcat (dest, src: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strncat (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ strcpy (dest, src: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strncpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ strcmp (s1, s2: ADDRESS) : INTEGER ;
PROCEDURE __BUILTIN__ strncmp (s1, s2: ADDRESS; n: CARDINAL) : INTEGER ;
PROCEDURE __BUILTIN__ strlen (s: ADDRESS) : INTEGER ;
PROCEDURE __BUILTIN__ strstr (haystack, needle: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strpbrk (s, accept: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strspn (s, accept: ADDRESS) : CARDINAL ;
PROCEDURE __BUILTIN__ strcspn (s, accept: ADDRESS) : CARDINAL ;
PROCEDURE __BUILTIN__ strchr (s: ADDRESS; c: INTEGER) : ADDRESS ;
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.  *)

PROCEDURE __BUILTIN__ longjmp (env: ADDRESS; val: INTEGER) ;
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.
*)

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.
*)

PROCEDURE __BUILTIN__ return_address (level: 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.

1.21 The PIM system module

```
DEFINITION MODULE SYSTEM ;

(*
   Description: Implements the SYSTEM dependent module
                in the Modula-2 compiler.
*)

EXPORT QUALIFIED BITSPERBYTE, BYTESPERWORD,
                 LOC, WORD, BYTE, ADDRESS, BITSET,
                 INTEGER8, INTEGER16, INTEGER32, INTEGER64,
                 CARDINAL8, CARDINAL16, CARDINAL32, CARDINAL64,
                 WORD16, WORD32, WORD64, BITSET8,
                 BITSET16, BITSET32, REAL32, REAL64,
```

```
                  REAL96, REAL128, COMPLEX32, COMPLEX64,
                  COMPLEX96, COMPLEX128,
                  ADR, TSIZE, ROTATE, SHIFT, THROW ;
                  (* SIZE is also exported if -fpim2 is used *)

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


(* all the following types are declared internally to gm2
TYPE
   LOC ;
   WORD ;
   BYTE ;
   ADDRESS ;
   BITSET ;
   INTEGER8 ;
   INTEGER16 ;
   INTEGER32 ;
   INTEGER64 ;
   CARDINAL8 ;
   CARDINAL16 ;
   CARDINAL32 ;
   CARDINAL64 ;
   WORD16 ;
   WORD32 ;
   WORD64 ;
   BITSET8 ;
   BITSET16 ;
   BITSET32 ;
   REAL32 ;
   REAL64 ;
   REAL96 ;
   REAL128 ;
   COMPLEX32 ;
   COMPLEX64 ;
   COMPLEX96 ;
   COMPLEX128 ;
*)


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

PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
```

```
  (* Returns the address of variable v. *)

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.
  *)

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

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.
  *)

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.
  *)

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.
  *)
*)

(* 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.
*)

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.
*)

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.
*)

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.
*)

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.
*)

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.
*)

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.

1.22 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, REAL96,
                    REAL128, COMPLEX32, COMPLEX64, COMPLEX96,
                    COMPLEX128,
                    ADDADR, SUBADR, DIFADR, MAKEADR, ADR, ROTATE,
                    SHIFT, CAST, TSIZE,

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

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

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

TYPE
   LOC ;
   ADDRESS ;
   BYTE ;
   WORD ;
   INTEGER8 ;
   INTEGER16 ;
   INTEGER32 ;
   INTEGER64 ;
   CARDINAL8 ;
   CARDINAL16 ;
   CARDINAL32 ;
   CARDINAL64 ;
   WORD16 ;
   WORD32 ;
   WORD64 ;
   BITSET8 ;
   BITSET16 ;
   BITSET32 ;
   REAL32 ;
   REAL64 ;
   REAL96 ;
```

```
   REAL128 ;
   COMPLEX32 ;
   COMPLEX64 ;
   COMPLEX96 ;
   COMPLEX128 ;

TYPE
  LOC; (* A system basic type. Values are the uninterpreted
          contents of the smallest addressable unit of storage *)
  ADDRESS = POINTER TO LOC;
  WORD = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

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

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

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

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

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.
  *)

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.
  *)

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

PROCEDURE ROTATE (val: <a packedset 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.
  *)

PROCEDURE SHIFT (val: <a packedset 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.
  *)

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.
  *)

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.
  *)
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.
  *)
*)
```

```
(* 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.
*)

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.
*)

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.
*)

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.
```

```
*)

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.
*)

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.
*)

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


END SYSTEM.
```

2 Obtaining GNU Modula-2.

2.1 Caveat

This code is stable on the x86_64 and x86_32 Debian GNU/Linux platforms. However it is less stable on other platforms. Please see the web site to see the current regression test results. Patches and development volunteers are always welcome! See [Contributing], page 67.

2.2 Getting GNU Modula-2

The easiest way to obtain GNU Modula-2 is to install i386, amd64 or ppc64 Debian GNU/Linux and then add these repository descriptions to your '`/etc/apt/sources.list`' file.

```
#
#  GNU Modula-2 repo
#

deb http://floppsie.comp.glam.ac.uk/debian/ lenny main
deb-src http://floppsie.comp.glam.ac.uk/debian/ lenny main
```

Now as root type:

```
$ apt-get update
$ apt-get install gm2-doc gm2
```

As a normal user you can obtain the source code via:

```
$ apt-get source gm2
```

If you are not running Debian GNU/Linux then you can either download the source tarball and build it manually or checkout the latest sources using CVS and combine them with the appropriate gcc tarball and then finally build it manually.

Combined GNU Modula-2 and patched GCC tarballs exist at `http://floppsie.comp.glam.ac.uk/dow`
Search for files which look like '`gm2+gcc-`*`version`*`.tar.gz`'. For example you should be able to download the latest version of GNU Modula-2 and GCC using these commands.

```
$ wget http://floppsie.comp.glam.ac.uk/download/c/gcc-4.1.2+gm2-cvs-latest.tar.gz
$ tar zxf gcc-4.1.2+gm2-cvs-latest.tar.gz
```

2.3 Building GNU Modula-2 from source under GNU/Linux

On a GNU/Linux system you should also be able to build it using these commands. The following commands assume your shell is '`/bin/bash`'. To build GNU Modula-2 type:

```
$ mkdir -p $HOME/opt
$ mkdir -p build-4.1.2
$ cd build-4.1.2
$ ../gcc-4.1.2+gm2-cvs-latest/configure --enable-languages=c,c++,gm2 \
     --disable-multilib --enable-checking=all --prefix=$HOME/opt
$ make "SHELL=/bin/bash"
```

To install GNU Modula-2, after a successful build, type:

```
$ make "SHELL=/bin/bash" install
$ cd ..
```

Now you should be able to perform:

```
$ export PATH=$HOME/opt/bin:$PATH
$ cd build-4.1.2/gcc/gm2/examples/hello
$ make post-install
```

which will create an '`a.out`' for the infamous hello world example.

2.4 Development sources via CVS

Development sources can be downloaded via CVS but they must be grafted carefully onto an existing GCC release. The notes in this section document how this is achieved, however a prepared CVS and GCC release is available here `http://floppsie.comp.glam.ac.uk/download/c/gcc-4.1.2+gm2-cvs-latest.tar.gz` and it is created upon each commit.

If you want to obtain the latest sources via CVS then type the following:

```
$ cvs -z3 -d:pserver:anoncvs@cvs.sv.gnu.org:/sources/gm2 co gm2
```

This will checkout a copy of GNU Modula-2 into one subdirectory, '`gm2`'. This version of GNU Modula-2 needs to be placed inside the GCC source tree in the position gcc-4.1.2/gcc before GNU Modula-2 can be built. Please check the GNU Modula-2 homepage `http://www.nongnu.org/gm2` for details about which GCC releases are supported by GNU Modula-2.

Once you have downloaded the correct GCC release from `http://gcc.gnu.org` or a mirror site you should unpack the GCC archive. Assuming that both the '`gcc-4.1.2`' and '`gm2`' directories are at the same level, you can graft '`gm2`' onto '`gcc-4.1.2`' by:

```
$ mv gm2 gcc-4.1.2/gcc
```

If the directory '`gcc-4.1.2/gcc/gm2/patches/gcc/4.1.2`' exists then the patch files inside that directory can be applied to the '`gcc-4.1.2`' tree. This is done via:

```
$ cd gcc-4.1.2
$ if [ -d gcc/gm2/patches/gcc/4.1.2 ] ; then
     for i in gcc/gm2/patches/gcc/4.1.2/* ; do
        if [ -f $i ] ; then
           patch -p1 < $i
        fi
     done
  fi
```

Note that if you download a tarball from `http://floppsie.comp.glam.ac.uk` then any patching will have already been applied. See Chapter 2 [Obtaining], page 49.

2.5 Stress testing GM2

Currently there are two automated methods to test GNU Modula-2. The first method is '`make gm2.paranoid`' in which `gm2` builds itself and finally the test runs both parent and child generations of the compiler and compares the output. Be warned that this test can take some time to execute. This test is invoked by:

```
$ cd host-build/gcc ; make gm2.paranoid
```

The second method used to test GNU Modula-2 is to run the regression test suite. The GNU Modula-2 regression test suite is available for download. To install and run the GNU Modula-2 regression suite you need to have installed the '`dejagnu`' and '`expect`' packages. Note that you need to ensure that you have at least the following releases of dejagnu components:

```
$ runtest --version

Expect version is 5.42.1
Tcl version is 8.4
Framework version is 1.4.4
```

otherwise some of the tests may not run.

If you have downloaded the combined GCC and GNU Modula-2 tarball `http://floppsie.comp.glam.ac.uk/download/c/gcc-4.1.2+gm2-cvs-latest.tar.gz` then this will also contain the GNU Modula-2 testsuite. In this case you can skip over the following '`cvs`' and '`tar`' commands.

However if you have downloaded GNU Modula-2 using CVS then you will also need to download and position the testsuite. Assuming that the root of the GCC source tree is in the current working directory you can use the following commands to install the test suite:

```
$ cvs -z3 -d:pserver:anoncvs@cvs.sv.gnu.org:/sources/gm2 co testsuite
$ tar cf - testsuite | ( cd gcc-version/gcc ; tar xf - )
```

Do not simply move the directory '`testsuite`' into '`gcc-`*`version`*`/gcc`' as the GNU Modula-2 regression tests have to be overlaid on top of the gcc testsuite.

Thereafter you can run the GNU Modula-2 testsuite by:

```
$ cd host-build/gcc
$ make check-gm2
```

Depending on the speed of your computer these tests may take a while to complete.

2.6 Building GNU Modula-2 under Cygwin

GNU Modula-2 now builds under Cygwin. The Cygwin version used was gcc version 3.4.4, gdc 0.12 using dmd 0.125. The following set of commands were used to build GNU Modula-2.

```
$ mkdir build
$ cd build
$ ../gcc-4.1.2+gm2-cvs-latest/configure --prefix=/gm2/opt \
  --disable-multilib --enable-checking=all \
  --enable-languages=c,c++,gm2
$ make
```

GNU Modula-2 can be installed by:

```
$ make install
```

You now need to modify your path to include '`gm2`'. This is done by:

```
$ export PATH=/gm2/opt/bin:$PATH
```

and now you can try out the `hello world` example:

```
$ cd ../gcc-4.1.2/gcc/gm2/examples/hello
$ make
$ ./a.exe
```

2.7 Moving the installation of GNU Modula-2 to another directory

This section documents how you can configure GNU Modula-2 to install in one directory and at a later time move the installation to another completely different directory. The example assumes the user has a bourne shell.

Let us assume that the build was configured as such:

```
$ mkdir build
$ cd build
$ ../gcc-4.1.2+gm2-cvs-latest/configure \
   --prefix=$HOME/private \
   --enable-languages=c,c++,gm2  --enable-checking \
   --disable-multilib
```

in this example we see that the installation directory would normally be '`$HOME/private`'. This is accomplished by the following commands:

```
$ make
$ make install
```

Now, assuming we have correct privileges, we may move the entire contents of '`$HOME/private`' to '`/usr/local/public`':

```
$ mv $HOME/private /usr/local/public
```

The new compiler and libraries can be referenced by modifying our `PATH` environment variable:

```
$ PATH=/usr/local/public/bin:$PATH
$ export PATH
$ GM2_ROOT=/usr/local/public
$ export GM2_ROOT
```

Finally we can rebuild the `hello world` example found in '`../gcc-4.1.2/gcc/gm2/examples/hello`' by:

```
$ cd ../gcc-4.1.2/gcc/gm2/examples/hello
$ make
$ ./a.out
```

2.8 Building GNU Modula-2 under Mac OSX

The easiest way to build GNU Modula-2 under Mac OSX is to use '`macports`'. At the time of writing GNU Modula-2 is not an official mac port, however you can build it locally and produce your own testing release quite easily.

Firstly you must install '`macports`' `http://www.macports.org/install.php`. Secondly you need to modify your '`macports`' configuration file '`/opt//local/etc/macports/sources.conf`' so that it allows users to maintain private ports. This requires that you add the line:

```
file:///Users/‘yourusername’/MacPorts/ports
```

which allows you to keep your own ‘macports’ under ‘$HOME/MacPorts/ports’. Now download the GNU Modula-2 Portfile and build the port index.

```
$ mkdir -p $HOME/MacPorts/ports/lang/gm2-latest
$ cd $HOME/MacPorts/ports/lang/gm2-latest
$ curl http://floppsie.comp.glam.ac.uk/download/c/patches/gcc/4.1.2/Portfile \
  -o Portfile
$ cd $HOME/MacPorts/port
$ portindex
Creating software index in /Users/‘yourusername’/MacPorts/ports
Adding port lang/gm2-latest

Total number of ports parsed:   1
Ports successfully parsed:      1
Ports failed:                   0
$ sudo port clean --all gm2-latest
--->  Cleaning gm2-latest
$ sudo port build gm2-latest
Password:
--->  Fetching gm2-latest
--->  Attempting to fetch gcc-core-4.1.2.tar.bz2 from ftp://ftp.lip6.fr/pub/gnu//gc
4.1.2
--->  Attempting to fetch gcc-g++-4.1.2.tar.bz2 from ftp://ftp.lip6.fr/pub/gnu//gcc
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.lip6.fr/pub/gnu//gc
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://www.mirrorservice.org/
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.gwdg.de/pub/linux/g
4.1.2/
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.nluug.nl/mirror/lan
4.1.2/
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://arn.se.distfiles.macpo
latest
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.funet.fi/pub/mirror
4.1.2/
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.funet.fi/pub/gnu/pr
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://mirrors.kernel.org/gnu
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.chg.ru/pub/gnu//gcc
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://mirrors.ibiblio.org/pu
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://distfiles.macports.org
latest
```

```
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://gcc.gnu.org/pub/gcc/rel
4.1.2/
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.dti.ad.jp/pub/GNU//
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from ftp://ftp.unicamp.br/pub/gnu/
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://mirror.aarnet.edu.au/p
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://aarnet.au.distfiles.ma
latest
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://mirror.pacific.net.au/
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://mirror.internode.on.ne
4.1.2
--->  Attempting to fetch gm2-cvs-latest.tar.bz2 from http://floppsie.comp.glam.ac.
--->  Verifying checksum(s) for gm2-latest
--->  Extracting gm2-latest
--->  Applying patches to gm2-latest
--->  Configuring gm2-latest
--->  Building gm2-latest
```

Now you can produce your own release of GNU Modula-2 via:

```
$ sudo port dmg gm2-latest
--->  Staging gm2-latest into destroot
--->  Creating pkg for gm2-latest-4.1.2
--->  Creating disk image for gm2-latest-4.1.2
```

You can run the regression testsuite by:

```
$ sudo port install dejagnu
$ cd /opt/local/var/macports/build/_opt_local_var_macports_sources_rsync.macports.o
$ sudo make check-gm2
$ sudo make gm2.paranoid
```

Here is a simple script which enables you to download, build and test the latest GNU Modula-2 cvs snapshot:

```
#!/bin/bash

cd $HOME/MacPorts/ports/lang/gm2-latest
curl http://floppsie.comp.glam.ac.uk/download/c/patches/gcc/4.1.2/Portfile -
o Portfile
cd $HOME/MacPorts/ports
portindex
cd $HOME
sudo port clean --all gm2-latest
sudo port build gm2-latest
cd /opt/local/var/macports/build/_opt_local_var_macports_sources_rsync.macports.org
sudo make gm2.paranoid
sudo make check-gm2
```

2.9 Building GNU Modula-2 under FreeBSD

[This section is out of date and has not been verified against gcc-4.1.2 and GNU Modula-2 release gm2-1.0].

When building GNU Modula-2 under FreeBSD, there are essentially three issues that need to be addressed.

The first, is the system shell, '`/bin/sh`'. GNU Modula-2's build script uses some '`bash`' constructs that are not understood by '`sh`'. Therefore '`bash`' must be installed and this can be obtained from the ports package collection (ports:shells/bash).

The second, is the compiler used to bootstrap GNU Modula-2. On FreeBSD4.x the system compiler is from the 2.95.x generation, and should work without problems. On 5-RELEASE and 6-CURRENT, the system compiler is from the 3.4.x generation or greater, and it is known to create a faulty GNU Modula-2 compiler. Therefore you will need to install an earlier gcc on your machine. Known to work are gcc 3.2.3 (ports:lang/gcc32), gcc 3.3.4, 3.3.5 and 3.3.6 (ports:lang/gcc33).

Finally, '`gmake`' is required.

It is recommended that the same options are used to configure GNU Modula-2 as those suggested in the ports collection. A number of options are not relevant for building GNU Modula-2 and these can be safely omitted. The only two which apply directly to GNU Modula-2's build process are `--with-system-zlib` and `--disable-nls`.

The example below assumes that `gcc-3.2.3` is installed (from the ports collection) and the compiler is called '`gcc32`'. The example assumes that the bash shell has been installed (as described above).

```
mkdir host-build
cd host-build
env CONFIG_SHELL=/usr/local/bin/bash CC=gcc32 ../gcc-version/configure
    --with-system-zlib --disable-nls --enable-languages=c,c++,gm2
gmake
```

If you choose to install the generated compiler, you are urged to make use of the name rewriting options of configure (`--program-prefix` and `--program-suffix` work fine), and to avoid possible conflicts with a port that installs it's own gcc, you may also want to add `--host`. Here is the author of this section's full configure line:

```
env CONFIG_SHELL=/usr/local/bin/bash CC=gcc32 ../gcc-version/configure
    --with-system-zlib --disable-nls --enable-languages=c,c++,gm2
    --program-prefix=m2 --host=i386-gm2bld-freebsd5.3
```

2.10 Licence of GNU Modula-2

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an

appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
   b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange,

for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any

third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license

to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

The GNU Project and GNU/Linux

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. (GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced "guh-NEW".) Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as "Linux", they are more accurately called GNU/Linux systems.

For more information, see:

```
http://www.gnu.org/
http://www.gnu.org/gnu/linux-and-gnu.html
```

Contributing to GNU Modula-2

Please do. But also please read the GNU Emacs info under

```
* Standards: (standards).       GNU coding standards.
* Intellectual Property::       Keeping Free Software Free
* Reading Non-Free Code::       Referring to Proprietary Programs
* Contributions::               Accepting Contributions
```

You might consider joining the GM2 Mailing list available via a web brouser at `http://lists.nongnu.org/mailman/listinfo/gm2` available via email: `mail:gm2-subscribe@nongnu.org` before you start coding.

Many thanks and enjoy your coding!

This section is still being written.

3 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.

```
PossiblyExportIdent := is a builtin which automatically exports an iden-
tifier
                     =:
Ident := is a builtin and checks for an identifier
       =:
IdentScope := a builtin which provides a context for error messages
            =:
Integer := is a builtin and checks for an integer
         =:
Real := is a builtin and checks for an real constant
      =:
string := is a builtin and checks for an string constant
        =:
FileUnit := ( DefinitionModule  | ImplementationOrProgramModule  )

          =:
ProgramModule := 'MODULE' Ident [ Priority  ] ';' { Import  }
                 Block Ident '.'
               =:
ImplementationModule := 'IMPLEMENTATION' 'MODULE' Ident [ Priority  ]
                        ';' { Import  } Block Ident '.'
                      =:
ImplementationOrProgramModule := ImplementationModule  |
                                 ProgramModule
                               =:
Number := Integer  | Real
        =:
Qualident := Ident { '.' Ident  }
           =:
ConstantDeclaration := PossiblyExportIdent '=' ConstExpression
                     =:
ConstExpression := SimpleConstExpr [ Relation SimpleConstExpr  ]

                 =:
Relation := '='  | '#'  | '<>'  | '<'  | '<='  | '>'  | '>='  |
            'IN'
          =:
```

```
SimpleConstExpr := UnaryOrConstTerm { AddOperator ConstTerm  }
                   =:
UnaryOrConstTerm := '+' ConstTerm  | '-' ConstTerm  |
                      ConstTerm
                    =:
AddOperator := '+'  | '-'  | 'OR'
               =:
ConstTerm := ConstFactor { MulOperator ConstFactor  }
             =:
MulOperator := '*'  | '/'  | 'DIV'  | 'MOD'  | 'REM'  |
                 'AND'  | '&'
               =:
ConstFactor := Number  | ConstString  | ConstSetOrQualidentOrFunction  |
                 '(' ConstExpression ')'  |
                 'NOT' ConstFactor  | ConstAttribute
               =:
ConstString := string
               =:
ComponentElement := ConstExpression [ '..' ConstExpression  ]
                    =:
ComponentValue := ComponentElement [ 'BY' ConstExpression  ]
                  =:
ArraySetRecordValue := ComponentValue { ',' ComponentValue  }
                       =:
Constructor := '{' [ ArraySetRecordValue  ] '}'
               =:
ConstSetOrQualidentOrFunction := Constructor  | Qualident [ Constructor  |
                                                            ConstActualParameters
                                 =:
ConstActualParameters := '(' [ ExpList  ] ')'
                         =:
ConstAttribute := '__ATTRIBUTE__' '__BUILTIN__' '(' '(' ConstAttributeExpression
                    ')' ')'
                  =:
ConstAttributeExpression := Ident  | '<' Qualident ',' Ident
                              '>'
                            =:
Element := ConstExpression [ '..' ConstExpression  ]
           =:
Alignment := [ '__ATTRIBUTE__' '(' '(' Ident '(' ConstExpression
                 ')' ')' ')'  ]
             =:
```

```
TypeDeclaration := ( IdentScope '=' Type Alignment  )
                   =:
Type := ( SimpleType  | ArrayType  | RecordType  |
          SetType  | PointerType  | ProcedureType  )
        =:
SimpleType := Qualident [ SubrangeType  ]  |
              Enumeration  | SubrangeType
              =:
Enumeration := '(' ( PossiblyExportIdentList  ) ')'
               =:
IdentList := Ident { ',' Ident  }
             =:
IdentScopeList := IdentScope { ',' IdentScope  }
                  =:
PossiblyExportIdentList := PossiblyExportIdent { ',' PossiblyExportIdent
                                                 }
                           =:
SubrangeType := '[' ConstExpression '..' ConstExpression ']'
                =:
ArrayType := 'ARRAY' SimpleType { ',' SimpleType  } 'OF' Type
             =:
RecordType := 'RECORD' FieldListSequence 'END'
              =:
FieldListSequence := FieldListStatement { ';' FieldListStatement  }
                     =:
FieldListStatement := [ FieldList  ]
                      =:
FieldList := IdentList ':' Type Alignment  |
             'CASE' CaseTag 'OF' Varient { '|' Varient  } [
   'ELSE' FieldListSequence  ] 'END'
            =:
TagIdent := [ Ident  ]
            =:
CaseTag := TagIdent [ ':' Qualident  ]
           =:
Varient := [ VarientCaseLabelList ':' FieldListSequence  ]
           =:
VarientCaseLabelList := VarientCaseLabels { ',' VarientCaseLabels  }
                        =:
VarientCaseLabels := ConstExpression [ '..' ConstExpression  ]
                     =:
CaseLabelList := CaseLabels { ',' CaseLabels  }
                 =:
```

```
CaseLabels := ConstExpression [ '..' ConstExpression  ]
           =:
SetType := ( 'SET'  | 'PACKEDSET'  ) 'OF' SimpleType
         =:
PointerType := 'POINTER' 'TO' Type
             =:
ProcedureType := 'PROCEDURE' [ FormalTypeList  ]
               =:
FormalTypeList := '(' ( ')' FormalReturn  |
                        ProcedureParameters ')' FormalReturn  )
                =:
FormalReturn := [ ':' OptReturnType  ]
              =:
OptReturnType := '[' Qualident ']'  | Qualident
               =:
ProcedureParameters := ProcedureParameter { ',' ProcedureParameter  }
                     =:
ProcedureParameter := '...'  | 'VAR' FormalType  |
                      FormalType
                    =:
VarIdent := PossiblyExportIdent [ '[' ConstExpression ']'  ]
          =:
VariableDeclaration := ( VarIdentList ':' Type Alignment  )
                     =:
VarIdentList := VarIdent { ',' VarIdent  }
              =:
Designator := Qualident { SubDesignator  }
            =:
SubDesignator := '.' Ident  | '[' ExpList ']'  |
                 '^'
               =:
ExpList := Expression { ',' Expression  }
         =:
Expression := SimpleExpression [ Relation SimpleExpression  ]

            =:
SimpleExpression := [ '+'  | '-'  ] Term { AddOperator Term  }
                  =:
Term := Factor { MulOperator Factor  }
      =:
Factor := Number  | string  | SetOrDesignatorOrFunction  |
          '(' Expression ')'  | 'NOT' Factor  |
          ConstAttribute
```

```
        =:
SetOrDesignatorOrFunction := ( Qualident [ Constructor  |
                                           SimpleDes [ ActualParameters  ]  ]  |
                               Constructor  )
                             =:
SimpleDes := { '.' Ident  | '[' ExpList ']'  |
                '^'  }
             =:
ActualParameters := '(' [ ExpList  ] ')'
                    =:
Statement := [ AssignmentOrProcedureCall  |
               IfStatement  | CaseStatement  |
               WhileStatement  | RepeatStatement  |
               LoopStatement  | ForStatement  |
               WithStatement  | AsmStatement  |
               'EXIT'  | 'RETURN' [ Expression  ]  |
               RetryStatement  ]
             =:
RetryStatement := 'RETRY'
                  =:
AssignmentOrProcedureCall := Designator ( ':=' Expression  |
                                          ActualParameters  |
                                          )
                             =:
StatementSequence := Statement { ';' Statement  }
                     =:
IfStatement := 'IF' Expression 'THEN' StatementSequence { 'ELSIF'
                                                          Expression
                                                          'THEN'
                                                          StatementSequence  }
               [ 'ELSE' StatementSequence  ] 'END'
             =:
CaseStatement := 'CASE' Expression 'OF' Case { '|' Case  }
                 [ 'ELSE' StatementSequence  ] 'END'
               =:
Case := [ CaseLabelList ':' StatementSequence  ]
      =:
WhileStatement := 'WHILE' Expression 'DO' StatementSequence 'END'
                  =:
RepeatStatement := 'REPEAT' StatementSequence 'UNTIL' Expression
                   =:
ForStatement := 'FOR' Ident ':=' Expression 'TO' Expression [
   'BY' ConstExpression  ] 'DO' StatementSequence 'END'
              =:
```

```
LoopStatement := 'LOOP' StatementSequence 'END'
                 =:
WithStatement := 'WITH' Designator 'DO' StatementSequence 'END'
                 =:
ProcedureDeclaration := ProcedureHeading ';' ( ProcedureBlock
                                               Ident  )
                        =:
DefineBuiltinProcedure := '__ATTRIBUTE__' '__BUILTIN__' '(' '('
                          Ident ')' ')'  |
                          '__INLINE__'  |
                         =:
ProcedureHeading := 'PROCEDURE' DefineBuiltinProcedure ( PossiblyExportIdent
                                                         [ FormalParameters  ]
                                                          )
                    =:
Builtin := '__BUILTIN__'  | '__INLINE__'  |
           =:
DefProcedureHeading := 'PROCEDURE' Builtin ( PossiblyExportIdent
                                             [ DefFormalParameters  ]
                                              )
                       =:
ProcedureBlock := { Declaration  } [ 'BEGIN' BlockBody  ] 'END'
                  =:
Block := { Declaration  } InitialBlock FinalBlock 'END'
         =:
InitialBlock := [ 'BEGIN' BlockBody  ]
                =:
FinalBlock := [ 'FINALLY' BlockBody  ]
              =:
BlockBody := NormalPart [ 'EXCEPT' ExceptionalPart  ]
             =:
NormalPart := StatementSequence
              =:
ExceptionalPart := StatementSequence
                   =:
Declaration := 'CONST' { ConstantDeclaration ';'  }  |
               'TYPE' { TypeDeclaration ';'  }  |
               'VAR' { VariableDeclaration ';'  }  |
               ProcedureDeclaration ';'  |
               ModuleDeclaration ';'
              =:
DefFormalParameters := '(' [ DefMultiFPSection  ] ')' FormalReturn
                       =:
```

```
DefMultiFPSection := DefExtendedFP  | FPSection [ ';' DefMultiFPSection  ]
                   =:
FormalParameters := '(' [ MultiFPSection  ] ')' FormalReturn
                  =:
MultiFPSection := ExtendedFP  | FPSection [ ';' MultiFPSection  ]
                =:
FPSection := NonVarFPSection  | VarFPSection
           =:
DefExtendedFP := DefOptArg  | '...'
               =:
ExtendedFP := OptArg  | '...'
            =:
VarFPSection := 'VAR' IdentScopeList ':' FormalType
              =:
NonVarFPSection := IdentScopeList ':' FormalType
                 =:
OptArg := '[' IdentScope ':' FormalType [ '=' ConstExpression  ]
          ']'
        =:
DefOptArg := '[' IdentScope ':' FormalType '=' ConstExpression
             ']'
            =:
FormalType := { 'ARRAY' 'OF'  } Qualident
            =:
ModuleDeclaration := 'MODULE' Ident [ Priority  ] ';' { Import
                                                      }
                     [ Export  ] Block Ident
                   =:
Priority := '[' ConstExpression ']'
          =:
Export := 'EXPORT' ( 'QUALIFIED' IdentList  |
                     'UNQUALIFIED' IdentList  |
                     IdentList  ) ';'
        =:
Import := 'FROM' Ident 'IMPORT' IdentList ';'  |
          'IMPORT' IdentList ';'
        =:
DefinitionModule := 'DEFINITION' 'MODULE' ( 'FOR' string  |
                                            ) Ident ';' { Import
                                                        }
                    [ Export  ] { Definition  } 'END' Ident
                    '.'
                  =:
```

```
Definition := 'CONST' { ConstantDeclaration ';'  }  |
              'TYPE' { PossiblyExportIdent ( ';'  | '=' Type
                                             Alignment ';'  )  }
               |
              'VAR' { VariableDeclaration ';'  }  |
              DefProcedureHeading ';'
            =:
AsmStatement := 'ASM' [ 'VOLATILE'  ] '(' AsmOperands ')'
              =:
AsmOperands := string [ ':' AsmList [ ':' AsmList [ ':' TrashList  ]  ]  ]
             =:
AsmList := [ AsmElement  ] { ',' AsmElement  }
         =:
AsmElement := string '(' Expression ')'
            =:
TrashList := [ string  ] { ',' string  }
           =:
```

4 PIM and ISO library definitions

This chapter contains M2F, ULM, PIM and ISO libraries. The ISO libraries are currently work in progress, many are complete but probably contain many bugs. The M2F libraries are very mature as the compiler uses them extensively. Permission has been kindly granted by the authors of the ULM libraries to include them with GNU Modula-2. These libraries (under the GNU GPL) were written at the University of Ulm and were originally shipped with the ULM sparc Modula-2 compiler.

4.1 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.

4.1.1 gm2-libs/StrIO

```
DEFINITION MODULE StrIO ;

(*
   Description: Provides simple string input output routines.
*)

EXPORT QUALIFIED ReadString, WriteString,
                 WriteLn ;


(*
   WriteLn - writes a carriage return and a newline
             character.
*)

PROCEDURE WriteLn ;


(*
   ReadString - reads a sequence of characters into a string.
                Line editing accepts Del, Ctrl H, Ctrl W and
                Ctrl U.
*)

PROCEDURE ReadString (VAR a: ARRAY OF CHAR) ;
```

```
(*
   WriteString - writes a string to the default output.
*)

PROCEDURE WriteString (a: ARRAY OF CHAR) ;


END StrIO.
```

4.1.2 gm2-libs/StrCase

```
DEFINITION MODULE StrCase ;


EXPORT QUALIFIED StrToUpperCase, StrToLowerCase, Cap, Lower ;


(*
   StrToUpperCase - converts string, a, to uppercase returning the
                    result in, b.
*)

PROCEDURE StrToUpperCase (a: ARRAY OF CHAR ; VAR b: ARRAY OF CHAR) ;


(*
   StrToLowerCase - converts string, a, to lowercase returning the
                    result in, b.
*)

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.
*)

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.
*)

PROCEDURE Lower (ch: CHAR) : CHAR ;


END StrCase.
```

4.1.3 gm2-libs/Builtins

```
DEFINITION MODULE Builtins ;

(*
    Description: provides a convenient place to list all the GNU Modula-2
                 built-in functions. These functions should be copied into
                 more generic modules.

                 For example the mathematical functions can be applied to
                 gm2-iso/LongMath. But each built-in function is here for
                 reference.
*)

FROM SYSTEM IMPORT ADDRESS ;

(* floating point intrinsic procedure functions *)

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

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

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

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

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

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

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

PROCEDURE __BUILTIN__ log10f (x: SHORTREAL) : SHORTREAL ;
```

```
PROCEDURE __BUILTIN__ log10 (x: REAL) : REAL ;
PROCEDURE __BUILTIN__ log10l (x: LONGREAL) : LONGREAL ;

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

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

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

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

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

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

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

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

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

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

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

```
(* complex arithmetic intrincic procedure functions *)

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

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

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

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

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

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

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

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

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

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

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

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

```
PROCEDURE __BUILTIN__ carccos (z: COMPLEX) : COMPLEX ;
PROCEDURE __BUILTIN__ carccosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

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

(* memory and string intrincic procedure functions *)

PROCEDURE __BUILTIN__ alloca (i: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ memcpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ index (s: ADDRESS; c: INTEGER) : ADDRESS ;
PROCEDURE __BUILTIN__ rindex (s: ADDRESS; c: INTEGER) : ADDRESS ;
PROCEDURE __BUILTIN__ memcmp (s1, s2: ADDRESS; n: CARDINAL) : INTEGER ;
PROCEDURE __BUILTIN__ memset (s: ADDRESS; c: INTEGER; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ memmove (s1, s2: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ strcat (dest, src: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strncat (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ strcpy (dest, src: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strncpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE __BUILTIN__ strcmp (s1, s2: ADDRESS) : INTEGER ;
PROCEDURE __BUILTIN__ strncmp (s1, s2: ADDRESS; n: CARDINAL) : INTEGER ;
PROCEDURE __BUILTIN__ strlen (s: ADDRESS) : INTEGER ;
PROCEDURE __BUILTIN__ strstr (haystack, needle: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strpbrk (s, accept: ADDRESS) : ADDRESS ;
PROCEDURE __BUILTIN__ strspn (s, accept: ADDRESS) : CARDINAL ;
PROCEDURE __BUILTIN__ strcspn (s, accept: ADDRESS) : CARDINAL ;
PROCEDURE __BUILTIN__ strchr (s: ADDRESS; c: INTEGER) : ADDRESS ;
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.  *)

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

```
*)

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.
*)

PROCEDURE __BUILTIN__ return_address (level: CARDINAL) : ADDRESS ;


END Builtins.
```

4.1.4 gm2-libs/Storage

```
DEFINITION MODULE Storage ;

(*
   Description: Implements the dynamic Storage handler for the
                Modula-2 compiler.
*)

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.
*)

PROCEDURE ALLOCATE (VAR a: ADDRESS ; Size: CARDINAL) ;


(*
   DEALLOCATE - return, Size, bytes to the heap.
                The variable, a, is set to NIL.
*)

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.
*)

PROCEDURE REALLOCATE (VAR a: ADDRESS; Size: CARDINAL) ;


(*
   Available - returns TRUE if, Size, bytes can be allocated.
*)

PROCEDURE Available (Size: CARDINAL) : BOOLEAN ;
```

```
END Storage.
```

4.1.5 gm2-libs/Environment

```
DEFINITION MODULE Environment ;

(*
    Description: provides access to the environment settings of a process.
*)

EXPORT QUALIFIED GetEnvironment ;


(*
   GetEnvironment - gets the environment variable, Env, and places
                    a copy of its value into string, a.
*)

PROCEDURE GetEnvironment (Env: ARRAY OF CHAR; VAR a: ARRAY OF CHAR) : BOOLEAN ;


END Environment.
```

4.1.6 gm2-libs/StrLib

```
DEFINITION MODULE StrLib ;

(*
   Description: Provides string manipulation
*)

EXPORT QUALIFIED StrConCat, StrLen, StrCopy, StrEqual, StrLess,
                 IsSubString, StrRemoveWhitePrefix ;


(*
   StrConCat - combines a and b into c.
*)

PROCEDURE StrConCat (a, b: ARRAY OF CHAR; VAR c: ARRAY OF CHAR) ;


(*
   StrLess - returns TRUE if string, a, alphabetically occurs before
             string, b.
*)

PROCEDURE StrLess (a, b: ARRAY OF CHAR) : BOOLEAN ;


(*
   StrEqual - performs a = b on two strings.
*)

PROCEDURE StrEqual (a, b: ARRAY OF CHAR) : BOOLEAN ;


(*
   StrLen - returns the length of string, a.
*)

PROCEDURE StrLen (a: ARRAY OF CHAR) : CARDINAL ;


(*
   StrCopy - effectively performs b := a with two strings.
*)

PROCEDURE StrCopy (a: ARRAY OF CHAR ; VAR b: ARRAY OF CHAR) ;
```

```
(*
   IsSubString - returns true if b is a subcomponent of a.
*)

PROCEDURE IsSubString (a, b: ARRAY OF CHAR) : BOOLEAN ;


(*
   StrRemoveWhitePrefix - copies string, into string, b, excluding any white
                          space infront of a.
*)

PROCEDURE StrRemoveWhitePrefix (a: ARRAY OF CHAR; VAR b: ARRAY OF CHAR) ;


END StrLib.
```

4.1.7 gm2-libs/MemUtils

```
DEFINITION MODULE MemUtils ;

(*
    Description: provides some basic memory utilities.
*)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED MemCopy, MemZero ;


(*
   MemCopy - copys a region of memory to the required destination.
*)

PROCEDURE MemCopy (from: ADDRESS; length: CARDINAL; to: ADDRESS) ;


(*
   MemZero - sets a region of memory: a..a+length to zero.
*)

PROCEDURE MemZero (a: ADDRESS; length: CARDINAL) ;


END MemUtils.
```

4.1.8 gm2-libs/LMathLib0

```
DEFINITION MODULE LMathLib0 ;

(*
    Description: provide access to the LONGREAL instrinics.
*)

CONST
   pi   = 3.14159265358979323846264338327950288419716939937510;
   exp1 = 2.71828182845904523536028747135266249775724709369995;


PROCEDURE __BUILTIN__ sqrt (x: LONGREAL) : LONGREAL ;
PROCEDURE exp (x: LONGREAL) : LONGREAL ;
PROCEDURE ln (x: LONGREAL) : LONGREAL ;
PROCEDURE __BUILTIN__ sin (x: LONGREAL) : LONGREAL ;
PROCEDURE __BUILTIN__ cos (x: LONGREAL) : LONGREAL ;
PROCEDURE tan (x: LONGREAL) : LONGREAL ;
PROCEDURE arctan (x: LONGREAL) : LONGREAL ;
PROCEDURE entier (x: LONGREAL) : INTEGER ;


END LMathLib0.
```

4.1.9 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 ;


PROCEDURE Read (VAR ch: CHAR) ;
PROCEDURE Write (ch: CHAR) ;
PROCEDURE Error (ch: CHAR) ;


(*
   UnBufferedMode - places file descriptor, fd, into an unbuffered mode.
*)

PROCEDURE UnBufferedMode (fd: INTEGER; input: BOOLEAN) ;


(*
   BufferedMode - places file descriptor, fd, into a buffered mode.
*)

PROCEDURE BufferedMode (fd: INTEGER; input: BOOLEAN) ;


(*
   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.
*)

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.
*)

PROCEDURE EchoOff (fd: INTEGER; input: BOOLEAN) ;


END IO.
```

4.1.10 gm2-libs/TimeString

```
DEFINITION MODULE TimeString ;

(*
    Description: Provides time related string manipulation procedures.
*)


EXPORT QUALIFIED GetTimeString ;


(*
   GetTimeString - places the time in ascii format into array, a.

*)

PROCEDURE GetTimeString (VAR a: ARRAY OF CHAR) ;


END TimeString.
```

4.1.11 gm2-libs/Scan

```
DEFINITION MODULE Scan ;

(*
   Description: 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.                  *)

PROCEDURE OpenSource (a: ARRAY OF CHAR) : BOOLEAN ;


(* CloseSource - closes the current source file from reading.     *)

PROCEDURE CloseSource ;


(* GetNextSymbol gets the next source symbol and returns it in a. *)

PROCEDURE GetNextSymbol (VAR a: ARRAY OF CHAR) ;


(* WriteError writes a message, a, under the source line, which   *)
(* attempts to pinpoint the Symbol at fault.                      *)

PROCEDURE WriteError (a: ARRAY OF CHAR) ;


(*
   TerminateOnError - exits with status 1 if we call WriteError.
*)

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.
*)

PROCEDURE DefineComments (Start, End: ARRAY OF CHAR; eoln: BOOLEAN) ;


END Scan.
```

4.1.12 gm2-libs/dtoa

```
DEFINITION MODULE dtoa ;

(*
    Description: provides routines to convert between a C double
                 and an ascii string.  The reason we include this
                 module as well as the ldtoa is that the long doubles
                 might be implemented in software on some targets.
                 Hence the libraries should use dtoa if converting
                 REAL or SHORTREAL to/from ascii.
*)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
   Mode = (maxsignificant, decimaldigits) ;


(*
   strtod - returns a REAL given a string, s.  It will set
            error to TRUE if the number is too large.
*)

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?
*)

PROCEDURE dtoa (d        : REAL;
                mode     : Mode;
                ndigits  : INTEGER;
        VAR decpt: INTEGER;
        VAR sign : BOOLEAN) : ADDRESS ;


END dtoa.
```

4.1.13 gm2-libs/sckt

```
DEFINITION MODULE sckt ;

(*
    Description: provides a minimal interface to tcp sockets.
*)

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 = ADDRESS ;
   tcpClientState = ADDRESS ;


(*
   tcpServerEstablish - returns a tcpState containing the relevant
                        information about a socket declared to receive
                        tcp connections.
*)

PROCEDURE tcpServerEstablish () : tcpServerState ;


(*
   tcpServerEstablishPort - returns a tcpState containing the relevant
                            information about a socket declared to recieve
                            tcp connections. This method attempts to use
                            the port specified by the parameter.
*)

PROCEDURE tcpServerEstablishPort (port: CARDINAL) : tcpServerState ;


(*
   tcpServerAccept - returns a file descriptor once a client has connected and
                     been accepted.
*)
```

```
PROCEDURE tcpServerAccept (s: tcpServerState) : INTEGER ;


(*
   tcpServerPortNo - returns the portNo from structure, s.
*)

PROCEDURE tcpServerPortNo (s: tcpServerState) : CARDINAL ;


(*
   tcpSocketFd - returns the sockFd from structure, s.
*)

PROCEDURE tcpServerSocketFd (s: tcpServerState) : INTEGER ;


(*
   getLocalIP - returns the IP address of this machine.
*)

PROCEDURE getLocalIP (s: tcpServerState) : CARDINAL ;


(*
   tcpServerIP - returns the IP address from structure, s.
*)

PROCEDURE tcpServerIP (s: tcpServerState) : CARDINAL ;


(*
   tcpServerClientIP - returns the IP address of the client who
                       has connected to server, s.
*)

PROCEDURE tcpServerClientIP (s: tcpServerState) : CARDINAL ;


(*
   tcpServerClientPortNo - returns the port number of the client who
                           has connected to server, s.
*)

PROCEDURE tcpServerClientPortNo (s: tcpServerState) : CARDINAL ;
```

```
(*
   tcpClientSocket - returns a file descriptor (socket) which has
                     connected to, serverName:portNo.
*)

PROCEDURE tcpClientSocket (serverName: ADDRESS; portNo: CARDINAL) : tcpClientState


(*
   tcpClientSocketIP - returns a file descriptor (socket) which has
                       connected to, ip:portNo.
*)

PROCEDURE tcpClientSocketIP (ip: CARDINAL; portNo: CARDINAL) : tcpClientState ;


(*
   tcpClientConnect - returns the file descriptor associated with, s,
                      once a connect has been performed.
*)

PROCEDURE tcpClientConnect (s: tcpClientState) : INTEGER ;


(*
   tcpClientPortNo - returns the portNo from structure, s.
*)

PROCEDURE tcpClientPortNo (s: tcpClientState) : INTEGER ;


(*
   tcpClientSocketFd - returns the sockFd from structure, s.
*)

PROCEDURE tcpClientSocketFd (s: tcpClientState) : INTEGER ;


(*
   tcpClientIP - returns the IP address from structure, s.
*)

PROCEDURE tcpClientIP (s: tcpClientState) : CARDINAL ;


END sckt.
```

4.1.14 gm2-libs/UnixArgs

```
DEFINITION MODULE UnixArgs ;

(*
   Description: Implements access to the C arguments argc and argv.
*)

FROM SYSTEM IMPORT ADDRESS ;

EXPORT QUALIFIED ArgC, ArgV ;


VAR
   ArgC: CARDINAL ;
   ArgV: ADDRESS ;


END UnixArgs.
```

4.1.15 gm2-libs/COROUTINES

```
DEFINITION MODULE COROUTINES;

CONST
   UnassignedPriority = 0 ;

TYPE
   INTERRUPTSOURCE = CARDINAL ;
   PROTECTION = [UnassignedPriority..7] ;

END COROUTINES.
```

4.1.16 gm2-libs/Break

```
DEFINITION MODULE Break ;

END Break.
```

4.1.17 gm2-libs/RTExceptions

```
DEFINITION MODULE RTExceptions ;

(*
    Description: 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 ;
   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.
*)

PROCEDURE Raise (number: CARDINAL;
                 file: ADDRESS; line: CARDINAL;
                 column: CARDINAL; function: ADDRESS;
                 message: ADDRESS) ;


(*
   SetExceptionBlock - sets, source, as the active EHB.
*)

PROCEDURE SetExceptionBlock (source: EHBlock) ;


(*
```

```
   GetExceptionBlock - returns the active EHB.
*)

PROCEDURE GetExceptionBlock () : EHBlock ;


(*
   GetTextBuffer - returns the address of the EHB buffer.
*)

PROCEDURE GetTextBuffer (e: EHBlock) : ADDRESS ;


(*
   GetTextBufferSize - return the size of the EHB text buffer.
*)

PROCEDURE GetTextBufferSize (e: EHBlock) : CARDINAL ;


(*
   GetNumber - return the exception number associated with,
               source.
*)

PROCEDURE GetNumber (source: EHBlock) : CARDINAL ;


(*
   InitExceptionBlock - creates and returns a new exception block.
*)

PROCEDURE InitExceptionBlock () : EHBlock ;


(*
   KillExceptionBlock - destroys the EHB, e, and all its handlers.
*)

PROCEDURE KillExceptionBlock (e: EHBlock) : EHBlock ;


(*
   PushHandler - install a handler in EHB, e.
*)

PROCEDURE PushHandler (e: EHBlock; number: CARDINAL; p: ProcedureHandler) ;
```

```
(*
   PopHandler - removes the handler associated with, number, from
                EHB, e.
*)

PROCEDURE PopHandler (e: EHBlock; number: CARDINAL) ;


(*
   DefaultErrorCatch - displays the current error message in
                       the current exception block and then
                       calls HALT.
*)

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.
*)

PROCEDURE BaseExceptionsThrow ;


(*
   IsInExceptionState - returns TRUE if the program is currently
                        in the exception state.
*)

PROCEDURE IsInExceptionState () : BOOLEAN ;


(*
   SetExceptionState - returns the current exception state and
                       then sets the current exception state to,
                       to.
*)

PROCEDURE SetExceptionState (to: BOOLEAN) : BOOLEAN ;
```

```
(*
   SwitchExceptionState - assigns, from, with the current exception
                          state and then assigns the current exception
                          to, to.
*)

PROCEDURE SwitchExceptionState (VAR from: BOOLEAN; to: BOOLEAN) ;


(*
   GetBaseExceptionBlock - returns the initial language exception block
                           created.
*)

PROCEDURE GetBaseExceptionBlock () : EHBlock ;


(*
   SetExceptionSource - sets the current exception source to, source.
*)

PROCEDURE SetExceptionSource (source: ADDRESS) ;


(*
   GetExceptionSource - returns the current exception source.
*)

PROCEDURE GetExceptionSource () : ADDRESS ;


END RTExceptions.
```

4.1.18 gm2-libs/LegacyReal

```
DEFINITION MODULE LegacyReal ;

(*
    Description: provides a legacy definition for REAL.
*)

TYPE
   REAL = SHORTREAL ;


END LegacyReal.
```

4.1.19 gm2-libs/Selective

```
DEFINITION MODULE Selective ;

(*
    Description: provides Modula-2 with access to the select(2) primitive.
*)

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 = ADDRESS ;    (* Hidden type in Selective.c *)
   Timeval = ADDRESS ;    (* Hidden type in Selective.c *)


PROCEDURE Select (nooffds: CARDINAL;
                  readfds, writefds, exceptfds: SetOfFd;
                  timeout: Timeval) : INTEGER ;

PROCEDURE InitTime (sec, usec: CARDINAL) : Timeval ;
PROCEDURE KillTime (t: Timeval) : Timeval ;
PROCEDURE GetTime (t: Timeval; VAR sec, usec: CARDINAL) ;
PROCEDURE SetTime (t: Timeval; sec, usec: CARDINAL) ;
PROCEDURE InitSet () : SetOfFd ;
PROCEDURE KillSet (s: SetOfFd) : SetOfFd ;
PROCEDURE FdZero (s: SetOfFd) ;
PROCEDURE FdSet (fd: INTEGER; s: SetOfFd) ;
PROCEDURE FdClr (fd: INTEGER; s: SetOfFd) ;
PROCEDURE FdIsSet (fd: INTEGER; s: SetOfFd) : BOOLEAN ;
PROCEDURE MaxFdsPlusOne (a, b: INTEGER) : INTEGER ;

(* you must use the raw routines with select - not the FIO buffered routines *)
PROCEDURE WriteCharRaw (fd: INTEGER; ch: CHAR) ;
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)
*)
```

```
PROCEDURE GetTimeOfDay (tv: Timeval) : INTEGER ;

END Selective.
```

4.1.20 gm2-libs/SysStorage

```
DEFINITION MODULE SysStorage ;

(*
    Description: provides dynamic allocation for the system components
                 of a realtime system. 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.
*)

PROCEDURE ALLOCATE (VAR a: ADDRESS ; Size: CARDINAL) ;


(*
   DEALLOCATE - return, Size, bytes to the heap.
                The variable, a, is set to NIL.
*)

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.
*)

PROCEDURE REALLOCATE (VAR a: ADDRESS; Size: CARDINAL) ;


(*
   Available - returns TRUE if, Size, bytes can be allocated.
*)

PROCEDURE Available (Size: CARDINAL) : BOOLEAN;
```

```
(*
   Init - initializes the heap.
*)

PROCEDURE Init ;


END SysStorage.
```

4.1.21 gm2-libs/FIO

```
DEFINITION MODULE FIO ;

(*
    Description: provides a simple buffered file input/output library.
*)


FROM SYSTEM IMPORT ADDRESS, BYTE ;

EXPORT QUALIFIED (* types *)
                 File,
                 (* procedures *)
                 OpenToRead, OpenToWrite, OpenForRandom, Close,
                 EOF, EOLN, 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,
                 (* variables *)
                 StdIn, StdOut, StdErr ;

TYPE
   File = CARDINAL ;

(* the following variables are initialized to their UNIX equivalents *)
VAR
   StdIn, StdOut, StdErr: File ;



(*
   IsNoError - returns a TRUE if no error has occured on file, f.
*)

PROCEDURE IsNoError (f: File) : BOOLEAN ;
```

```
(*
   IsActive - returns TRUE if the file, f, is still active.
*)

PROCEDURE IsActive (f: File) : BOOLEAN ;


(*
   Exists - returns TRUE if a file named, fname exists for reading.
*)

PROCEDURE Exists (fname: ARRAY OF CHAR) : 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.
*)

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.
*)

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.
*)
```

```
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.
*)

PROCEDURE Close (f: File) ;


(* the following functions are functionally equivalent to the above
   except they allow C style names.
*)

PROCEDURE exists        (fname: ADDRESS; flength: CARDINAL) : BOOLEAN ;
PROCEDURE openToRead    (fname: ADDRESS; flength: CARDINAL) : File ;
PROCEDURE openToWrite   (fname: ADDRESS; flength: CARDINAL) : File ;
PROCEDURE openForRandom (fname: ADDRESS; flength: CARDINAL; towrite, newfile: BOOLE


(*
   FlushBuffer - flush contents of the FIO file, f, to libc.
*)

PROCEDURE FlushBuffer (f: File) ;


(*
   ReadNBytes - reads nBytes of a file into memory area, a, 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.
*)

PROCEDURE ReadNBytes (f: File; nBytes: CARDINAL; a: ADDRESS) : CARDINAL ;


(*
   ReadAny - reads HIGH(a) bytes into, a. All input
             is fully buffered, unlike ReadNBytes and thus is more
             suited to small reads.
*)

PROCEDURE ReadAny (f: File; VAR a: ARRAY OF BYTE) ;
```

```
(*
   WriteNBytes - writes nBytes of a file into memory area, a, 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.
*)

PROCEDURE WriteNBytes (f: File; nBytes: CARDINAL; a: 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.
*)

PROCEDURE WriteAny (f: File; VAR a: ARRAY OF BYTE) ;


(*
   WriteChar - writes a single character to file, f.
*)

PROCEDURE WriteChar (f: File; ch: CHAR) ;


(*
   EOF - tests to see whether a file, f, has reached end of file.
*)

PROCEDURE EOF (f: File) : BOOLEAN ;


(*
   EOLN - tests to see whether a file, f, is upon a newline.
          It does NOT consume the newline.
*)

PROCEDURE EOLN (f: File) : BOOLEAN ;


(*
   ReadChar - returns a character read from file, f.
              Sensible to check with IsNoError or EOF after calling
              this function.
```

```
*)

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
                or end of file was seen.
*)

PROCEDURE UnReadChar (f: File ; ch: CHAR) ;


(*
   WriteLine - writes out a linefeed to file, f.
*)

PROCEDURE WriteLine (f: File) ;


(*
   WriteString - writes a string to file, f.
*)

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.
*)

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.
*)

PROCEDURE WriteCardinal (f: File; c: CARDINAL) ;
```

```
(*
   ReadCardinal - reads a CARDINAL from file, f.
                  It reads a bit image of a CARDINAL
                  from file, f.
*)

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)
*)

PROCEDURE GetUnixFileDescriptor (f: File) : INTEGER ;


(*
   SetPositionFromBeginning - sets the position from the beginning of the file.
*)

PROCEDURE SetPositionFromBeginning (f: File; pos: LONGINT) ;


(*
   SetPositionFromEnd - sets the position from the end of the file.
*)

PROCEDURE SetPositionFromEnd (f: File; pos: LONGINT) ;


(*
   FindPosition - returns the current absolute position in file, f.
*)

PROCEDURE FindPosition (f: File) : LONGINT ;


(*
   GetFileName - assigns, a, with the filename associated with, f.
*)

PROCEDURE GetFileName (f: File; VAR a: ARRAY OF CHAR) ;
```

```
(*
   getFileName - returns the address of the filename associated with, f.
*)

PROCEDURE getFileName (f: File) : ADDRESS ;


(*
   getFileNameLength - returns the number of characters associated with filename, f
*)

PROCEDURE getFileNameLength (f: File) : CARDINAL ;


END FIO.
```

4.1.22 gm2-libs/SMathLib0

```
DEFINITION MODULE SMathLib0 ;

(*
    Description: provide access to the SHORTREAL instrinics.
*)

CONST
   pi   = 3.14159265358979323846264338327950288419 72;
   exp1 = 2.71828182845904523536028747135266249775 72;


PROCEDURE __BUILTIN__ sqrt (x: SHORTREAL) : SHORTREAL ;
PROCEDURE exp (x: SHORTREAL) : SHORTREAL ;
PROCEDURE ln (x: SHORTREAL) : SHORTREAL ;
PROCEDURE __BUILTIN__ sin (x: SHORTREAL) : SHORTREAL ;
PROCEDURE __BUILTIN__ cos (x: SHORTREAL) : SHORTREAL ;
PROCEDURE tan (x: SHORTREAL) : SHORTREAL ;
PROCEDURE arctan (x: SHORTREAL) : SHORTREAL ;
PROCEDURE entier (x: SHORTREAL) : INTEGER ;


END SMathLib0.
```

4.1.23 gm2-libs/RTint

```
DEFINITION MODULE RTint ;

(*
    Description: provides users of the COROUTINES library with the
                 ability to create interrupt sources based on
                 file descriptors and timeouts.
*)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
   DespatchVector = PROCEDURE (CARDINAL, CARDINAL, ADDRESS) ;


(*
   InitInputVector - returns an interrupt vector which is associated
                     with the file descriptor, fd.
*)

PROCEDURE InitInputVector (fd: INTEGER; pri: CARDINAL) : CARDINAL ;


(*
   InitOutputVector - returns an interrupt vector which is associated
                      with the file descriptor, fd.
*)

PROCEDURE InitOutputVector (fd: INTEGER; pri: CARDINAL) : CARDINAL ;


(*
   InitTimeVector - returns an interrupt vector associated with
                    the relative time.
*)

PROCEDURE InitTimeVector (micro, secs: CARDINAL; pri: CARDINAL) : CARDINAL ;


(*
   ReArmTimeVector - reprimes the vector, vec, to deliver an interrupt
                     at the new relative time.
*)

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.
*)

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.
*)

PROCEDURE AttachVector (vec: CARDINAL; p: ADDRESS) : ADDRESS ;


(*
   IncludeVector - includes, vec, into the despatcher list of
                   possible interrupt causes.
*)

PROCEDURE IncludeVector (vec: CARDINAL) ;


(*
   ExcludeVector - excludes, vec, from the despatcher list of
                   possible interrupt causes.
*)

PROCEDURE ExcludeVector (vec: CARDINAL) ;


(*
   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.
*)

PROCEDURE Listen (untilInterrupt: BOOLEAN;
                  call: DespatchVector;
```

```
                    pri: CARDINAL) ;

END RTint.
```

4.1.24 gm2-libs/NumberIO

```
DEFINITION MODULE NumberIO ;

(*
   Description: Provides all the input/output of numbers, and also the conversion
                of numbers to strings and visa versa.
*)

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 ;


PROCEDURE ReadCard (VAR x: CARDINAL) ;

PROCEDURE WriteCard (x, n: CARDINAL) ;

PROCEDURE ReadHex (VAR x: CARDINAL) ;

PROCEDURE WriteHex (x, n: CARDINAL) ;

PROCEDURE ReadInt (VAR x: INTEGER) ;

PROCEDURE WriteInt (x: INTEGER ; n: CARDINAL) ;

PROCEDURE CardToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

PROCEDURE StrToCard (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

PROCEDURE HexToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

PROCEDURE StrToHex (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

PROCEDURE IntToStr (x: INTEGER ; n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

PROCEDURE StrToInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

PROCEDURE ReadOct (VAR x: CARDINAL) ;

PROCEDURE WriteOct (x, n: CARDINAL) ;

PROCEDURE OctToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

PROCEDURE StrToOct (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;
```

```
PROCEDURE ReadBin (VAR x: CARDINAL) ;

PROCEDURE WriteBin (x, n: CARDINAL) ;

PROCEDURE BinToStr (x, n: CARDINAL ; VAR a: ARRAY OF CHAR) ;

PROCEDURE StrToBin (a: ARRAY OF CHAR ; VAR x: CARDINAL) ;

PROCEDURE StrToBinInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

PROCEDURE StrToHexInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

PROCEDURE StrToOctInt (a: ARRAY OF CHAR ; VAR x: INTEGER) ;

END NumberIO.
```

4.1.25 gm2-libs/M2EXCEPTION

```
DEFINITION MODULE M2EXCEPTION;

(* Provides facilities for identifying language exceptions *)

TYPE
  M2Exceptions =
    (indexException,     rangeException,         caseSelectException,  invalidLocat
     functionException,  wholeValueException,    wholeDivException,    realValueExc
     realDivException,   complexValueException,  complexDivException,  protExceptio
     sysException,       coException,            exException
    );

PROCEDURE M2Exception (): M2Exceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     of a language exception, returns the corresponding enumeration value, and othe
     raises an exception.
  *)

PROCEDURE IsM2Exception (): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state because of the
     of a language exception, returns TRUE, and otherwise returns FALSE.
  *)

END M2EXCEPTION.
```

4.1.26 gm2-libs/wrapc

```
DEFINITION MODULE wrapc ;

(*
   Description: Provides a Modula-2 interface to the C
                library functionality.
*)

FROM SYSTEM IMPORT ADDRESS ;

EXPORT QUALIFIED strtime, filesize, getrand, getusername, filemtime,
                 getnameuidgid, signbit, signbitf, signbitl ;


(*
   strtime - returns the C string for the equivalent C asctime
             function.
*)

PROCEDURE strtime () : ADDRESS ;


(*
   filesize - assigns the size of a file, f, into low, high and
              returns zero if successful.
*)

PROCEDURE filesize (f: INTEGER; VAR low, high: CARDINAL) : INTEGER ;


(*
   filemtime - returns the mtime of a file, f.
*)

PROCEDURE filemtime (f: INTEGER) : INTEGER ;


(*
   getrand - returns a random number between 0..n-1
*)

PROCEDURE getrand (n: INTEGER) : INTEGER ;


(*
   getusername - returns a C string describing the current user.
```

```
*)

PROCEDURE getusername () : ADDRESS ;


(*
   getnameuidgid - fills in the, uid, and, gid, which represents
                   user, name.
*)

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.
*)

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


END wrapc.
```

4.1.27 gm2-libs/cxxabi

```
DEFINITION MODULE FOR "C" cxxabi ;

(*
    Description: provides a minimal Modula-2 implementation of the
                 C++ABI exception handling routines used in g++.
                 This should only be used by the compiler.
*)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT UNQUALIFIED __cxa_begin_catch, __cxa_end_catch, __cxa_rethrow ;


PROCEDURE __cxa_begin_catch (a: ADDRESS) : ADDRESS ;
PROCEDURE __cxa_end_catch ;
PROCEDURE __cxa_rethrow ;


END cxxabi.
```

4.1.28 gm2-libs/StringConvert

```
DEFINITION MODULE StringConvert ;

(*
    Description: provides functions to convert numbers to and from strings.
*)

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.
*)

PROCEDURE IntegerToString (i: INTEGER; width: CARDINAL; padding: CHAR; sign: BOOLEA
                           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
                      abcdef are used, and if FALSE then ABCDEF are used.
*)
```

```
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.
*)

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.
*)

PROCEDURE StringToCardinal (s: String; base: CARDINAL; VAR found: BOOLEAN) : CARDIN


(*
   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.
*)

PROCEDURE LongIntegerToString (i: LONGINT; width: CARDINAL; padding: CHAR;
                               sign: BOOLEAN; base: CARDINAL; lower: BOOLEAN) : Str


(*
   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.
   *)

   PROCEDURE StringToLongInteger (s: String; base: CARDINAL; VAR found: BOOLEAN) : LON


   (*
      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.
   *)

   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.
   *)

   PROCEDURE StringToLongCardinal (s: String; base: CARDINAL; VAR found: BOOLEAN) : LO


   (*
      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.
   *)

   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.
                           The parameter found is set TRUE if a number was found.
*)

PROCEDURE StringToShortCardinal (s: String; base: CARDINAL;
                                 VAR found: BOOLEAN) : SHORTCARD ;


(*
   stoi - decimal string to INTEGER
*)

PROCEDURE stoi (s: String) : INTEGER ;


(*
   itos - integer to decimal string.
*)

PROCEDURE itos (i: INTEGER; width: CARDINAL; padding: CHAR; sign: BOOLEAN) : String


(*
   ctos - cardinal to decimal string.
*)

PROCEDURE ctos (c: CARDINAL; width: CARDINAL; padding: CHAR) : String ;


(*
   stoc - decimal string to CARDINAL
*)

PROCEDURE stoc (s: String) : CARDINAL ;


(*
   hstoi - hexidecimal string to INTEGER
*)

PROCEDURE hstoi (s: String) : INTEGER ;
```

```
(*
   ostoi - octal string to INTEGER
*)

PROCEDURE ostoi (s: String) : INTEGER ;


(*
   bstoi - binary string to INTEGER
*)

PROCEDURE bstoi (s: String) : INTEGER ;


(*
   hstoc - hexidecimal string to CARDINAL
*)

PROCEDURE hstoc (s: String) : CARDINAL ;


(*
   ostoc - octal string to CARDINAL
*)

PROCEDURE ostoc (s: String) : CARDINAL ;


(*
   bstoc - binary string to CARDINAL
*)

PROCEDURE bstoc (s: String) : CARDINAL ;


(*
   StringToLongreal - returns a LONGREAL and sets found to TRUE
                      if a legal number is seen.
*)

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
                        will return the value of x which is converted
                        into as a fixed point number with exhaustive
                        precision.
*)

PROCEDURE LongrealToString (x: LONGREAL;
                            TotalWidth, FractionWidth: CARDINAL) : String ;


(*
   stor - returns a REAL given a string.
*)

PROCEDURE stor (s: String) : REAL ;


(*
   stolr - returns a LONGREAL given a string.
*)

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
*)

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 .

                     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
*)

PROCEDURE ToDecimalPlaces (s: String; n: CARDINAL) : String ;


END StringConvert.
```

4.1.29 gm2-libs/Args

```
DEFINITION MODULE Args ;

(*
    Description: provides a simple interface to the command
                 line arguments.
*)

EXPORT QUALIFIED GetArg, Narg ;


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

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


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

PROCEDURE Narg() : CARDINAL ;


END Args.
```

4.1.30 gm2-libs/MathLib0

```
DEFINITION MODULE MathLib0 ;

(*
    Description: provides access to math functions.
*)

CONST
   pi   = 3.14159265358979323846264338327950288419 72;
   exp1 = 2.71828182845904523536028747135266249775 72;


PROCEDURE __BUILTIN__ sqrt (x: REAL) : REAL ;
PROCEDURE exp (x: REAL) : REAL ;
PROCEDURE ln (x: REAL) : REAL ;
PROCEDURE __BUILTIN__ sin (x: REAL) : REAL ;
PROCEDURE __BUILTIN__ cos (x: REAL) : REAL ;
PROCEDURE tan (x: REAL) : REAL ;
PROCEDURE arctan (x: REAL) : REAL ;
PROCEDURE entier (x: REAL) : INTEGER ;


END MathLib0.
```

4.1.31 gm2-libs/FormatStrings

```
DEFINITION MODULE FormatStrings ;

(*
    Description: provides a pseudo printf capability for GM2.
*)

FROM SYSTEM IMPORT BYTE ;
FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED Sprintf0, Sprintf1, Sprintf2, Sprintf3, Sprintf4 ;


(*
   Sprintf0 - returns a String containing, s, after it has had its
              escape sequences translated.
*)

PROCEDURE Sprintf0 (s: String) : String ;


(*
   Sprintf1 - returns a String containing, s, together with encapsulated
              entity, w. It only formats the first %s or %d with n.
*)

PROCEDURE Sprintf1 (s: String; w: ARRAY OF BYTE) : String ;


(*
   Sprintf2 - returns a string, s, which has been formatted.
*)

PROCEDURE Sprintf2 (s: String; w1, w2: ARRAY OF BYTE) : String ;


(*
   Sprintf3 - returns a string, s, which has been formatted.
*)

PROCEDURE Sprintf3 (s: String; w1, w2, w3: ARRAY OF BYTE) : String ;


(*
   Sprintf4 - returns a string, s, which has been formatted.
*)
```

```
PROCEDURE Sprintf4 (s: String; w1, w2, w3, w4: ARRAY OF BYTE) : String ;

END FormatStrings.
```

4.1.32 gm2-libs/StdIO

```
DEFINITION MODULE StdIO ;

(*
   Description: Exports a general Read and Write procedure that ALL character
                processes should use.
*)

EXPORT QUALIFIED ProcRead, ProcWrite,
                 Read, Write,
                 PushOutput, PopOutput, GetCurrentOutput,
                 PushInput, PopInput, GetCurrentInput ;


TYPE
   ProcWrite = PROCEDURE (CHAR) ;
   ProcRead  = PROCEDURE (VAR CHAR) ;


(*
   Read - is the generic procedure that all higher application layers
          should use to receive a character.
*)

PROCEDURE Read (VAR ch: CHAR) ;


(*
   Write - is the generic procedure that all higher application layers
           should use to emit a character.
*)

PROCEDURE Write (ch: CHAR) ;


(*
   PushOutput - pushes the current Write procedure onto a stack,
                any future references to Write will actually invoke
                procedure, p.
*)

PROCEDURE PushOutput (p: ProcWrite) ;


(*
   PopOutput - restores Write to use the previous output procedure.
```

```
*)

PROCEDURE PopOutput ;


(*
   GetCurrentOutput - returns the current output procedure.
*)

PROCEDURE GetCurrentOutput () : ProcWrite ;


(*
   PushInput - pushes the current Read procedure onto a stack,
               any future references to Read will actually invoke
               procedure, p.
*)

PROCEDURE PushInput (p: ProcRead) ;


(*
   PopInput - restores Write to use the previous output procedure.
*)

PROCEDURE PopInput ;


(*
   GetCurrentInput - returns the current input procedure.
*)

PROCEDURE GetCurrentInput () : ProcRead ;


END StdIO.
```

4.1.33 gm2-libs/Assertion

```
DEFINITION MODULE Assertion ;

(*
   Description: Provides an assert procedure.
*)

EXPORT QUALIFIED Assert ;


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

PROCEDURE Assert (Condition: BOOLEAN) ;


END Assertion.
```

4.1.34 gm2-libs/ASCII

```
DEFINITION MODULE ASCII ;

(*
   Desciption: Defines all ascii constants (as in man ASCII)
               Note that lf, eof and EOL are added
*)

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 ;

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

END ASCII.
```

4.1.35 gm2-libs/cbuiltin

```
DEFINITION MODULE FOR "C" cbuiltin ;

(*
    Description: provides replacement routines in case the builtins are
                 not used by GNU Modula-2.  This module is called by
                 implementation modules which implement builtins
                 (see Builtins.mod for an example).
*)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT UNQUALIFIED alloca, memcpy,
                   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 ;

PROCEDURE alloca (i: CARDINAL) : ADDRESS ;
PROCEDURE memcpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
PROCEDURE sinf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE sin (x: REAL) : REAL ;
PROCEDURE sinl (x: LONGREAL) : LONGREAL ;
PROCEDURE cosf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE cos (x: REAL) : REAL ;
PROCEDURE cosl (x: LONGREAL) : LONGREAL ;
PROCEDURE atan2f (x, y: SHORTREAL) : SHORTREAL ;
PROCEDURE atan2 (x, y: REAL) : REAL ;
PROCEDURE atan2l (x, y: LONGREAL) : LONGREAL ;
PROCEDURE sqrtf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE sqrt (x: REAL) : REAL ;
PROCEDURE sqrtl (x: LONGREAL) : LONGREAL ;
PROCEDURE fabsf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE fabs (x: REAL) : REAL ;
PROCEDURE fabsl (x: LONGREAL) : LONGREAL ;
PROCEDURE logf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE log (x: REAL) : REAL ;
PROCEDURE logl (x: LONGREAL) : LONGREAL ;
PROCEDURE expf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE exp (x: REAL) : REAL ;
PROCEDURE expl (x: LONGREAL) : LONGREAL ;
PROCEDURE log10f (x: SHORTREAL) : SHORTREAL ;
PROCEDURE log10 (x: REAL) : REAL ;
PROCEDURE log10l (x: LONGREAL) : LONGREAL ;
PROCEDURE exp10f (x: SHORTREAL) : SHORTREAL ;
PROCEDURE exp10 (x: REAL) : REAL ;
PROCEDURE exp10l (x: LONGREAL) : LONGREAL ;
PROCEDURE ilogbf (x: SHORTREAL) : INTEGER ;
PROCEDURE ilogb (x: REAL) : INTEGER ;
PROCEDURE ilogbl (x: LONGREAL) : INTEGER ;

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

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

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

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

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

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

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

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

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

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

PROCEDURE cpowf (base: SHORTCOMPLEX; exp: SHORTREAL) : SHORTCOMPLEX ;
PROCEDURE cpow (base: COMPLEX; exp: REAL) : COMPLEX ;
PROCEDURE cpowl (base: LONGCOMPLEX; exp: LONGREAL) : LONGCOMPLEX ;

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

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

PROCEDURE clogf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
PROCEDURE clog (z: COMPLEX) : COMPLEX ;
PROCEDURE clogl (z: LONGCOMPLEX) : LONGCOMPLEX ;

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

```
PROCEDURE csinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

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

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

PROCEDURE casinf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
PROCEDURE casin (z: COMPLEX) : COMPLEX ;
PROCEDURE casinl (z: LONGCOMPLEX) : LONGCOMPLEX ;

PROCEDURE cacosf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
PROCEDURE cacos (z: COMPLEX) : COMPLEX ;
PROCEDURE cacosl (z: LONGCOMPLEX) : LONGCOMPLEX ;

PROCEDURE catanf (z: SHORTCOMPLEX) : SHORTCOMPLEX ;
PROCEDURE catan (z: COMPLEX) : COMPLEX ;
PROCEDURE catanl (z: LONGCOMPLEX) : LONGCOMPLEX ;

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

END cbuiltin.
```

4.1.36 gm2-libs/FpuIO

```
DEFINITION MODULE FpuIO ;

(*
   Description: Implements a fixed format input/output for REAL,
                LONGREAL and LONGINT numbers
*)

EXPORT QUALIFIED ReadReal, WriteReal, StrToReal, RealToStr,
                 ReadLongReal, WriteLongReal, StrToLongReal, LongRealToStr,
                 ReadLongInt, WriteLongInt, StrToLongInt, LongIntToStr ;


PROCEDURE ReadReal (VAR x: REAL) ;
PROCEDURE WriteReal (x: REAL; TotalWidth, FractionWidth: CARDINAL) ;
PROCEDURE StrToReal (a: ARRAY OF CHAR ; VAR x: REAL) ;
PROCEDURE RealToStr (x: REAL; TotalWidth, FractionWidth: CARDINAL; VAR a: ARRAY OF

PROCEDURE ReadLongReal (VAR x: LONGREAL) ;
PROCEDURE WriteLongReal (x: LONGREAL; TotalWidth, FractionWidth: CARDINAL) ;
PROCEDURE StrToLongReal (a: ARRAY OF CHAR ; VAR x: LONGREAL) ;
PROCEDURE LongRealToStr (x: LONGREAL; TotalWidth, FractionWidth: CARDINAL; VAR a: A

PROCEDURE ReadLongInt (VAR x: LONGINT) ;
PROCEDURE WriteLongInt (x: LONGINT; n: CARDINAL) ;
PROCEDURE StrToLongInt (a: ARRAY OF CHAR ; VAR x: LONGINT) ;
PROCEDURE LongIntToStr (x: LONGINT; n: CARDINAL; VAR a: ARRAY OF CHAR) ;


END FpuIO.
```

4.1.37 gm2-libs/M2RTS

```
DEFINITION MODULE M2RTS ;

(*
   Description: Implements the run time system facilities of Modula-2.
*)

FROM SYSTEM IMPORT BITSET, ADDRESS ;
EXPORT QUALIFIED HALT, Halt,
                 InstallTerminationProcedure, ExecuteTerminationProcedures,
                 InstallInitialProcedure, ExecuteInitialProcedures,
                 ExitOnHalt, Terminate, Length, ErrorMessage,
                 AssignmentException,
                 IncException, DecException, InclException, ExclException,
                 ShiftException, RotateException,
                 StaticArraySubscriptException, DynamicArraySubscriptException,
                 ForLoopBeginException, ForLoopToException, ForLoopEndException,
                 PointerNilException, NoReturnException,
                 WholeNonPosDivException, WholeNonPosModException,
                 WholeZeroDivException, WholeZeroRemException,
                 CaseException, NoException ;


(*
   ExecuteTerminationProcedures - calls each installed termination
                                  procedure in reverse order.
*)

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.
*)

PROCEDURE InstallTerminationProcedure (p: PROC) : BOOLEAN ;


(*
   ExecuteInitialProcedures - executes the initial procedures installed
                              by InstallInitialProcedure.
*)
```

```
PROCEDURE ExecuteInitialProcedures ;


(*
   InstallInitialProcedure - installs a procedure to be executed just
                             before the BEGIN code section of the main
                             program module.
*)

PROCEDURE InstallInitialProcedure (p: PROC) : BOOLEAN ;


(*
   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.
*)

PROCEDURE Terminate ;


(*
   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.
*)

PROCEDURE HALT ([exitcode: INTEGER = -1]) ;


(*
   Halt - provides a more user friendly version of HALT, which takes
          four parameters to aid debugging.
*)

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.
```

```
*)

PROCEDURE ExitOnHalt (e: INTEGER) ;


(*
   ErrorMessage - emits an error message to stderr and then calls exit (1).
*)

PROCEDURE ErrorMessage (message: ARRAY OF CHAR;
                        file: ARRAY OF CHAR;
                        line: CARDINAL;
                        function: ARRAY OF CHAR) ;


(*
   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.
*)

PROCEDURE Length (a: ARRAY OF CHAR) : CARDINAL ;


(*
   The following are the runtime exception handler routines.
*)

PROCEDURE AssignmentException (filename: ADDRESS; line, column: CARDINAL; scope: AD
PROCEDURE IncException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE DecException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE InclException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE ExclException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE ShiftException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS
PROCEDURE RotateException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRES
PROCEDURE StaticArraySubscriptException (filename: ADDRESS; line, column: CARDINAL;
PROCEDURE DynamicArraySubscriptException (filename: ADDRESS; line, column: CARDINAL
PROCEDURE ForLoopBeginException (filename: ADDRESS; line, column: CARDINAL; scope:
PROCEDURE ForLoopToException (filename: ADDRESS; line, column: CARDINAL; scope: ADD
PROCEDURE ForLoopEndException (filename: ADDRESS; line, column: CARDINAL; scope: AD
PROCEDURE PointerNilException (filename: ADDRESS; line, column: CARDINAL; scope: AD
PROCEDURE NoReturnException (filename: ADDRESS; line, column: CARDINAL; scope: ADDR
PROCEDURE CaseException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE WholeNonPosDivException (filename: ADDRESS; line, column: CARDINAL; scope
PROCEDURE WholeNonPosModException (filename: ADDRESS; line, column: CARDINAL; scope
PROCEDURE WholeZeroDivException (filename: ADDRESS; line, column: CARDINAL; scope:
PROCEDURE WholeZeroRemException (filename: ADDRESS; line, column: CARDINAL; scope:
```

```
PROCEDURE NoException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS) ;

END M2RTS.
```

4.1.38 gm2-libs/errno

```
DEFINITION MODULE errno ;

(*
    Description: provides a Modula-2 interface to the C errno.
*)

CONST
    EINTR  =  4 ;   (* system call interrupted *)
    ERANGE = 34 ;   (* result is too large     *)
    EAGAIN = 11 ;   (* retry the system call   *)

PROCEDURE geterrno () : INTEGER ;


END errno.
```

4.1.39 gm2-libs/DynamicStrings

```
DEFINITION MODULE DynamicStrings ;

(*
    Description: provides a dynamic string type and common methods.
*)

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 ;


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

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


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

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.
*)
```

```
PROCEDURE Fin (s: String) ;


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

PROCEDURE InitStringCharStar (a: ADDRESS) : String ;


(*
   InitStringChar - initializes and returns a String to contain the single characte
*)

PROCEDURE InitStringChar (ch: CHAR) : String ;


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

PROCEDURE Mark (s: String) : String ;


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

PROCEDURE Length (s: String) : CARDINAL ;


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

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


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

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


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

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

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


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

PROCEDURE Dup (s: String) : String ;


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

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


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

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


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

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


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

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.
*)
```

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


(*
   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
*)

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.
*)

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.
*)

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.
*)

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

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

PROCEDURE RemoveWhitePrefix (s: String) : String ;


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

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.
*)

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.
*)

PROCEDURE ToLower (s: String) : String ;


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

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.
*)

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


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

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.
*)

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


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

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

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

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


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

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


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

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


(*
   SliceDB - debug version of Slice.
*)

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

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

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.
*)
```

```
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,
*)

PROCEDURE PopAllocationExemption (halt: BOOLEAN; e: String) : String ;


END DynamicStrings.
```

4.1.40 gm2-libs/CmdArgs

```
DEFINITION MODULE CmdArgs ;

(*
   Description: CmdArgs - implements procedures to retrieve arguments from
                a string.
*)

EXPORT QUALIFIED GetArg, Narg ;


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

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.
*)

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


END CmdArgs.
```

4.1.41 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.
*)

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.
*)

PROCEDURE DebugString (a: ARRAY OF CHAR) ;


END Debug.
```

4.1.42 gm2-libs/SFIO

```
DEFINITION MODULE SFIO ;

(*
    Description: provides a String interface to the opening routines of FIO
*)

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.
*)

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.
*)

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.
*)

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
                    be left alone, allowing this descriptor to seek
                    and modify an existing file.
*)

PROCEDURE OpenForRandom (fname: String; towrite, newfile: BOOLEAN) : File ;


(*
   WriteS - writes a string, s, to, file. It returns the String, s.
*)

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.
*)

PROCEDURE ReadS (file: File) : String ;


END SFIO.
```

4.1.43 gm2-libs/SYSTEM

```
DEFINITION MODULE SYSTEM ;

(*
   Description: Implements the SYSTEM dependent module
                in the Modula-2 compiler.
*)

EXPORT QUALIFIED BITSPERBYTE, BYTESPERWORD,
                 LOC, WORD, BYTE, ADDRESS, BITSET,
                 INTEGER8, INTEGER16, INTEGER32, INTEGER64,
                 CARDINAL8, CARDINAL16, CARDINAL32, CARDINAL64,
                 WORD16, WORD32, WORD64, BITSET8,
                 BITSET16, BITSET32, REAL32, REAL64,
                 REAL96, REAL128, COMPLEX32, COMPLEX64,
                 COMPLEX96, COMPLEX128,
                 ADR, TSIZE, ROTATE, SHIFT, THROW ;
                 (* SIZE is also exported if -fpim2 is used *)

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


(* all the following types are declared internally to gm2
TYPE
   LOC ;
   WORD ;
   BYTE ;
   ADDRESS ;
   BITSET ;
   INTEGER8 ;
   INTEGER16 ;
   INTEGER32 ;
   INTEGER64 ;
   CARDINAL8 ;
   CARDINAL16 ;
   CARDINAL32 ;
   CARDINAL64 ;
   WORD16 ;
   WORD32 ;
   WORD64 ;
   BITSET8 ;
   BITSET16 ;
   BITSET32 ;
   REAL32 ;
```

```
   REAL64 ;
   REAL96 ;
   REAL128 ;
   COMPLEX32 ;
   COMPLEX64 ;
   COMPLEX96 ;
   COMPLEX128 ;
*)


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

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

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.
  *)

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

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.
  *)

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.
  *)

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.
  *)
*)

(* 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.
*)

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.
*)

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.
```

```
*)

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.
*)

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.
*)

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.
*)

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


END SYSTEM.
```

4.1.44 gm2-libs/SArgs

```
DEFINITION MODULE SArgs ;

(*
    Description: provides a String interface to the command line arguments.
*)

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.
*)

PROCEDURE GetArg (VAR s: String ; i: CARDINAL) : BOOLEAN ;


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

PROCEDURE Narg() : CARDINAL ;


END SArgs.
```

4.1.45 gm2-libs/SysExceptions

```
DEFINITION MODULE SysExceptions ;

(*
    Description: provides a mechanism for the underlying libraries to
                 configure the hardware exception routines either via
                 Posix signal or maybe an alternative method.  Used by
                 both the ISO and PIM libraries.  It is written to be
                 ISO compliant though.
*)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
   PROCEXCEPTION = PROCEDURE (ADDRESS) ;

PROCEDURE InitExceptionHandlers (indexf, range, casef, invalidloc,
                                 function, wholevalue, wholediv,
                                 realvalue, realdiv, complexvalue,
                                 complexdiv, protection, systemf,
                                 coroutine, exception: PROCEXCEPTION) ;


END SysExceptions.
```

4.1.46 gm2-libs/termios

```
DEFINITION MODULE termios ;

(*
    Description: provides a procedural interface to termios.
*)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
   TERMIOS = ADDRESS ;

   ControlChar = (vintr, vquit, verase, vkill, veof, vtime, vmin,
                  vswtc, vstart, vstop, vsusp, veol, vreprint, vdiscard,
                  vwerase, vlnext, veol2) ;

   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.
*)

PROCEDURE InitTermios () : TERMIOS ;


(*
   KillTermios - delete data structure.
```

```
*)

PROCEDURE KillTermios (t: TERMIOS) : TERMIOS ;


(*
   cfgetospeed - return output baud rate.
*)

PROCEDURE cfgetospeed (t: TERMIOS) : INTEGER ;


(*
   cfgetispeed - return input baud rate.
*)

PROCEDURE cfgetispeed (t: TERMIOS) : INTEGER ;


(*
   cfsetospeed - set output baud rate.
*)

PROCEDURE cfsetospeed (t: TERMIOS; b: CARDINAL) : INTEGER ;


(*
   cfsetispeed - set input baud rate.
*)

PROCEDURE cfsetispeed (t: TERMIOS; b: CARDINAL) : INTEGER ;


(*
   cfsetspeed - set input and output baud rate.
*)

PROCEDURE cfsetspeed (t: TERMIOS; b: CARDINAL) : INTEGER ;


(*
   tcgetattr - get state of, fd, into, t.
*)

PROCEDURE tcgetattr (fd: INTEGER; t: TERMIOS) : INTEGER ;
```

```
(*
   The following three functions return the different option values.
*)

PROCEDURE tcsnow () : INTEGER ;   (* alter fd now *)
PROCEDURE tcsdrain () : INTEGER ; (* alter when all output has been sent *)
PROCEDURE tcsflush () : INTEGER ; (* like drain, except discard any pending input *


(*
   tcsetattr - set state of, fd, to, t, using option.
*)

PROCEDURE tcsetattr (fd: INTEGER; option: INTEGER; t: TERMIOS) : INTEGER ;


(*
   cfmakeraw - sets, t, to raw mode.
*)

PROCEDURE cfmakeraw (t: TERMIOS) ;


(*
   tcsendbreak - send zero bits for duration.
*)

PROCEDURE tcsendbreak (fd: INTEGER; duration: INTEGER) : INTEGER ;


(*
   tcdrain - waits for pending output to be written on, fd.
*)

PROCEDURE tcdrain (fd: INTEGER) : INTEGER ;


(*
   tcflushi - flush input.
*)

PROCEDURE tcflushi (fd: INTEGER) : INTEGER ;


(*
   tcflusho - flush output.
*)
```

```
PROCEDURE tcflusho (fd: INTEGER) : INTEGER ;


(*
   tcflushio - flush input and output.
*)

PROCEDURE tcflushio (fd: INTEGER) : INTEGER ;


(*
   tcflowoni - restart input on, fd.
*)

PROCEDURE tcflowoni (fd: INTEGER) : INTEGER ;


(*
   tcflowoffi - stop input on, fd.
*)

PROCEDURE tcflowoffi (fd: INTEGER) : INTEGER ;


(*
   tcflowono - restart output on, fd.
*)

PROCEDURE tcflowono (fd: INTEGER) : INTEGER ;


(*
   tcflowoffo - stop output on, fd.
*)

PROCEDURE tcflowoffo (fd: INTEGER) : INTEGER ;


(*
   GetFlag - sets a flag value from, t, in, b, and returns TRUE
             if, t, supports, f.
*)

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.
*)

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.
*)

PROCEDURE GetChar (t: TERMIOS; c: ControlChar; VAR ch: CHAR) : BOOLEAN ;


(*
   SetChar - sets a CHAR value in, t, and returns TRUE if, c,
             is supported.
*)

PROCEDURE SetChar (t: TERMIOS; c: ControlChar; ch: CHAR) : BOOLEAN ;


END termios.
```

4.1.47 gm2-libs/Indexing

```
DEFINITION MODULE Indexing ;

(*
    Description: provides a dynamic indexing mechanism.
*)

FROM SYSTEM IMPORT ADDRESS ;
EXPORT QUALIFIED Index, InitIndex, KillIndex, GetIndice, PutIndice,
                 HighIndice, LowIndice, InBounds, IsIndiceInIndex,
                 RemoveIndiceFromIndex, IncludeIndiceIntoIndex,
                 ForeachIndiceInIndexDo, DebugIndex ;

TYPE
   Index ;
   IndexProcedure = PROCEDURE (ADDRESS) ;


(*
   InitIndex - creates and returns an Index.
*)

PROCEDURE InitIndex (low: CARDINAL) : Index ;


(*
   KillIndex - returns Index to free storage.
*)

PROCEDURE KillIndex (i: Index) : Index ;


(*
   DebugIndex - turns on debugging within an index.
*)

PROCEDURE DebugIndex (i: Index) : Index ;


(*
   InBounds - returns TRUE if indice, n, is within the bounds
              of the dynamic array.
*)

PROCEDURE InBounds (i: Index; n: CARDINAL) : BOOLEAN ;
```

```
(*
   HighIndice - returns the last legally accessible indice of this array.
*)

PROCEDURE HighIndice (i: Index) : CARDINAL ;


(*
   LowIndice - returns the first legally accessible indice of this array.
*)

PROCEDURE LowIndice (i: Index) : CARDINAL ;


(*
   PutIndice - places, a, into the dynamic array at position i[n]
*)

PROCEDURE PutIndice (i: Index; n: CARDINAL; a: ADDRESS) ;


(*
   GetIndice - retrieves, element i[n] from the dynamic array.
*)

PROCEDURE GetIndice (i: Index; n: CARDINAL) : ADDRESS ;


(*
   IsIndiceInIndex - returns TRUE if, a, is in the index, i.
*)

PROCEDURE IsIndiceInIndex (i: Index; a: ADDRESS) : BOOLEAN ;


(*
   RemoveIndiceFromIndex - removes, a, from Index, i.
*)

PROCEDURE RemoveIndiceFromIndex (i: Index; a: ADDRESS) ;


(*
   IncludeIndiceIntoIndex - if the indice is not in the index, then
                            add it at the end.
*)
```

```
PROCEDURE IncludeIndiceIntoIndex (i: Index; a: ADDRESS) ;


(*
   ForeachIndiceInIndexDo - for each j indice of i, call procedure p(i[j])
*)

PROCEDURE ForeachIndiceInIndexDo (i: Index; p: IndexProcedure) ;


END Indexing.
```

4.1.48 gm2-libs/ldtoa

```
DEFINITION MODULE ldtoa ;

(*
    Description: provides routines to convert between a C long double
                 and an ascii string.
*)


FROM SYSTEM IMPORT ADDRESS ;

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.
*)

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?
*)

PROCEDURE ldtoa (d        : LONGREAL;
                 mode     : Mode;
                 ndigits  : INTEGER;
                 VAR decpt: INTEGER;
                 VAR sign : BOOLEAN) : ADDRESS ;


END ldtoa.
```

4.1.49 gm2-libs/libc

```
DEFINITION MODULE FOR "C" libc ;

(*
   Description: Provides an interface to the C library functions.
*)

FROM SYSTEM IMPORT ADDRESS ;

EXPORT UNQUALIFIED time_t, timeb, tm, ptrToTM,
                   write, read,
                   system, abort,
                   malloc, free,
                   exit, isatty,
                   getenv, getpid,
                   dup, close, open, lseek,
                   readv, writev,
                   perror, creat,
                   getcwd, chown, strlen, strcpy, strncpy,
                   unlink, setenv,
                   memcpy, memset, printf, realloc,
                   rand, srand,
                   time, localtime, ftime,
                   shutdown, rename, setjmp, longjmp, atexit,
                   ttyname ;

TYPE
   time_t = LONGINT ;

   ptrToTM = POINTER TO tm ;
   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 ;

   timeb = RECORD
              time    : time_t ;
```

```
            millitm : SHORTCARD ;
            timezone: SHORTCARD ;
            dstflag : SHORTCARD ;
         END ;


(*
     int write(d, buf, nbytes)
     int d;
     char *buf;
     int nbytes;
*)

PROCEDURE write (d: INTEGER; buf: ADDRESS; nbytes: INTEGER) : INTEGER ;


(*
     int read(d, buf, nbytes)
     int d;
     char *buf;
     int nbytes;
*)

PROCEDURE read (d: INTEGER; buf: ADDRESS; nbytes: INTEGER) : INTEGER ;


(*
     int system(string)
     char *string;
*)

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.
*)
```

```
PROCEDURE abort ;


(*
     malloc - memory allocator.

     char *malloc(size)
     unsigned 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.
*)

PROCEDURE malloc (size: CARDINAL) : ADDRESS ;


(*
     free - memory deallocator.

     free(ptr)
     char *ptr;

     free() releases a previously allocated block.  Its argument
     is a pointer to a block previously allocated by malloc,
     calloc, realloc, malloc, or memalign.
*)

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.
*)

PROCEDURE realloc (ptr: ADDRESS; size: CARDINAL) : ADDRESS ;
```

```
(*
   isatty - does this descriptor refer to a terminal.
*)

PROCEDURE isatty (fd: INTEGER) : INTEGER ;


(*
   exit - returns control to the invoking process. Result, r, is
          returned.
*)

PROCEDURE exit (r: INTEGER) ;


(*
   getenv - returns the C string for the equivalent C environment
            variable.
*)

PROCEDURE getenv (s: ADDRESS) : ADDRESS ;


(*
   getpid - returns the UNIX process identification number.
*)

PROCEDURE getpid () : INTEGER ;


(*
   dup - duplicates the file descriptor, d.
*)

PROCEDURE dup (d: INTEGER) : INTEGER ;


(*
   close - closes the file descriptor, d.
*)

PROCEDURE close (d: INTEGER) : INTEGER ;


(*
   open - open the file, filename with flag and mode.
```

```
*)

PROCEDURE open (filename: ADDRESS; oflag: INTEGER; ...) : INTEGER ;


(*
   creat - creates a new file
*)

PROCEDURE creat (filename: ADDRESS; mode: CARDINAL) : INTEGER;


(*
   lseek - calls unix lseek:

           off_t lseek(int fildes, off_t offset, int whence);
*)

PROCEDURE lseek (fd: INTEGER; offset: LONGINT; whence: INTEGER) : LONGINT ;


(*
   perror - writes errno and string. (ARRAY OF CHAR is translated onto ADDRESS).
*)

PROCEDURE perror (string: ARRAY OF CHAR);


(*
   readv - reads an io vector of bytes.
*)

PROCEDURE readv (fd: INTEGER; v: ADDRESS; n: INTEGER) : INTEGER ;


(*
   writev - writes an io vector of bytes.
*)

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.
*)

PROCEDURE getcwd (buf: ADDRESS; size: INTEGER) : 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.
*)

PROCEDURE chown (filename: ADDRESS; uid, gid: INTEGER) : INTEGER ;


(*
   strlen - returns the length of string, a.
*)

PROCEDURE strlen (a: ADDRESS) : INTEGER ;


(*
   strcpy - copies string, src, into, dest.
            It returns dest.
*)

PROCEDURE strcpy (dest, src: ADDRESS) : ADDRESS ;


(*
   strncpy - copies string, src, into, dest, copying at most, n, bytes.
             It returns dest.
*)

PROCEDURE strncpy (dest, src: ADDRESS; n: CARDINAL) : ADDRESS ;
```

```
(*
   unlink - removes file and returns 0 if successful.
*)

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.
*)

PROCEDURE memcpy (dest, src: ADDRESS; size: CARDINAL) : ADDRESS ;


(*
   memset - fill memory with a constant byte

   SYNOPSIS

   #include <string.h>

   void *memset(void *s, int c, size_t n);
*)

PROCEDURE memset (s: ADDRESS; c: INTEGER; size: CARDINAL) : ADDRESS ;


(*
   int printf(const char *format, ...);
*)

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.
```

```
*)

PROCEDURE setenv (name: ADDRESS; value: ADDRESS; overwrite: INTEGER) : INTEGER ;


(*
   srand - initialize the random number seed.
*)

PROCEDURE srand (seed: INTEGER) ;


(*
   rand - return a random integer.
*)

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.
*)

PROCEDURE time (a: ADDRESS) : time_t ;


(*
   localtime - returns a pointer to the libc copy of the tm
               structure.
*)

PROCEDURE localtime (VAR t: time_t) : ADDRESS ;


(*
   ftime - return date and time.
*)

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.
*)

PROCEDURE shutdown (s: INTEGER; how: INTEGER) : INTEGER ;


(*
   rename - change the name or location of a file
*)

PROCEDURE rename (oldpath, newpath: ADDRESS) : INTEGER ;


(*
   setjmp - returns 0 if returning directly, and non-zero
            when returning from longjmp using the saved
            context.
*)

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
             val must not be zero.
*)

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


(*
   atexit - execute, proc, when the function exit is called.
*)

PROCEDURE atexit (proc: PROC) ;


(*
   ttyname - returns a pointer to a string determining the ttyname.
*)

PROCEDURE ttyname (filedes: INTEGER) : ADDRESS ;
```

```
END libc.
```

4.1.50 gm2-libs/PushBackInput

```
DEFINITION MODULE PushBackInput ;

(*
    Description: provides a method for pushing back and consuming input
                 from a standard file descriptor. Insipred by software
                 tools.
*)

FROM FIO IMPORT File ;
FROM DynamicStrings IMPORT String ;

EXPORT QUALIFIED Open, PutCh, GetCh, Error, WarnError, WarnString,
                 Close, SetDebug, GetExitStatus,
                 PutString, GetColumnPosition, GetCurrentLine ;


(*
   Open - opens a file for reading.
*)

PROCEDURE Open (a: ARRAY OF CHAR) : File ;


(*
   GetCh - gets a character from either the push back stack or
           from file, f.
*)

PROCEDURE GetCh (f: File) : CHAR ;


(*
   PutCh - pushes a character onto the push back stack, it also
           returns the character which has been pushed.
*)

PROCEDURE PutCh (f: File; ch: CHAR) : CHAR ;


(*
   PutString - pushes a string onto the push back stack.
*)

PROCEDURE PutString (f: File; a: ARRAY OF CHAR) ;
```

```
(*
   Error - emits an error message with the appropriate file, line combination.
*)

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 o
               1 will be issued.
*)

PROCEDURE WarnError (a: ARRAY OF CHAR) ;


(*
   WarnString - emits an error message with the appropriate file, line combination.
                It does not terminate but when the program finishes an exit status
                1 will be issued.
*)

PROCEDURE WarnString (s: String) ;


(*
   Close - closes the opened file.
*)

PROCEDURE Close (f: File) ;


(*
   GetExitStatus - returns the exit status which will be 1 if any warnings were iss
*)

PROCEDURE GetExitStatus () : CARDINAL ;


(*
   SetDebug - sets the debug flag on or off.
*)

PROCEDURE SetDebug (d: BOOLEAN) ;
```

```
(*
   GetColumnPosition - returns the column position of the current character.
*)

PROCEDURE GetColumnPosition () : CARDINAL ;


(*
   GetCurrentLine - returns the current line number.
*)

PROCEDURE GetCurrentLine () : CARDINAL ;


END PushBackInput.
```

4.1.51 gm2-libs/libm

```
DEFINITION MODULE FOR "C" libm ;

(*
    Description: provides access to libm. Users are strongly advised to
                 use MathLib0 or RealMath as call 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,
                   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 ;

PROCEDURE sin (x: REAL) : REAL ;
PROCEDURE sinl (x: LONGREAL) : LONGREAL ;
PROCEDURE sinf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE cos (x: REAL) : REAL ;
PROCEDURE cosl (x: LONGREAL) : LONGREAL ;
PROCEDURE cosf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE tan (x: REAL) : REAL ;
PROCEDURE tanl (x: LONGREAL) : LONGREAL ;
PROCEDURE tanf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE sqrt (x: REAL) : REAL ;
PROCEDURE sqrtl (x: LONGREAL) : LONGREAL ;
PROCEDURE sqrtf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE asin (x: REAL) : REAL ;
PROCEDURE asinl (x: LONGREAL) : LONGREAL ;
PROCEDURE asinf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE acos (x: REAL) : REAL ;
PROCEDURE acosl (x: LONGREAL) : LONGREAL ;
PROCEDURE acosf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE atan (x: REAL) : REAL ;
PROCEDURE atanl (x: LONGREAL) : LONGREAL ;
```

```
PROCEDURE atanf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE atan2 (x, y: REAL) : REAL ;
PROCEDURE atan2l (x, y: LONGREAL) : LONGREAL ;
PROCEDURE atan2f (x, y: SHORTREAL) : SHORTREAL ;
PROCEDURE exp (x: REAL) : REAL ;
PROCEDURE expl (x: LONGREAL) : LONGREAL ;
PROCEDURE expf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE log (x: REAL) : REAL ;
PROCEDURE logl (x: LONGREAL) : LONGREAL ;
PROCEDURE logf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE exp10 (x: REAL) : REAL ;
PROCEDURE exp10l (x: LONGREAL) : LONGREAL ;
PROCEDURE exp10f (x: SHORTREAL) : SHORTREAL ;
PROCEDURE pow (x, y: REAL) : REAL ;
PROCEDURE powl (x, y: LONGREAL) : LONGREAL ;
PROCEDURE powf (x, y: SHORTREAL) : SHORTREAL ;
PROCEDURE floor (x: REAL) : REAL ;
PROCEDURE floorl (x: LONGREAL) : LONGREAL ;
PROCEDURE floorf (x: SHORTREAL) : SHORTREAL ;
PROCEDURE ceil (x: REAL) : REAL ;
PROCEDURE ceill (x: LONGREAL) : LONGREAL ;
PROCEDURE ceilf (x: SHORTREAL) : SHORTREAL ;

END libm.
```

4.1.52 gm2-libs/SEnvironment

```
DEFINITION MODULE SEnvironment ;

(*
    Description: provides access to the environment settings of a process.
*)

FROM DynamicStrings IMPORT String ;
EXPORT QUALIFIED GetEnvironment ;


(*
   GetEnvironment - gets the environment variable, env, and places
                    a copy of its value into String, s.
                    TRUE is returned if successful.
*)

PROCEDURE GetEnvironment (env: String; VAR s: String) : BOOLEAN ;


END SEnvironment.
```

4.2 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.

4.2.1 gm2-libs-pim/FileSystem

```
DEFINITION MODULE FileSystem ;

(*
    Description: provides GNU Modula-2 with a PIM [234] FileSystem
                 compatible module. 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,
```

```
                WriteWord, WriteChar, WriteByte, WriteNBytes ;

TYPE
   File = RECORD
             res     : Response ;
             flags   : FlagSet ;
             eof     : BOOLEAN ;
             lastWord: WORD ;
             lastByte: BYTE ;
             fio     : FIO.File ;
             highpos,
             lowpos  : CARDINAL ;
             name    : String ;
          END ;

   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 = SET OF Flag;

   Response = (done, notdone, notsupported, callerror,
               unknownfile, paramerror, toomanyfiles,
               userdeverror) ;

   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.
*)

PROCEDURE Create (VAR f: File) ;


(*
   Close - closes an open file.
*)
```

```
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.
*)

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.
*)

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.
*)

PROCEDURE Delete (name: ARRAY OF CHAR; VAR f: File) ;


(*
   ReadWord - reads a WORD, w, from file, f.
              f.res is set appropriately.
*)

PROCEDURE ReadWord (VAR f: File; VAR w: WORD) ;


(*
   WriteWord - writes one word to a file, f.
               f.res is set appropriately.
*)

PROCEDURE WriteWord (VAR f: File; w: WORD) ;
```

```
(*
   ReadChar - reads one character from a file, f.
*)

PROCEDURE ReadChar (VAR f: File; VAR ch: CHAR) ;


(*
   WriteChar - writes a character, ch, to a file, f.
               f.res is set appropriately.
*)

PROCEDURE WriteChar (VAR f: File; ch: CHAR) ;


(*
   ReadByte - reads a BYTE, b, from file, f.
              f.res is set appropriately.
*)

PROCEDURE ReadByte (VAR f: File; VAR b: BYTE) ;


(*
   WriteByte - writes one BYTE, b, to a file, f.
               f.res is set appropriately.
*)

PROCEDURE WriteByte (VAR f: File; b: BYTE) ;


(*
   ReadNBytes - reads a sequence of bytes from a file, f.
*)

PROCEDURE ReadNBytes (VAR f: File; a: ADDRESS; amount: CARDINAL;
                      VAR actuallyRead: CARDINAL) ;


(*
   WriteNBytes - writes a sequence of bytes to file, f.
*)

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.
*)

PROCEDURE Again (VAR f: File) ;


(*
   SetRead - puts the file, f, into the read state.
             The file position is unchanged.
*)

PROCEDURE SetRead (VAR f: File) ;


(*
   SetWrite - puts the file, f, into the write state.
              The file position is unchanged.
*)

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.
*)

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.
*)

PROCEDURE SetOpen (VAR f: File) ;


(*
   Reset - places a file, f, into the open state and reset the
           position to the start of the file.
```

```
*)

PROCEDURE Reset (VAR f: File) ;


(*
   SetPos - lseek to a position within a file.
*)

PROCEDURE SetPos (VAR f: File; high, low: CARDINAL) ;


(*
   GetPos - return the position within a file.
*)

PROCEDURE GetPos (VAR f: File; VAR high, low: CARDINAL) ;


(*
   Length - returns the length of file, in, high, and, low.
*)

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.
*)

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.
*)

PROCEDURE FileNameChar (ch: CHAR) ;


END FileSystem.
```

4.2.2 gm2-libs-pim/BitByteOps

```
DEFINITION MODULE BitByteOps ;

(*
    Description: provides a Logitech-3.0 compatible library for GNU Modula-2.
*)

FROM SYSTEM IMPORT BYTE ;


(*
   GetBits - returns the bits firstBit..lastBit from source.
             Bit 0 of byte maps onto the firstBit of source.
*)

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.
*)

PROCEDURE SetBits (VAR byte: BYTE; firstBit, lastBit: CARDINAL;
                   pattern: BYTE) ;


(*
   ByteAnd - returns a bitwise (left AND right)
*)

PROCEDURE ByteAnd (left, right: BYTE) : BYTE ;


(*
   ByteOr - returns a bitwise (left OR right)
*)

PROCEDURE ByteOr (left, right: BYTE) : BYTE ;


(*
   ByteXor - returns a bitwise (left XOR right)
*)
```

```
PROCEDURE ByteXor (left, right: BYTE) : BYTE ;


(*
   ByteNot - returns a byte with all bits inverted.
*)

PROCEDURE ByteNot (byte: BYTE) : BYTE ;


(*
   ByteShr - returns a, byte, which has been shifted, count
             bits to the right.
*)

PROCEDURE ByteShr (byte: BYTE; count: CARDINAL) : BYTE ;


(*
   ByteShl - returns a, byte, which has been shifted, count
             bits to the left.
*)

PROCEDURE ByteShl (byte: BYTE; count: CARDINAL) : BYTE ;


(*
   ByteSar - shift byte arthemetic right.  Preserves the top
             end bit and as the value is shifted right.
*)

PROCEDURE ByteSar (byte: BYTE; count: CARDINAL) : BYTE ;


(*
   ByteRor - returns a, byte, which has been rotated, count
             bits to the right.
*)

PROCEDURE ByteRor (byte: BYTE; count: CARDINAL) : BYTE ;


(*
   ByteRol - returns a, byte, which has been rotated, count
             bits to the left.
*)
```

```
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.
*)

PROCEDURE HighNibble (byte: BYTE) : BYTE ;


(*
   LowNibble - returns the low nibble only from, byte.
               The top nibble is replaced by zeros.
*)

PROCEDURE LowNibble (byte: BYTE) : BYTE ;


(*
   Swap - swaps the low and high nibbles in the, byte.
*)

PROCEDURE Swap (byte: BYTE) : BYTE ;


END BitByteOps.
```

4.2.3 gm2-libs-pim/Terminal

```
DEFINITION MODULE Terminal ;

(*
    Description: provides a Logitech 3.0 compatible and PIM [234] compatible
                 Terminal module. 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.
*)

PROCEDURE Read (VAR ch: CHAR) ;


(*
   KeyPressed - returns TRUE if a character can be read without blocking
                the caller.
*)

PROCEDURE KeyPressed () : BOOLEAN ;


(*
   ReadString - reads a sequence of characters.
                Tabs are expanded into 8 spaces and <cr> or <lf> terminates
                the string.
*)

PROCEDURE ReadString (VAR s: ARRAY OF CHAR) ;


(*
   ReadAgain - makes the last character readable again.
*)

PROCEDURE ReadAgain ;


(*
   Write - writes a single character to the Termbase module.
```

```
*)

PROCEDURE Write (ch: CHAR) ;


(*
   WriteString - writes out a string which is terminated by a <nul>
                 character or the end of string HIGH(s).
*)

PROCEDURE WriteString (s: ARRAY OF CHAR) ;


(*
   WriteLn - writes a lf character.
*)

PROCEDURE WriteLn ;


END Terminal.
```

4.2.4 gm2-libs-pim/Random

```
DEFINITION MODULE Random ;

(*
    Description: provides a Logitech-3.0 compatible library
*)

FROM SYSTEM IMPORT BYTE ;
EXPORT QUALIFIED Randomize, RandomInit, RandomCard, RandomInt, RandomReal ;


(*
   Randomize - initialize the random number generator with a seed
               based on the microseconds.
*)

PROCEDURE Randomize ;


(*
   RandomInit - initialize the random number generator with value, seed.
*)

PROCEDURE RandomInit (seed: CARDINAL) ;


(*
   RandomBytes - fills in an array with random values.
*)

PROCEDURE RandomBytes (a: ARRAY OF BYTE) ;


(*
   RandomInt - return an INTEGER in the range 0..bound-1
*)

PROCEDURE RandomInt (bound: INTEGER) : INTEGER ;


(*
   RandomCard - return a CARDINAL in the range 0..bound-1
*)

PROCEDURE RandomCard (bound: CARDINAL) : CARDINAL ;
```

```
(*
   RandomReal - return a REAL number in the range 0.0..1.0
*)

PROCEDURE RandomReal () : REAL ;


(*
   RandomLongReal - return a LONGREAL number in the range 0.0..1.0
*)

PROCEDURE RandomLongReal () : LONGREAL ;


END Random.
```

4.2.5 gm2-libs-pim/InOut

```
DEFINITION MODULE InOut ;

(*
    Description: provides a compatible PIM [234] InOut module.
*)

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 = ASCII.EOL ;

VAR
   Done  : BOOLEAN ;
   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.
*)

PROCEDURE OpenInput (defext: ARRAY OF CHAR) ;


(*
   CloseInput - closes an opened input file and returns input back to
                StdIn.
*)

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.
*)

PROCEDURE OpenOutput (defext: ARRAY OF CHAR) ;


(*
   CloseOutput - closes an opened output file and returns output back to
                 StdOut.
*)

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.
*)

PROCEDURE Read (VAR ch: CHAR) ;


(*
   ReadString - reads a sequence of characters. Leading white space
                is ignored and the string is terminated with a character
                <= ' '
*)

PROCEDURE ReadString (VAR s: ARRAY OF CHAR) ;


(*
   WriteString - writes a string to the output file.
*)

PROCEDURE WriteString (s: ARRAY OF CHAR) ;


(*
   Write - writes out a single character, ch, to the current output file.
*)

PROCEDURE Write (ch: CHAR) ;


(*
```

```
   WriteLn - writes a newline to the output file.
*)

PROCEDURE WriteLn ;


(*
   ReadInt - reads a string and converts it into an INTEGER, x.
             Done is set if an INTEGER is read.
*)

PROCEDURE ReadInt (VAR x: INTEGER) ;


(*
   ReadInt - reads a string and converts it into an INTEGER, x.
             Done is set if an INTEGER is read.
*)

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.
*)

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.
*)

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.
*)

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.
*)

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 <= ' '.
*)

PROCEDURE ReadS () : String ;


(*
   WriteS - writes a String to the output device.
            It returns the string, s.
*)

PROCEDURE WriteS (s: String) : String ;


END InOut.
```

4.2.6 gm2-libs-pim/BlockOps

```
DEFINITION MODULE BlockOps ;

(*
    Description: provides a Logitech compatible module for moving blocks
                 of memory.
*)

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.
*)

PROCEDURE BlockMoveForward (dest, src: ADDRESS; n: CARDINAL) ;


(*
   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.
*)

PROCEDURE BlockMoveBackward (dest, src: ADDRESS; n: CARDINAL) ;


(*
   BlockClear - fills, block..block+n-1, with zero's.
*)

PROCEDURE BlockClear (block: ADDRESS; n: CARDINAL) ;


(*
   BlockSet - fills, n, bytes starting at, block, with a pattern
              defined at address pattern..pattern+patternSize-1.
*)

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.
*)

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.
*)

PROCEDURE BlockPosition (block: ADDRESS; blockSize: CARDINAL;
                         pattern: ADDRESS; patternSize: CARDINAL) : CARDINAL ;


END BlockOps.
```

4.2.7 gm2-libs-pim/TimeDate

```
DEFINITION MODULE TimeDate ;

(*
    Description: provides a Logitech-3.0 compatible library module.
                 Advised to use cleaner designed modules based on 'man 3 strtime'
                 and friends for new projects as the day value here is ugly
                 [maybe it mapped onto MSDOS].
*)

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
   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.
*)

PROCEDURE GetTime (VAR curTime: Time) ;


(*
   SetTime - does nothing, but provides compatibility with
             the Logitech-3.0 library.
*)

PROCEDURE SetTime (curTime: Time) ;


(*
   CompareTime - compare two dates and time which returns:
```

```
                  -1  if t1 < t2
                   0  if t1 = t2
                   1  if t1 > t2
*)

PROCEDURE CompareTime (t1, t2: Time) : INTEGER ;


(*
   TimeToZero - initializes, t, to zero.
*)

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
*)

PROCEDURE TimeToString (t: Time; VAR s: ARRAY OF CHAR) ;


END TimeDate.
```

4.2.8 gm2-libs-pim/Conversions

```
DEFINITION MODULE Conversions ;

(*
    Description: provides a Logitech-3.0 compatible library.
*)

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.
*)

PROCEDURE ConvertOctal    (num, len: CARDINAL; VAR str: ARRAY OF CHAR) ;
PROCEDURE ConvertHex      (num, len: CARDINAL; VAR str: ARRAY OF CHAR) ;
PROCEDURE ConvertCardinal (num, len: CARDINAL; VAR str: ARRAY OF CHAR) ;

(*
   The INTEGER counterparts will add a '-' if, num, is <0
*)

PROCEDURE ConvertInteger  (num: INTEGER; len: CARDINAL; VAR str: ARRAY OF CHAR) ;
PROCEDURE ConvertLongInt  (num: LONGINT; len: CARDINAL; VAR str: ARRAY OF CHAR) ;
PROCEDURE ConvertShortInt (num: SHORTINT; len: CARDINAL; VAR str: ARRAY OF CHAR) ;


END Conversions.
```

4.2.9 gm2-libs-pim/Display

```
DEFINITION MODULE Display ;

(*
    Description: provides a Logitech 3.0 compatible Display module.
*)

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.
*)

PROCEDURE Write (ch: CHAR) ;


END Display.
```

4.2.10 gm2-libs-pim/Break

```
DEFINITION MODULE Break ;

(*
    Description: provides a Logitech compatible Break handler module
                 which catches ctrl-c.
*)

EXPORT QUALIFIED EnableBreak, DisableBreak, InstallBreak, UnInstallBreak ;


(*
   EnableBreak - enable the current break handler.
*)

PROCEDURE EnableBreak ;


(*
   DisableBreak - disable the current break handler (and all
                  installed handlers).
*)

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.
*)

PROCEDURE InstallBreak (p: PROC) ;


(*
   UnInstallBreak - pops the break handler stack.
*)

PROCEDURE UnInstallBreak ;


END Break.
```

4.2.11 gm2-libs-pim/ErrorCode

```
DEFINITION MODULE ErrorCode ;

(*
    Description: provides a Logitech-3.0 compatible module which
                 handles exiting from an application with an exit
                 value.
*)

EXPORT QUALIFIED SetErrorCode, GetErrorCode, ExitToOS ;


(*
   SetErrorCode - sets the exit value which will be used if
                  the application terminates normally.
*)

PROCEDURE SetErrorCode (value: INTEGER) ;


(*
   GetErrorCode - returns the current value to be used upon
                  application termination.
*)

PROCEDURE GetErrorCode (VAR value: INTEGER) ;


(*
   ExitToOS - terminate the application and exit returning
              the last value set by SetErrorCode to the OS.
*)

PROCEDURE ExitToOS ;


END ErrorCode.
```

4.2.12 gm2-libs-pim/RealConversions

```
DEFINITION MODULE RealConversions ;

(*
    Description: provides a Logitech-3.0 compatible module.
*)

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.
*)

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.
*)

PROCEDURE RealToString (r: REAL; digits, width: INTEGER;
                        VAR str: ARRAY OF CHAR; VAR ok: BOOLEAN) ;


(*
   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

                      if r is a NaN then the string 'nan' is returned formatted and
                      ok will be FALSE.
*)
```

```
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.
*)

PROCEDURE StringToReal (str: ARRAY OF CHAR; VAR r: REAL; VAR ok: BOOLEAN) ;


(*
   StringToLongReal - converts, str, into a LONGREAL, r. The parameter, ok, is
                      set to TRUE if the conversion was successful.
*)

PROCEDURE StringToLongReal (str: ARRAY OF CHAR; VAR r: LONGREAL; VAR ok: BOOLEAN) ;


END RealConversions.
```

4.2.13 gm2-libs-pim/CardinalIO

```
DEFINITION MODULE CardinalIO ;

(*
    Description: provides a PIM and Logitech compatible module.
*)

EXPORT QUALIFIED Done,
                 ReadCardinal, WriteCardinal, ReadHex, WriteHex,
                 ReadLongCardinal, WriteLongCardinal, ReadLongHex,
                 WriteLongHex,
                 ReadShortCardinal, WriteShortCardinal, ReadShortHex,
                 WriteShortHex ;


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.
*)

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.
*)

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.
*)

PROCEDURE ReadHex (VAR c: CARDINAL) ;
```

```
(*
   WriteHex - writes out a CARDINAL, c, in hexadecimal format padding
              with, n, characters (leading with '0')
*)

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.
*)

PROCEDURE ReadLongCardinal (VAR c: LONGCARD) ;


(*
   WriteLongCardinal - 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.
*)

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.
*)

PROCEDURE ReadLongHex (VAR c: LONGCARD) ;


(*
   WriteLongHex - writes out a LONGCARD, c, in hexadecimal format padding
                  with, n, characters (leading with '0')
*)

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.
```

```
*)

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.
*)

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.
*)

PROCEDURE ReadShortHex (VAR c: SHORTCARD) ;


(*
   WriteShortHex - writes out a SHORTCARD, c, in hexadecimal format padding
                   with, n, characters (leading with '0')
*)

PROCEDURE WriteShortHex (c: SHORTCARD; n: CARDINAL) ;


END CardinalIO.
```

4.2.14 gm2-libs-pim/LongIO

```
DEFINITION MODULE LongIO ;

(*
    Description: provides a Logitech-3.0 compatible library for GNU Modula-2.
*)

EXPORT QUALIFIED Done, ReadLongInt, WriteLongInt ;

VAR
   Done: BOOLEAN ;

PROCEDURE ReadLongInt (VAR i: LONGINT) ;
PROCEDURE WriteLongInt (i: LONGINT; n: CARDINAL) ;


END LongIO.
```

4.2.15 gm2-libs-pim/DebugPMD

```
DEFINITION MODULE DebugPMD ;

END DebugPMD.
```

4.2.16 gm2-libs-pim/Delay

```
DEFINITION MODULE Delay ;

(*
    Description: provides a Logitech-3.0 compatible module for
                 GNU Modula-2.
*)

EXPORT QUALIFIED Delay ;


(*
   milliSec - delays the program by approximately, milliSec, milliseconds.
*)

PROCEDURE Delay (milliSec: INTEGER) ;


END Delay.
```

4.2.17 gm2-libs-pim/Strings

```
DEFINITION MODULE Strings ;

(*
    Description: provides a Logitech-3.0 compatible library
*)

EXPORT QUALIFIED Assign, Insert, Delete, Pos, Copy, ConCat, Length,
                 CompareStr ;

(*
   Assign - dest := source.
*)

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)
*)

PROCEDURE Insert (substr: ARRAY OF CHAR; VAR str: ARRAY OF CHAR;
                  index: CARDINAL) ;


(*
   Delete - delete len characters from, str, starting at, index.
*)

PROCEDURE Delete (VAR str: ARRAY OF CHAR; index: CARDINAL; length: CARDINAL) ;


(*
   Pos - return the first position of, substr, in, str.
*)

PROCEDURE Pos (substr, str: ARRAY OF CHAR) : CARDINAL ;


(*
   Copy - copy at most, length, characters in, substr, to, str,
          starting at position, index.
*)

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.
*)

PROCEDURE ConCat (s1, s2: ARRAY OF CHAR; VAR dest: ARRAY OF CHAR) ;


(*
   Length - return the length of string, s.
*)

PROCEDURE Length (s: ARRAY OF CHAR) : CARDINAL ;


(*
   CompareStr - compare two strings, left, and, right.
*)

PROCEDURE CompareStr (left, right: ARRAY OF CHAR) : INTEGER ;


END Strings.
```

4.2.18 gm2-libs-pim/DebugTrace

```
DEFINITION MODULE DebugTrace ;

(*
    Description: provides a compatible module for the
                 Logitech-3.0 PIM Modula-2 compiler.
                 It does nothing other satisfy an import for
                 legacy source code.
*)

END DebugTrace.
```

4.2.19 gm2-libs-pim/NumberConversion

```
DEFINITION MODULE NumberConversion ;

(*
    Description: provides a Logitech compatible NumberConversion library.
*)



END NumberConversion.
```

4.2.20 gm2-libs-pim/FloatingUtilities

```
DEFINITION MODULE FloatingUtilities ;

(*
    Description: provides a Logitech-3.0 compatible library
*)

EXPORT QUALIFIED Frac, Round, Float, Trunc,
                 Fracl, Roundl, Floatl, Truncl ;


(*
   Frac - returns the fractional component of, r.
*)

PROCEDURE Frac (r: REAL) : REAL ;


(*
   Int - returns the integer part of r. It rounds the value towards zero.
*)

PROCEDURE Int (r: REAL) : INTEGER ;


(*
   Round - returns the number rounded to the nearest integer.
*)

PROCEDURE Round (r: REAL) : INTEGER ;


(*
   Float - returns a REAL value corresponding to, i.
*)

PROCEDURE Float (i: INTEGER) : REAL ;


(*
   Trunc - round to the nearest integer not larger in absolute
           value.
*)

PROCEDURE Trunc (r: REAL) : INTEGER ;
```

```
(*
   Fracl - returns the fractional component of, r.
*)

PROCEDURE Fracl (r: LONGREAL) : LONGREAL ;


(*
   Intl - returns the integer part of r. It rounds the value towards zero.
*)

PROCEDURE Intl (r: LONGREAL) : LONGINT ;


(*
   Roundl - returns the number rounded to the nearest integer.
*)

PROCEDURE Roundl (r: LONGREAL) : LONGINT ;


(*
   Floatl - returns a REAL value corresponding to, i.
*)

PROCEDURE Floatl (i: INTEGER) : LONGREAL ;


(*
   Truncl - round to the nearest integer not larger in absolute
            value.
*)

PROCEDURE Truncl (r: LONGREAL) : LONGINT ;


END FloatingUtilities.
```

4.2.21 gm2-libs-pim/Keyboard

```
DEFINITION MODULE Keyboard ;

(*
    Description: provides compatibility with Logitech 3.0 Keyboard module.
*)

EXPORT QUALIFIED Read, KeyPressed ;


(*
   Read - reads a character from StdIn. If necessary it will wait
          for a key to become present on StdIn.
*)

PROCEDURE Read (VAR ch: CHAR) ;


(*
   KeyPressed - returns TRUE if a character can be read from StdIn
                without blocking the caller.
*)

PROCEDURE KeyPressed () : BOOLEAN ;


END Keyboard.
```

4.2.22 gm2-libs-pim/BitWordOps

```
DEFINITION MODULE BitWordOps ;

(*
    Description: provides a Logitech-3.0 compatible library for GNU Modula-2.
*)

FROM SYSTEM IMPORT WORD ;


(*
   GetBits - returns the bits firstBit..lastBit from source.
             Bit 0 of word maps onto the firstBit of source.
*)

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.
*)

PROCEDURE SetBits (VAR word: WORD; firstBit, lastBit: CARDINAL;
                   pattern: WORD) ;


(*
   WordAnd - returns a bitwise (left AND right)
*)

PROCEDURE WordAnd (left, right: WORD) : WORD ;


(*
   WordOr - returns a bitwise (left OR right)
*)

PROCEDURE WordOr (left, right: WORD) : WORD ;


(*
   WordXor - returns a bitwise (left XOR right)
*)
```

```
PROCEDURE WordXor (left, right: WORD) : WORD ;


(*
   WordNot - returns a word with all bits inverted.
*)

PROCEDURE WordNot (word: WORD) : WORD ;


(*
   WordShr - returns a, word, which has been shifted, count
             bits to the right.
*)

PROCEDURE WordShr (word: WORD; count: CARDINAL) : WORD ;


(*
   WordShl - returns a, word, which has been shifted, count
             bits to the left.
*)

PROCEDURE WordShl (word: WORD; count: CARDINAL) : WORD ;


(*
   WordSar - shift word arthemetic right.  Preserves the top
             end bit and as the value is shifted right.
*)

PROCEDURE WordSar (word: WORD; count: CARDINAL) : WORD ;


(*
   WordRor - returns a, word, which has been rotated, count
             bits to the right.
*)

PROCEDURE WordRor (word: WORD; count: CARDINAL) : WORD ;


(*
   WordRol - returns a, word, which has been rotated, count
             bits to the left.
*)
```

```
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.
*)

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.
*)

PROCEDURE LowByte (word: WORD) : WORD ;


(*
   Swap - byte flips the contents of word.
*)

PROCEDURE Swap (word: WORD) : WORD ;


END BitWordOps.
```

4.2.23 gm2-libs-pim/Termbase

```
DEFINITION MODULE Termbase ;

(*
    Description: provides GNU Modula-2 with a PIM 234 compatible Termbase
                 module. Definition module complies with Logitech 3.0.
                 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 = PROCEDURE (VAR CHAR) ;
   WriteProcedure = PROCEDURE (CHAR) ;
   StatusProcedure = PROCEDURE () : BOOLEAN ;


(*
   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.
*)

PROCEDURE AssignRead (rp: ReadProcedure; sp: StatusProcedure;
                      VAR Done: BOOLEAN) ;


(*
   UnAssignRead - undo the last call to AssignRead and set Done to TRUE
                  on success.
*)

PROCEDURE UnAssignRead (VAR Done: BOOLEAN) ;


(*
   Read - reads a single character using the currently active read
          procedure.
*)
```

```
PROCEDURE Read (VAR ch: CHAR) ;


(*
   KeyPressed - returns TRUE if a character is available to be read.
*)

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.
*)

PROCEDURE AssignWrite (wp: WriteProcedure; VAR Done: BOOLEAN) ;


(*
   UnAssignWrite - undo the last call to AssignWrite and set Done to TRUE
                   on success.
*)

PROCEDURE UnAssignWrite (VAR Done: BOOLEAN) ;


(*
   Write - writes a single character using the currently active write
           procedure.
*)

PROCEDURE Write (VAR ch: CHAR) ;


END Termbase.
```

4.2.24 gm2-libs-pim/BitBlockOps

```
DEFINITION MODULE BitBlockOps ;

(*
    Description: provides a Logitech compatible module.
*)


FROM SYSTEM IMPORT ADDRESS ;


(*
   BlockAnd - performs a bitwise AND on blocks
              [dest..dest+size-1] := [dest..dest+size-1] AND
                                     [src..src+size-1]
*)

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]
*)

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]
*)

PROCEDURE BlockXor (dest, src: ADDRESS; size: CARDINAL) ;


(*
   BlockNot - performs a bitsize NOT on the block as defined
              by:  [dest..dest+size-1]
*)

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.
*)

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
              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.
*)

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.
*)

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.
*)

PROCEDURE BlockRol (dest: ADDRESS; size, count: CARDINAL) ;


END BitBlockOps.
```

4.2.25 gm2-libs-pim/RealInOut

```
DEFINITION MODULE RealInOut ;

(*
    Description: provides a compatible RealInOut PIM 234 module.
*)

EXPORT QUALIFIED SetNoOfDecimalPlaces,
                 ReadReal, WriteReal, WriteRealOct,
                 ReadLongReal, WriteLongReal, WriteLongRealOct,
                 ReadShortReal, WriteShortReal, WriteShortRealOct,
                 Done ;

CONST
   DefaultDecimalPlaces = 6 ;

VAR
   Done: BOOLEAN ;


(*
   SetNoOfDecimalPlaces - number of decimal places WriteReal and
                          WriteLongReal should emit.  This procedure
                          can be used to override the default
                          DefaultDecimalPlaces constant.
*)

PROCEDURE SetNoOfDecimalPlaces (places: CARDINAL) ;


(*
   ReadReal - reads a real number, legal syntaxes include:
              100, 100.0, 100e0, 100E0, 100E-1, E2, +1E+2, 1e+2
*)

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.
*)

PROCEDURE WriteReal (x: REAL; n: CARDINAL) ;
```

```
(*
   WriteRealOct - writes the real to terminal in octal words.
*)

PROCEDURE WriteRealOct (x: REAL) ;


(*
   ReadLongReal - reads a LONGREAL number, legal syntaxes include:
                  100, 100.0, 100e0, 100E0, 100E-1, E2, +1E+2, 1e+2
*)

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.
*)

PROCEDURE WriteLongReal (x: LONGREAL; n: CARDINAL) ;


(*
   WriteLongRealOct - writes the LONGREAL to terminal in octal words.
*)

PROCEDURE WriteLongRealOct (x: LONGREAL) ;


(*
   ReadShortReal - reads a SHORTREAL number, legal syntaxes include:
                   100, 100.0, 100e0, 100E0, 100E-1, E2, +1E+2, 1e+2
*)

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.
*)

PROCEDURE WriteShortReal (x: SHORTREAL; n: CARDINAL) ;
```

```
(*
   WriteShortRealOct - writes the SHORTREAL to terminal in octal words.
*)

PROCEDURE WriteShortRealOct (x: SHORTREAL) ;


END RealInOut.
```

4.3 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`).

4.3.1 gm2-libs-coroutines/KeyBoardLEDs

```
DEFINITION MODULE KeyBoardLEDs ;

(*
    Description: provides a simple module to manipulate the keyboard
                 LEDs in Linux.
*)

EXPORT QUALIFIED SwitchLeds,
                 SwitchScroll, SwitchNum, SwitchCaps ;


(*
   SwitchLeds - switch the keyboard LEDs to the state defined
                by the BOOLEAN variables. TRUE = ON.
*)
```

```
PROCEDURE SwitchLeds (NumLock, CapsLock, ScrollLock: BOOLEAN) ;


(*
   SwitchScroll - switchs the scroll LED on or off.
*)

PROCEDURE SwitchScroll (Scroll: BOOLEAN) ;


(*
   SwitchNum - switches the Num LED on or off.
*)

PROCEDURE SwitchNum (Num: BOOLEAN) ;


(*
   SwitchCaps - switches the Caps LED on or off.
*)

PROCEDURE SwitchCaps (Caps: BOOLEAN) ;


END KeyBoardLEDs.
```

4.3.2 gm2-libs-coroutines/Executive

```
DEFINITION MODULE Executive ;

(*
    Description: provides a simple multitasking executive.
*)

EXPORT QUALIFIED SEMAPHORE, DESCRIPTOR,
                 InitProcess, KillProcess, Resume, Suspend, InitSemaphore,
                 Wait, Signal, WaitForIO, Ps, GetCurrentProcess,
                 RotateRunQueue, ProcessName, DebugProcess ;

TYPE
   SEMAPHORE ;         (* defines dijkstra's semaphores *)
   DESCRIPTOR ;        (* handle onto a process         *)


(*
   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.
*)

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.
*)

PROCEDURE KillProcess ;


(*
   Resume - resumes a suspended process. If all is successful then the process, p,
            is returned. If it fails then NIL is returned.
*)

PROCEDURE Resume (d: DESCRIPTOR) : DESCRIPTOR ;
```

```
(*
   Suspend - suspend the calling process.
             The process can only continue running if another process
             Resumes it.
*)

PROCEDURE Suspend ;


(*
   InitSemaphore - creates a semaphore whose initial value is, v, and
                   whose name is, Name.
*)

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
          and then it will decrement this value.
*)

PROCEDURE Wait (s: SEMAPHORE) ;


(*
   Signal - performs dijkstra's V operation on a semaphore.
            A process which calls the procedure will increment
            the semaphores value.
*)

PROCEDURE Signal (s: SEMAPHORE) ;


(*
   WaitForIO - waits for an interrupt to occur on vector, VectorNo.
*)

PROCEDURE WaitForIO (VectorNo: CARDINAL) ;


(*
   Ps - displays a process list together with process status.
```

```
*)

PROCEDURE Ps ;


(*
   GetCurrentProcess - returns the descriptor of the current running
                       process.
*)

PROCEDURE GetCurrentProcess () : DESCRIPTOR ;


(*
   RotateRunQueue - rotates the process run queue.
                    It does not call the scheduler.
*)

PROCEDURE RotateRunQueue ;


(*
   ProcessName - displays the name of process, d, through
                 DebugString.
*)

PROCEDURE ProcessName (d: DESCRIPTOR) ;


(*
   DebugProcess - gdb debug handle to enable users to debug deadlocked
                  semaphore processes.
*)

PROCEDURE DebugProcess (d: DESCRIPTOR) ;


END Executive.
```

4.3.3 gm2-libs-coroutines/Debug

```
DEFINITION MODULE Debug ;

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

EXPORT QUALIFIED Halt, DebugString, PushOutput ;

TYPE
   WriteP = PROCEDURE (CHAR) ;


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

          It then terminates by calling 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.
*)

PROCEDURE DebugString (a: ARRAY OF CHAR) ;


(*
   PushOutput - pushes the output procedure, p, which is used Debug.
*)

PROCEDURE PushOutput (p: WriteP) ;


(*
   PopOutput - pops the current output procedure from the stack.
*)

PROCEDURE PopOutput ;
```

```
END Debug.
```

4.3.4 gm2-libs-coroutines/TimerHandler

```
DEFINITION MODULE TimerHandler ;

(*
    Description: provides a simple timer handler for the
                 Executive.
                 It also provides the Executive with a basic
                 round robin scheduler.
*)

EXPORT QUALIFIED TicksPerSecond, GetTicks,
                 EVENT,
                 Sleep, ArmEvent, WaitOn, Cancel, ReArmEvent ;


CONST
   TicksPerSecond =   25 ;  (* Number of ticks per second.         *)

TYPE
   EVENT ;


(*
   GetTicks - returns the number of ticks since boottime.
*)

PROCEDURE GetTicks () : CARDINAL ;


(*
   Sleep - suspends the current process for a time, t.
           The time is measured in ticks.
*)

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.
*)

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.
*)

PROCEDURE WaitOn (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.
*)

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.
*)

PROCEDURE ReArmEvent (e: EVENT; t: CARDINAL) : BOOLEAN ;


END TimerHandler.
```

4.3.5 gm2-libs-coroutines/SYSTEM

```
DEFINITION MODULE SYSTEM ;

(*
   Description: Implements the SYSTEM dependent module
                in the Modula-2 compiler. 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: *)
                 ADDRESS, WORD, BYTE, BITSET, ADR, TSIZE, THROW,
                 (* SIZE is exported depending upon -fpim2 and
                    -fpedantic *)
                 (* and the rest are implemented in SYSTEM.mod *)
                 PROCESS, TRANSFER, NEWPROCESS, IOTRANSFER,
                 LISTEN,
                 ListenLoop, TurnInterrupts ;


TYPE
   PROCESS  = RECORD
                 context: ADDRESS ;
                 ints   : PROTECTION ;
              END ;


(*
   TRANSFER - save the current volatile environment into, p1.
              Restore the volatile environment from, p2.
*)

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. n, is the amount
                in bytes of this workspace. new, is the new process.
*)
```

```
PROCEDURE NEWPROCESS (p: PROC; a: ADDRESS; n: 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.
*)

PROCEDURE IOTRANSFER (VAR First, Second: PROCESS; InterruptNo: CARDINAL) ;


(*
   LISTEN - briefly listen for any interrupts.
*)

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.
*)

PROCEDURE ListenLoop ;


(*
   TurnInterrupts - switches processor interrupts to the protection
                    level, to.  It returns the old value.
```

```
*)

PROCEDURE TurnInterrupts (to: PROTECTION) : PROTECTION ;


END SYSTEM.
```

4.4 M2 ISO Libraries

This directory contains the ISO definition modules and some corresponding implementation modules. All definition files except '`M2RTS.def`', '`RTio.def`' and '`ErrnoCategory.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 (except '`ClientSocket`', '`M2RTS.def`', '`RTio.def`', '`RTgen.def`', '`RTgenif.def`', '`RTfio.def`', '`RTentity.def`' '`SimpleCipher.def`' and '`ErrnoCategory.def`') belong to ISO/IEC (International Organization for Standardization and International Electrotechnical Commission). All implementation modules and '`M2RTS.def`', '`RTio.def`', '`RTgen.def`', '`RTgenif.def`', '`RTfio.def`', '`RTentity.def`', '`SimpleCipher.def`', '`wraptime.def`', '`wrapsock.def`' and '`ErrnoCategory.def`' are Copyright of the FSF and are held under the LGPL.

The ISO definition modules are allowed to be distributed with the compiler.

The following implementation modules are complete: `ChanConsts` `CharClass` `ClientSocket` `ConvStringLong` `ConvStringReal` `ConvTypes` `EXCEPTIONS` `IOChan` `IOConsts` `IOLink` `LongConv` `LongIO` `LongMath` `LongStr` `M2EXCEPTION` `M2RTS` `ProgramArgs` `RawIO` `RealConv` `RealIO` `RealMath` `RealStr` `RndFile` `RTdata` `RTentity` `RTfio` `RTgen` `RTgenif` `RTio` `SLongIO.mod` `SRealIO.mod` `SimpleCipher` `SeqFile` `StdChans` `Storage` `StreamFile` `Strings` `STextIO` `SWholeIO` `SysClock` `SYSTEM` `TermFile` `TERMINATION` `TextIO` `wrapsock` `wraptime` `WholeConv` `WholeIO` and `WholeStr`.

The following modules are incomplete: `GeneralUser`, `LongComplex`, `LowLong`, `LowReal`, `Processes` and `Semaphores`.

4.4.1 gm2-libs-iso/ChanConsts

```
DEFINITION MODULE ChanConsts;

  (* Common types and values for channel open requests and results *)

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
                         stream requested/applies *)
  );

  FlagSet = SET OF ChanFlags;

  (* Singleton values of FlagSet, to allow for example, read + write *)

CONST
  read = FlagSet{readFlag};   (* input operations are requested/available *)
```

```
  write = FlagSet{writeFlag}; (* output operations are requested/available *)
  old = FlagSet{oldFlag};     (* a file may/must/did exist before the channel is op
  text = FlagSet{textFlag};   (* text operations are requested/available *)
  raw = FlagSet{rawFlag};     (* raw operations are requested/available *)
  interactive = FlagSet{interactiveFlag}; (* interactive use is requested/applies *
  echo = FlagSet{echoFlag};   (* echoing by interactive device on removal of charac
                                 input stream requested/applies *)

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 de
     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
     wrongFileType,     (* the file is of the wrong type to support the required ope
     noTextOperations,  (* text operations have been requested, but are not supporte
     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
                          in combination with the requested operations *)
     otherProblem       (* open failed for some other reason *)
    );

END ChanConsts.
```

4.4.2 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 ;


(*
   InitChanId - return a new ChanId.
*)

PROCEDURE InitChanId () : ChanId ;


(*
   KillChanId - deallocate a ChanId.
*)

PROCEDURE KillChanId (c: ChanId) : ChanId ;


(*
   NilChanId - return a NIL pointer.
*)

PROCEDURE NilChanId () : ChanId ;


(*
   GetDeviceId - returns the device id, from, c.
*)

PROCEDURE GetDeviceId (c: ChanId) : IOLink.DeviceId ;


(*
   SetDeviceId - sets the device id in, c.
```

```
*)

PROCEDURE SetDeviceId (c: ChanId; d: IOLink.DeviceId) ;


(*
   GetDevicePtr - returns the device table ptr, from, c.
*)

PROCEDURE GetDevicePtr (c: ChanId) : IOLink.DeviceTablePtr ;


(*
   SetDevicePtr - sets the device table ptr in, c.
*)

PROCEDURE SetDevicePtr (c: ChanId; p: IOLink.DeviceTablePtr) ;


(*
   GetFile - returns the file field from, c.
*)

PROCEDURE GetFile (c: ChanId) : FIO.File ;


(*
   SetFile - sets the file field in, c.
*)

PROCEDURE SetFile (c: ChanId; f: FIO.File) ;


END RTio.
```

4.4.3 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 ;
   DeviceType = (seqfile, streamfile, programargs, stdchans, term, socket, rndfile)


(*
   InitChanDev - initialize and return a ChanDev.
*)

PROCEDURE InitChanDev (t: DeviceType; d: DeviceId; g: GenDevIF) : ChanDev ;


(*
   KillChanDev - deallocates, g.
*)

PROCEDURE KillChanDev (g: GenDevIF) : GenDevIF ;


(*
   RaiseEOFinLook - returns TRUE if the Look procedure
                    should raise an exception if it
                    sees end of file.
*)

PROCEDURE RaiseEOFinLook (g: ChanDev) : BOOLEAN ;


(*
   RaiseEOFinSkip - returns TRUE if the Skip procedure
                    should raise an exception if it
```

```
                    sees end of file.
*)

PROCEDURE RaiseEOFinSkip (g: ChanDev) : BOOLEAN ;


PROCEDURE doLook (g: ChanDev;
                  d: DeviceTablePtr;
                  VAR ch: CHAR;
                  VAR r: ReadResults) ;

PROCEDURE doSkip (g: ChanDev;
                  d: DeviceTablePtr) ;

PROCEDURE doSkipLook (g: ChanDev;
                      d: DeviceTablePtr;
                      VAR ch: CHAR;
                      VAR r: ReadResults) ;

PROCEDURE doWriteLn (g: ChanDev;
                     d: DeviceTablePtr) ;

PROCEDURE doReadText (g: ChanDev;
                      d: DeviceTablePtr;
                      to: ADDRESS;
                      maxChars: CARDINAL;
                      VAR charsRead: CARDINAL) ;

PROCEDURE doWriteText (g: ChanDev;
                       d: DeviceTablePtr;
                       from: ADDRESS;
                       charsToWrite: CARDINAL) ;

PROCEDURE doReadLocs (g: ChanDev;
                      d: DeviceTablePtr;
                      to: ADDRESS;
                      maxLocs: CARDINAL;
                      VAR locsRead: CARDINAL) ;

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.
```

```
*)

PROCEDURE checkErrno (g: ChanDev; d: DeviceTablePtr) ;


END RTgen.
```

4.4.4 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 = 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.
*)

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.
*)

PROCEDURE clientOpenIP (c: clientInfo;
                        ip: CARDINAL;
                        portNo: CARDINAL) : OpenResults ;


(*
   getClientPortNo - returns the portNo from structure, c.
*)

PROCEDURE getClientPortNo (c: clientInfo) : CARDINAL ;
```

```
(*
   getClientHostname - fills in the hostname of the server
                       the to which the client is connecting.
*)

PROCEDURE getClientHostname (c: clientInfo;
                             hostname: ADDRESS; high: CARDINAL) ;


(*
   getClientSocketFd - returns the sockFd from structure, c.
*)

PROCEDURE getClientSocketFd (c: clientInfo) : INTEGER ;


(*
   getClientIP - returns the sockFd from structure, s.
*)

PROCEDURE getClientIP (c: clientInfo) : CARDINAL ;


(*
   getPushBackChar - returns TRUE if a pushed back character
                     is available.
*)

PROCEDURE getPushBackChar (c: clientInfo; VAR ch: CHAR) : BOOLEAN ;


(*
   setPushBackChar - returns TRUE if it is able to push back a
                     character.
*)

PROCEDURE setPushBackChar (c: clientInfo; ch: CHAR) : BOOLEAN ;


(*
   getSizeOfClientInfo - returns the sizeof (opaque data type).
*)

PROCEDURE getSizeOfClientInfo () : CARDINAL ;
```

```
END wrapsock.
```

4.4.5 gm2-libs-iso/pth

```
DEFINITION MODULE FOR "C" pth ;

(*
    Description: provides the absolute minimal interface to the libpth
                 necessary for COROUTINES, rtInt.
*)

FROM SYSTEM IMPORT ADDRESS ;

TYPE
   proc       = PROCEDURE (ADDRESS) ;
   size_t     = CARDINAL ;
   pth_uctx_t = ADDRESS ;


PROCEDURE pth_select (p1: INTEGER;
                      p2: ADDRESS;
                      p3: ADDRESS;
                      p4: ADDRESS;
                      p5: ADDRESS) : INTEGER ;

PROCEDURE pth_uctx_create (p: ADDRESS) : INTEGER ;

PROCEDURE pth_uctx_make (p1: pth_uctx_t;
                         p2: ADDRESS;
                         p3: size_t;
                         p4: ADDRESS;
                         p5: proc;
                         p6: ADDRESS;
                         p7: pth_uctx_t) : INTEGER;

PROCEDURE pth_uctx_save (p1: pth_uctx_t) : INTEGER ;

PROCEDURE pth_uctx_switch (p1: pth_uctx_t; p2: pth_uctx_t) : INTEGER ;

PROCEDURE pth_init () : INTEGER ;


END pth.
```

4.4.6 gm2-libs-iso/CharClass

```
DEFINITION MODULE CharClass;

  (* Classification of values of the type CHAR *)

PROCEDURE IsNumeric (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a numeric character *)

PROCEDURE IsLetter (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a letter *)

PROCEDURE IsUpper (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as an upper case letter *)

PROCEDURE IsLower (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch is classified as a lower case letter *)

PROCEDURE IsControl (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch represents a control function *)

PROCEDURE IsWhiteSpace (ch: CHAR): BOOLEAN;
  (* Returns TRUE if and only if ch represents a space character or a format effect

END CharClass.
```

4.4.7 gm2-libs-iso/Strings

```
DEFINITION MODULE Strings;

  (* Facilities for manipulating strings *)

TYPE
  String1 = ARRAY [0..0] OF CHAR;
    (* String1 is provided for constructing a value of a single-character string ty
       single character value in order to pass CHAR values to ARRAY OF CHAR paramet
    *)

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 i
   variable parameter.  They all have the property that if the length of the constr
   value exceeds the capacity of the variable parameter, a truncated value is assig
   if the length of the constructed string value is less than the capacity of the v
   parameter, a string terminator is appended before assignment is performed.
*)

PROCEDURE Assign (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Copies source to destination *)

PROCEDURE Extract (source: ARRAY OF CHAR; startIndex, numberToExtract: CARDINAL;
                   VAR destination: ARRAY OF CHAR);
  (* Copies at most numberToExtract characters from source to destination, starting
     startIndex in source.
  *)

PROCEDURE Delete (VAR stringVar: ARRAY OF CHAR; startIndex, numberToDelete:
CARDINAL);
  (* Deletes at most numberToDelete characters from stringVar, starting at position
     startIndex.
  *)

PROCEDURE Insert (source: ARRAY OF CHAR; startIndex: CARDINAL;
                  VAR destination: ARRAY OF CHAR);
  (* Inserts source into destination at position startIndex *)

PROCEDURE Replace (source: ARRAY OF CHAR; startIndex: CARDINAL;
                   VAR destination: ARRAY OF CHAR);
  (* Copies source into destination, starting at position startIndex. Copying stops
```

```
     all of source has been copied, or when the last character of the string value
     destination has been replaced.
  *)

PROCEDURE Append (source: ARRAY OF CHAR; VAR destination: ARRAY OF CHAR);
  (* Appends source to destination. *)

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.
*)

PROCEDURE CanAssignAll (sourceLength: CARDINAL; VAR destination: ARRAY OF CHAR): BO
  (* Returns TRUE if a number of characters, indicated by sourceLength, will fit in
     destination; otherwise returns FALSE.
  *)

PROCEDURE CanExtractAll (sourceLength, startIndex, numberToExtract: CARDINAL;
                         VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there are numberToExtract characters starting at startIndex an
     within the sourceLength of some string, and if the capacity of destination is
     sufficient to hold numberToExtract characters; otherwise returns FALSE.
  *)

PROCEDURE CanDeleteAll (stringLength, startIndex, numberToDelete: CARDINAL): BOOLEA
  (* Returns TRUE if there are numberToDelete characters starting at startIndex and
     within the stringLength of some string; otherwise returns FALSE.
  *)

PROCEDURE CanInsertAll (sourceLength, startIndex: CARDINAL;
                        VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is room for the insertion of sourceLength characters fro
     some string into destination starting at startIndex; otherwise returns FALSE.
  *)

PROCEDURE CanReplaceAll (sourceLength, startIndex: CARDINAL;
                         VAR destination: ARRAY OF CHAR): BOOLEAN;
  (* Returns TRUE if there is room for the replacement of sourceLength characters i
     destination starting at startIndex; otherwise returns FALSE.
  *)

PROCEDURE CanAppendAll (sourceLength: CARDINAL; VAR destination: ARRAY OF CHAR): BO
  (* Returns TRUE if there is sufficient room in destination to append a string of
     length sourceLength to the string in destination; otherwise returns FALSE.
  *)
```

```
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, a
   location of substrings within strings.
*)

TYPE
  CompareResults = (less, equal, greater);

PROCEDURE Compare (stringVal1, stringVal2: ARRAY OF CHAR): CompareResults;
  (* Returns less, equal, or greater, according as stringVal1 is lexically less tha
     equal to, or greater than stringVal2.
  *)

PROCEDURE Equal (stringVal1, stringVal2: ARRAY OF CHAR): BOOLEAN;
  (* Returns Strings.Compare(stringVal1, stringVal2) = Strings.equal *)

PROCEDURE FindNext (pattern, stringToSearch: ARRAY OF CHAR; startIndex: CARDINAL;
                    VAR patternFound: BOOLEAN; VAR posOfPattern: CARDINAL);
  (* Looks forward for next occurrence of pattern in stringToSearch, starting the s
     position startIndex. If startIndex < LENGTH(stringToSearch) and pattern is fou
     patternFound is returned as TRUE, and posOfPattern contains the start position
     stringToSearch of pattern. Otherwise patternFound is returned as FALSE, and po
     is unchanged.
  *)

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 re
     position of the first character of the pattern if found. The search for the pa
     begins at startIndex. If pattern is found, patternFound is returned as TRUE, a
     posOfPattern contains the start position in stringToSearch of pattern in the r
     [0..startIndex]. Otherwise patternFound is returned as FALSE, and posOfPattern
  *)

PROCEDURE FindDiff (stringVal1, stringVal2: ARRAY OF CHAR;
                    VAR differenceFound: BOOLEAN; VAR posOfDifference: CARDINAL);
  (* Compares the string values in stringVal1 and stringVal2 for differences. If th
     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.
  *)
```

```
PROCEDURE Capitalize (VAR stringVar: ARRAY OF CHAR);
  (* Applies the function CAP to each character of the string value in stringVar. *


END Strings.
```

4.4.8 gm2-libs-iso/LongConv

```
DEFINITION MODULE LongConv;

  (* Low-level LONGREAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults; (* strAllRight, strOutOfRange,
                                          strWrongFormat, strEmpty *)

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.
  *)

PROCEDURE FormatReal (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to LONGREAL. *)

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.
  *)

PROCEDURE LengthFloatReal (real: LONGREAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point string
     representation of real with sigFigs significant figures.
  *)

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.
  *)

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.
  *)

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.
```

4.4.9 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}
  *)

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.
  *)

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.
  *)

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.
  *)

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.
  *)
```

```
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.
```

4.4.10 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 *)

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.
  *)

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.
  *)

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.
  *)

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 *)

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 *)

PROCEDURE WriteChar (cid: IOChan.ChanId; ch: CHAR);
  (* Writes the value of ch to the output stream cid. *)

PROCEDURE WriteLn (cid: IOChan.ChanId);
  (* Writes a line mark to the output stream cid. *)

PROCEDURE WriteString (cid: IOChan.ChanId; s: ARRAY OF CHAR);
  (* Writes the string value in s to the output stream cid. *)

END TextIO.
```

4.4.11 gm2-libs-iso/IOLink

```
DEFINITION MODULE IOLink;

(* Types and procedures for the standard implementation of channels *)

IMPORT IOChan, IOConsts, ChanConsts, SYSTEM;

TYPE
  DeviceId;
    (* Values of this type are used to identify new device modules,
       and are normally obtained by them during their initialization.
    *)

PROCEDURE AllocateDeviceId (VAR did: DeviceId);
  (* Allocates a unique value of type DeviceId, and assigns this
     value to did. *)

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.
  *)

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 = POINTER TO DeviceTable;
    (* Values of this type are used to refer to device tables *)

TYPE
  LookProc      = PROCEDURE (DeviceTablePtr, VAR CHAR, VAR IOConsts.ReadResults) ;
  SkipProc      = PROCEDURE (DeviceTablePtr) ;
  SkipLookProc  = PROCEDURE (DeviceTablePtr, VAR CHAR, VAR IOConsts.ReadResults) ;
  WriteLnProc   = PROCEDURE (DeviceTablePtr) ;
  TextReadProc  = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL, VAR CARDINAL
  TextWriteProc = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL) ;
  RawReadProc   = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL, VAR CARDINAL
  RawWriteProc  = PROCEDURE (DeviceTablePtr, SYSTEM.ADDRESS, CARDINAL) ;
  GetNameProc   = PROCEDURE (DeviceTablePtr, VAR ARRAY OF CHAR) ;
  ResetProc     = PROCEDURE (DeviceTablePtr) ;
```

```
  FlushProc     = PROCEDURE (DeviceTablePtr) ;
  FreeProc      = PROCEDURE (DeviceTablePtr) ;
     (* Carry out the operations involved in closing the corresponding
        channel, including flushing buffers, but do not unmake the
        channel.
     *)


TYPE
  DeviceData = SYSTEM.ADDRESS;

  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.
*)

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.
*)

PROCEDURE IsDevice (cid: IOChan.ChanId; did: DeviceId) : BOOLEAN;


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.
*)

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 ex
     is raised, and the string value in s is included in the exception message.
  *)

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.
  *)

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.
```

4.4.12 gm2-libs-iso/ClientSocket

```
DEFINITION MODULE ClientSocket ;

(*
    Description: provides a client TCP interface for ChanId's.
*)

FROM IOChan IMPORT ChanId ;
FROM ChanConsts IMPORT FlagSet, OpenResults ;


(*
   OpenSocket - opens a TCP client connection to host:port.
*)

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.
*)

PROCEDURE Close (VAR cid: ChanId) ;


(*
   IsSocket - tests if the channel identified by cid is open as
              a client socket stream.
*)

PROCEDURE IsSocket (cid: ChanId) : BOOLEAN ;


END ClientSocket.
```

4.4.13 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, REAL96,
                 REAL128, COMPLEX32, COMPLEX64, COMPLEX96,
                 COMPLEX128,
                 ADDADR, SUBADR, DIFADR, MAKEADR, ADR, ROTATE,
                 SHIFT, CAST, TSIZE,

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

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

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

TYPE
   LOC ;
   ADDRESS ;
   BYTE ;
   WORD ;
   INTEGER8 ;
   INTEGER16 ;
   INTEGER32 ;
   INTEGER64 ;
   CARDINAL8 ;
```

```
   CARDINAL16 ;
   CARDINAL32 ;
   CARDINAL64 ;
   WORD16 ;
   WORD32 ;
   WORD64 ;
   BITSET8 ;
   BITSET16 ;
   BITSET32 ;
   REAL32 ;
   REAL64 ;
   REAL96 ;
   REAL128 ;
   COMPLEX32 ;
   COMPLEX64 ;
   COMPLEX96 ;
   COMPLEX128 ;

TYPE
  LOC; (* A system basic type. Values are the uninterpreted
          contents of the smallest addressable unit of storage *)
  ADDRESS = POINTER TO LOC;
  WORD = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

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

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

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

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

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.
  *)

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.
  *)

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

PROCEDURE ROTATE (val: <a packedset 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.
  *)

PROCEDURE SHIFT (val: <a packedset 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.
  *)

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.
  *)

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.
  *)
```

```
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.
  *)
*)


(* 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.
*)

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.
*)

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.
```

```
*)

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.
*)

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.
*)

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.
*)

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


END SYSTEM.
```

4.4.14 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}
  *)

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.
  *)

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.
  *)

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.
  *)

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.
```

```
  *)

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.
```

4.4.15 gm2-libs-iso/ShortComplexMath

```
DEFINITION MODULE ShortComplexMath;

  (* Mathematical functions for the type SHORTCOMPLEX *)

CONST
  i =    CMPLX (0.0, 1.0);
  one =  CMPLX (1.0, 0.0);
  zero = CMPLX (0.0, 0.0);

PROCEDURE abs (z: SHORTCOMPLEX): SHORTREAL;
  (* Returns the length of z *)

PROCEDURE arg (z: SHORTCOMPLEX): SHORTREAL;
  (* Returns the angle that z subtends to the positive real axis *)

PROCEDURE conj (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the complex conjugate of z *)

PROCEDURE power (base: SHORTCOMPLEX; exponent: SHORTREAL): SHORTCOMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE sqrt (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the principal square root of z *)

PROCEDURE exp (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the complex exponential of z *)

PROCEDURE ln (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

PROCEDURE sin (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the sine of z *)

PROCEDURE cos (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the cosine of z *)

PROCEDURE tan (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the tangent of z *)

PROCEDURE arcsin (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the arcsine of z *)

PROCEDURE arccos (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the arccosine of z *)
```

```
PROCEDURE arctan (z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the arctangent of z *)

PROCEDURE polarToComplex (abs, arg: SHORTREAL): SHORTCOMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

PROCEDURE scalarMult (scalar: SHORTREAL; z: SHORTCOMPLEX): SHORTCOMPLEX;
  (* Returns the scalar product of scalar with z *)

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; otherwis
     returns FALSE.
  *)

END ShortComplexMath.
```

4.4.16 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.
*)

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.
*)

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.
*)

PROCEDURE Close (VAR cid: ChanId) ;


(*
```

```
   IsSocket - tests if the channel identified by cid is open as
              a server socket stream.
*)

PROCEDURE IsSocket (cid: ChanId) : BOOLEAN ;


END ServerSocket.
```

4.4.17 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.
*)

PROCEDURE writeString (cid: IOChan.ChanId; s: String) ;


(*
   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.
*)

PROCEDURE writeFieldWidth (cid: IOChan.ChanId;
                           s: String; width: CARDINAL) ;


END StringChan.
```

4.4.18 gm2-libs-iso/SIOResult

```
DEFINITION MODULE SIOResult;

  (* Read results for the default input channel *)

IMPORT IOConsts;

TYPE
  ReadResults = IOConsts.ReadResults;

  (*
    ReadResults =   (* This type is used to classify the result of an input operati
    (
      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 *)
    );
  *)

PROCEDURE ReadResult (): ReadResults;
  (* Returns the result for the last read operation on the default input channel. *

END SIOResult.
```

4.4.19 gm2-libs-iso/WholeConv

```
DEFINITION MODULE WholeConv;

  (* Low-level whole-number/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults;
        (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

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.
  *)

PROCEDURE FormatInt (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to INTEGER. *)

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.
   *)

PROCEDURE LengthInt (int: INTEGER): CARDINAL;
  (* Returns the number of characters in the string representation of
     int.
   *)

PROCEDURE ScanCard (inputCh: CHAR; VAR chClass: ConvTypes.ScanClass;
                    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.
   *)

PROCEDURE FormatCard (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to CARDINAL.
   *)

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.
   *)

PROCEDURE LengthCard (card: CARDINAL): CARDINAL;
  (* Returns the number of characters in the string representation of
     card.
   *)

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.
```

4.4.20 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}
  *)

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.
  *)

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. *)

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.
  *)

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.
```

4.4.21 gm2-libs-iso/LongMath

```
DEFINITION MODULE LongMath;

  (* Mathematical functions for the type LONGREAL *)

CONST
  pi   = 3.14159265358979323846264338327950288419 72;
  exp1 = 2.71828182845904523536028747135266249775 72;

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

PROCEDURE __BUILTIN__ exp (x: LONGREAL): LONGREAL;
  (* Returns the exponential of x *)

PROCEDURE __BUILTIN__ ln (x: LONGREAL): LONGREAL;
  (* Returns the natural logarithm of x *)

  (* The angle in all trigonometric functions is measured in radians *)

PROCEDURE __BUILTIN__ sin (x: LONGREAL): LONGREAL;
  (* Returns the sine of x *)

PROCEDURE __BUILTIN__ cos (x: LONGREAL): LONGREAL;
  (* Returns the cosine of x *)

PROCEDURE tan (x: LONGREAL): LONGREAL;
  (* Returns the tangent of x *)

PROCEDURE arcsin (x: LONGREAL): LONGREAL;
  (* Returns the arcsine of x *)

PROCEDURE arccos (x: LONGREAL): LONGREAL;
  (* Returns the arccosine of x *)

PROCEDURE arctan (x: LONGREAL): LONGREAL;
  (* Returns the arctangent of x *)

PROCEDURE power (base, exponent: LONGREAL): LONGREAL;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE round (x: LONGREAL): INTEGER;
  (* Returns the value of x rounded to the nearest integer *)

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 LongMath.
```

4.4.22 gm2-libs-iso/SRawIO

```
DEFINITION MODULE SRawIO;

  (* Reading and writing data over default channels using raw operations, that is,
     conversion or interpretation. The read result is of the type IOConsts.ReadResu
  *)

IMPORT SYSTEM;

PROCEDURE Read (VAR to: ARRAY OF SYSTEM.LOC);
  (* Reads storage units from the default input channel, and assigns them to succes
     components of to.  The read result is set to the value allRight, wrongFormat,
     endOfInput.
  *)

PROCEDURE Write (from: ARRAY OF SYSTEM.LOC);
  (* Writes storage units to the default output channel from successive components
  *)

END SRawIO.
```

4.4.23 gm2-libs-iso/STextIO

```
DEFINITION MODULE STextIO;

  (* Input and output of character and string types over default channels. The read
     is of the type IOConsts.ReadResults.
  *)

  (* The following procedures do not read past line marks *)

PROCEDURE ReadChar (VAR ch: CHAR);
  (* If possible, removes a character from the default input stream, and assigns th
     corresponding value to ch.  The read result is set to allRight, endOfLine or
     endOfInput.
  *)

PROCEDURE ReadRestLine (VAR s: ARRAY OF CHAR);
  (* Removes any remaining characters from the default input stream before the next
     mark, copying to s as many as can be accommodated as a string value.  The read
     is set to the value allRight, outOfRange, endOfLine, or endOfInput.
  *)

PROCEDURE ReadString (VAR s: ARRAY OF CHAR);
  (* Removes only those characters from the default input stream before the next li
     that can be accommodated in s as a string value, and copies them to s. The rea
     is set to the value allRight, endOfLine, or endOfInput.
  *)

PROCEDURE ReadToken (VAR s: ARRAY OF CHAR);
  (* Skips leading spaces, and then removes characters from the default input strea
     the next space or line mark, copying to s as many as can be accommodated as a
     value.  The read result is set to the value allRight, outOfRange, endOfLine, o
     endOfInput.
  *)

  (* The following procedure reads past the next line mark *)

PROCEDURE SkipLine;
  (* Removes successive items from the default input stream up to and including the
     line mark or until the end of input is reached. The read result is set to the
     allRight, or endOfInput.
  *)


  (* Output procedures *)

PROCEDURE WriteChar (ch: CHAR);
```

```
  (* Writes the value of ch to the default output stream. *)

PROCEDURE WriteLn;
  (* Writes a line mark to the default output stream. *)

PROCEDURE WriteString (s: ARRAY OF CHAR);
  (* Writes the string value of s to the default output stream. *)

END STextIO.
```

4.4.24 gm2-libs-iso/LowLong

```
DEFINITION MODULE LowLong;

  (* Access to underlying properties of the type LONGREAL *)

CONST
  radix      = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, radix> )) ;      (* ZType *)
  places     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, places> )) ;     (* ZType *)
  expoMin    = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, expoMin> )) ;    (* ZType *)
  expoMax    = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, expoMax> )) ;    (* ZType *)
  large      = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, large> )) ;      (* RType *)
  small      = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, small> )) ;      (* RType *)
  IEC559     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, IEC559> )) ;     (* BOOLEAN
  LIA1       = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, LIA1> )) ;       (* BOOLEAN
  ISO        = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, ISO> )) ;        (* BOOLEAN
  IEEE       = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, IEEE> )) ;       (* BOOLEAN
  rounds     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, rounds> )) ;     (* BOOLEAN
  gUnderflow = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, gUnderflow> )) ; (* BOOLEAN
  exception  = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, exception> )) ;  (* BOOLEAN
  extend     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, extend> )) ;     (* BOOLEAN
  nModes     = __ATTRIBUTE__ __BUILTIN__ (( <LONGREAL, nModes> )) ;     (* ZType *)

TYPE
  Modes = PACKEDSET OF [0 .. nModes-1];

PROCEDURE exponent (x: LONGREAL): INTEGER;
  (* Returns the exponent value of x *)

PROCEDURE fraction (x: LONGREAL): LONGREAL;
  (* Returns the significand (or significant part) of x *)

PROCEDURE sign (x: LONGREAL): LONGREAL;
  (* Returns the signum of x *)

PROCEDURE succ (x: LONGREAL): LONGREAL;
  (* Returns the next value of the type LONGREAL greater than x *)

PROCEDURE ulp (x: LONGREAL): LONGREAL;
  (* Returns the value of a unit in the last place of x *)

PROCEDURE pred (x: LONGREAL): LONGREAL;
  (* Returns the previous value of the type LONGREAL less than x *)

PROCEDURE intpart (x: LONGREAL): LONGREAL;
  (* Returns the integer part of x *)
```

```
PROCEDURE fractpart (x: LONGREAL): LONGREAL;
  (* Returns the fractional part of x *)

PROCEDURE scale (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of x * radix ** n *)

PROCEDURE trunc (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of the first n places of x *)

PROCEDURE round (x: LONGREAL; n: INTEGER): LONGREAL;
  (* Returns the value of x rounded to the first n places *)

PROCEDURE synthesize (expart: INTEGER; frapart: LONGREAL): LONGREAL;
  (* Returns a value of the type LONGREAL constructed from the given expart and fra

PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type LON

PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

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; otherwis
     returns FALSE.
  *)

END LowLong.
```

4.4.25 gm2-libs-iso/M2EXCEPTION

```
DEFINITION MODULE M2EXCEPTION;

(* Provides facilities for identifying language exceptions *)

TYPE
  M2Exceptions =
    (indexException,     rangeException,         caseSelectException,  invalidLocat
     functionException,  wholeValueException,    wholeDivException,    realValueExc
     realDivException,   complexValueException,  complexDivException,  protExceptio
     sysException,       coException,            exException
    );

PROCEDURE M2Exception (): M2Exceptions;
  (* If the current coroutine is in the exceptional execution state because of the
     of a language exception, returns the corresponding enumeration value, and othe
     raises an exception.
  *)

PROCEDURE IsM2Exception (): BOOLEAN;
  (* If the current coroutine is in the exceptional execution state because of the
     of a language exception, returns TRUE, and otherwise returns FALSE.
  *)

END M2EXCEPTION.
```

4.4.26 gm2-libs-iso/LowShort

```
DEFINITION MODULE LowShort;

  (* Access to underlying properties of the type SHORTREAL *)

CONST
  radix      = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, radix> )) ;      (* ZType *
  places     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, places> )) ;     (* ZType *
  expoMin    = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, expoMin> )) ;    (* ZType *
  expoMax    = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, expoMax> )) ;    (* ZType *
  large      = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, large> )) ;      (* RType *
  small      = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, small> )) ;      (* RType *
  IEC559     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, IEC559> )) ;     (* BOOLEAN
  LIA1       = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, LIA1> )) ;       (* BOOLEAN
  ISO        = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, ISO> )) ;        (* BOOLEAN
  IEEE       = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, IEEE> )) ;       (* BOOLEAN
  rounds     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, rounds> )) ;     (* BOOLEAN
  gUnderflow = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, gUnderflow> )) ; (* BOOLEAN
  exception  = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, exception> )) ;  (* BOOLEAN
  extend     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, extend> )) ;     (* BOOLEAN
  nModes     = __ATTRIBUTE__ __BUILTIN__ (( <SHORTREAL, nModes> )) ;     (* ZType *

TYPE
  Modes = PACKEDSET OF [0 .. nModes-1];

PROCEDURE exponent (x: SHORTREAL): INTEGER;
  (* Returns the exponent value of x *)

PROCEDURE fraction (x: SHORTREAL): SHORTREAL;
  (* Returns the significand (or significant part) of x *)

PROCEDURE sign (x: SHORTREAL): SHORTREAL;
  (* Returns the signum of x *)

PROCEDURE succ (x: SHORTREAL): SHORTREAL;
  (* Returns the next value of the type SHORTREAL greater than x *)

PROCEDURE ulp (x: SHORTREAL): SHORTREAL;
  (* Returns the value of a unit in the last place of x *)

PROCEDURE pred (x: SHORTREAL): SHORTREAL;
  (* Returns the previous value of the type SHORTREAL less than x *)

PROCEDURE intpart (x: SHORTREAL): SHORTREAL;
  (* Returns the integer part of x *)
```

```
PROCEDURE fractpart (x: SHORTREAL): SHORTREAL;
  (* Returns the fractional part of x *)

PROCEDURE scale (x: SHORTREAL; n: INTEGER): SHORTREAL;
  (* Returns the value of x * radix ** n *)

PROCEDURE trunc (x: SHORTREAL; n: INTEGER): SHORTREAL;
  (* Returns the value of the first n places of x *)

PROCEDURE round (x: SHORTREAL; n: INTEGER): SHORTREAL;
  (* Returns the value of x rounded to the first n places *)

PROCEDURE synthesize (expart: INTEGER; frapart: SHORTREAL): SHORTREAL;
  (* Returns a value of the type SHORTREAL constructed from the given expart and fr

PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type SHO

PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

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; otherwis
     returns FALSE.
  *)

END LowShort.
```

4.4.27 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.
  *)

  (* 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}
  *)

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.
  *)

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.
  *)

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.
  *)

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.
  *)

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.
```

4.4.28 gm2-libs-iso/ConvStringReal

```
DEFINITION MODULE ConvStringReal ;

(*
    Description: provides a set of procedures which translate
                 floating point numbers to their String equivalent.
                 This is not part of the ISO standard, but it is
                 used by a number of ISO modules.
*)

FROM DynamicStrings IMPORT String ;


(*
   RealToFloatString - converts a real with, sigFigs, into a string
                       and returns the result as a string.
*)

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.
*)

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.
*)

PROCEDURE RealToFixedString (real: REAL; place: INTEGER) : String ;


END ConvStringReal.
```

4.4.29 gm2-libs-iso/RealMath

```
DEFINITION MODULE RealMath;

  (* Mathematical functions for the type REAL *)

CONST
  pi   = 3.14159265358979323846264338327950288419 72;
  exp1 = 2.71828182845904523536028747135266249775 72;

PROCEDURE __BUILTIN__ sqrt (x: REAL): REAL;
  (* Returns the positive square root of x *)

PROCEDURE __BUILTIN__ exp (x: REAL): REAL;
  (* Returns the exponential of x *)

PROCEDURE __BUILTIN__ ln (x: REAL): REAL;
  (* Returns the natural logarithm of x *)

  (* The angle in all trigonometric functions is measured in radians *)

PROCEDURE __BUILTIN__ sin (x: REAL): REAL;
  (* Returns the sine of x *)

PROCEDURE __BUILTIN__ cos (x: REAL): REAL;
  (* Returns the cosine of x *)

PROCEDURE tan (x: REAL): REAL;
  (* Returns the tangent of x *)

PROCEDURE arcsin (x: REAL): REAL;
  (* Returns the arcsine of x *)

PROCEDURE arccos (x: REAL): REAL;
  (* Returns the arccosine of x *)

PROCEDURE arctan (x: REAL): REAL;
  (* Returns the arctangent of x *)

PROCEDURE power (base, exponent: REAL) : REAL;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE round (x: REAL) : INTEGER;
  (* Returns the value of x rounded to the nearest integer *)

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; otherwis
      returns FALSE.
  *)

END RealMath.
```

4.4.30 gm2-libs-iso/RealConv

```
DEFINITION MODULE RealConv;

  (* Low-level REAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)
  ConvResults = ConvTypes.ConvResults;

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.
   *)

PROCEDURE FormatReal (str: ARRAY OF CHAR): ConvResults;
  (* Returns the format of the string value for conversion to REAL. *)

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.
  *)

PROCEDURE LengthFloatReal (real: REAL; sigFigs: CARDINAL): CARDINAL;
  (* Returns the number of characters in the floating-point string
     representation of real with sigFigs significant figures.
  *)

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.
  *)

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.
  *)

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.
```

4.4.31 gm2-libs-iso/TERMINATION

```
DEFINITION MODULE TERMINATION;

(* Provides facilities for enquiries concerning the occurrence of termination event

PROCEDURE IsTerminating (): BOOLEAN ;
  (* Returns true if any coroutine has started  program termination and false other

PROCEDURE HasHalted (): BOOLEAN ;
  (* Returns true if a call to HALT has been made and false otherwise. *)

END TERMINATION.
```

4.4.32 gm2-libs-iso/GeneralUserExceptions

```
DEFINITION MODULE GeneralUserExceptions;

(* Provides facilities for general user-defined exceptions *)

TYPE
  GeneralExceptions = (problem, disaster);

PROCEDURE RaiseGeneralException (exception: GeneralExceptions;
                                 text: ARRAY OF CHAR);
  (* Raises exception using text as the associated message *)

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.
  *)

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.
```

4.4.33 gm2-libs-iso/LongComplexMath

```
DEFINITION MODULE LongComplexMath;

  (* Mathematical functions for the type LONGCOMPLEX *)

CONST
  i =    CMPLX (0.0, 1.0);
  one =  CMPLX (1.0, 0.0);
  zero = CMPLX (0.0, 0.0);

PROCEDURE abs (z: LONGCOMPLEX): LONGREAL;
  (* Returns the length of z *)

PROCEDURE arg (z: LONGCOMPLEX): LONGREAL;
  (* Returns the angle that z subtends to the positive real axis *)

PROCEDURE conj (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the complex conjugate of z *)

PROCEDURE power (base: LONGCOMPLEX; exponent: LONGREAL): LONGCOMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE sqrt (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the principal square root of z *)

PROCEDURE exp (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the complex exponential of z *)

PROCEDURE ln (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

PROCEDURE sin (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the sine of z *)

PROCEDURE cos (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the cosine of z *)

PROCEDURE tan (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the tangent of z *)

PROCEDURE arcsin (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arcsine of z *)

PROCEDURE arccos (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arccosine of z *)
```

```
PROCEDURE arctan (z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the arctangent of z *)

PROCEDURE polarToComplex (abs, arg: LONGREAL): LONGCOMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

PROCEDURE scalarMult (scalar: LONGREAL; z: LONGCOMPLEX): LONGCOMPLEX;
  (* Returns the scalar product of scalar with z *)

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; otherwis
     returns FALSE.
  *)

END LongComplexMath.
```

4.4.34 gm2-libs-iso/Storage

```
DEFINITION MODULE Storage;

  (* Facilities for dynamically allocating and deallocating storage *)

IMPORT SYSTEM;

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.
  *)

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.
  *)

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 = (
    nilDeallocation,             (* first argument to DEALLOCATE is NIL *)
    pointerToUnallocatedStorage, (* storage to deallocate not allocated by ALLOCATE
    wrongStorageToUnallocate     (* amount to deallocate is not amount allocated *)
  );

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.
  *)

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.
```

4.4.35 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  = ADDRESS ;
   timezone = ADDRESS ;
   tm       = ADDRESS ;


(*
   InitTimeval - returns a newly created opaque type.
*)

PROCEDURE InitTimeval () : timeval ;


(*
   KillTimeval - deallocates the memory associated with an
                 opaque type.
*)

PROCEDURE KillTimeval (tv: timeval) : timeval ;


(*
   InitTimezone - returns a newly created opaque type.
*)

PROCEDURE InitTimezone () : timezone ;


(*
   KillTimezone - deallocates the memory associated with an
                  opaque type.
*)

PROCEDURE KillTimezone (tv: timezone) : timezone ;
```

```
(*
   InitTM - returns a newly created opaque type.
*)

PROCEDURE InitTM () : tm ;


(*
   KillTM - deallocates the memory associated with an
            opaque type.
*)

PROCEDURE KillTM (tv: tm) : tm ;


(*
   gettimeofday - calls gettimeofday(2) with the same parameters, tv,
                  and, tz.  It returns 0 on success.
*)

PROCEDURE gettimeofday (tv: timeval; tz: timezone) : INTEGER ;


(*
   settimeofday - calls settimeofday(2) with the same parameters, tv,
                  and, tz.  It returns 0 on success.
*)

PROCEDURE settimeofday (tv: timeval; tz: timezone) : INTEGER ;


(*
   GetFractions - returns the tv_usec field inside the timeval structure
                  as a CARDINAL.
*)

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.
*)

PROCEDURE localtime_r (tv: timeval; m: tm) : tm ;


(*
   GetYear - returns the year from the structure, m.
*)

PROCEDURE GetYear (m: tm) : CARDINAL ;


(*
   GetMonth - returns the month from the structure, m.
*)

PROCEDURE GetMonth (m: tm) : CARDINAL ;


(*
   GetDay - returns the day of the month from the structure, m.
*)

PROCEDURE GetDay (m: tm) : CARDINAL ;


(*
   GetHour - returns the hour of the day from the structure, m.
*)

PROCEDURE GetHour (m: tm) : CARDINAL ;


(*
   GetMinute - returns the minute within the hour from the structure, m.
*)

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.
*)
```

```
PROCEDURE GetSecond (m: tm) : CARDINAL ;


(*
   GetSummerTime - returns a boolean indicating whether summer time is
                   set.
*)

PROCEDURE GetSummerTime (tz: timezone) : BOOLEAN ;


(*
   GetDST - returns the number of minutes west of GMT.
*)

PROCEDURE GetDST (tz: timezone) : INTEGER ;


(*
   SetTimeval - sets the fields in timeval, tv, with:
                second, minute, hour, day, month, year, fractions.
*)

PROCEDURE SetTimeval (tv: timeval;
                      second, minute, hour, day,
                      month, year, yday, wday, isdst: CARDINAL) ;


(*
   SetTimezone - set the timezone field inside timeval, tv.
*)

PROCEDURE SetTimezone (tv: timeval;
                       zone: CARDINAL; minuteswest: INTEGER) ;


END wraptime.
```

4.4.36 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 ;


(*
   doreadchar - returns a CHAR from the file associated with, g.
*)

PROCEDURE doreadchar (g: GenDevIF; d: DeviceTablePtr) : CHAR ;


(*
   dounreadchar - pushes a CHAR back onto the file associated
                  with, g.
*)

PROCEDURE dounreadchar (g: GenDevIF; d: DeviceTablePtr; ch: CHAR) : CHAR ;


(*
   dogeterrno - returns the errno relating to the generic device.
*)

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.
*)

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.
*)

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.
*)

PROCEDURE dowriteln (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


(*
   iseof - returns TRUE if end of file is seen.
*)

PROCEDURE iseof (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


(*
   iseoln - returns TRUE if end of line is seen.
*)

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.
*)

PROCEDURE iserror (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


END RTfio.
```

4.4.37 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 = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  read = FlagSet{ChanConsts.readFlag};
  (* input operations are requested/available *)
  write = FlagSet{ChanConsts.writeFlag};
  (* output operations are requested/available *)
  text = FlagSet{ChanConsts.textFlag};
  (* text operations are requested/available *)
  raw = FlagSet{ChanConsts.rawFlag};
  (* raw operations are requested/available *)
  echo = FlagSet{ChanConsts.echoFlag};
  (* echoing by interactive device on reading of
     characters from input stream requested/applies
  *)

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.
  *)

PROCEDURE IsTermFile (cid: ChanId): BOOLEAN;
```

```
  (* Tests if the channel identified by cid is open to
     the terminal. *)

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.
```

4.4.38 gm2-libs-iso/ComplexMath

```
DEFINITION MODULE ComplexMath;

  (* Mathematical functions for the type COMPLEX *)

CONST
  i =    CMPLX (0.0, 1.0);
  one =  CMPLX (1.0, 0.0);
  zero = CMPLX (0.0, 0.0);

PROCEDURE __BUILTIN__ abs (z: COMPLEX): REAL;
  (* Returns the length of z *)

PROCEDURE __BUILTIN__ arg (z: COMPLEX): REAL;
  (* Returns the angle that z subtends to the positive real axis *)

PROCEDURE __BUILTIN__ conj (z: COMPLEX): COMPLEX;
  (* Returns the complex conjugate of z *)

PROCEDURE __BUILTIN__ power (base: COMPLEX; exponent: REAL): COMPLEX;
  (* Returns the value of the number base raised to the power exponent *)

PROCEDURE __BUILTIN__ sqrt (z: COMPLEX): COMPLEX;
  (* Returns the principal square root of z *)

PROCEDURE __BUILTIN__ exp (z: COMPLEX): COMPLEX;
  (* Returns the complex exponential of z *)

PROCEDURE __BUILTIN__ ln (z: COMPLEX): COMPLEX;
  (* Returns the principal value of the natural logarithm of z *)

PROCEDURE __BUILTIN__ sin (z: COMPLEX): COMPLEX;
  (* Returns the sine of z *)

PROCEDURE __BUILTIN__ cos (z: COMPLEX): COMPLEX;
  (* Returns the cosine of z *)

PROCEDURE __BUILTIN__ tan (z: COMPLEX): COMPLEX;
  (* Returns the tangent of z *)

PROCEDURE __BUILTIN__ arcsin (z: COMPLEX): COMPLEX;
  (* Returns the arcsine of z *)

PROCEDURE __BUILTIN__ arccos (z: COMPLEX): COMPLEX;
  (* Returns the arccosine of z *)
```

```
PROCEDURE __BUILTIN__ arctan (z: COMPLEX): COMPLEX;
  (* Returns the arctangent of z *)

PROCEDURE polarToComplex (abs, arg: REAL): COMPLEX;
  (* Returns the complex number with the specified polar coordinates *)

PROCEDURE scalarMult (scalar: REAL; z: COMPLEX): COMPLEX;
  (* Returns the scalar product of scalar with z *)

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.
```

4.4.39 gm2-libs-iso/LowReal

```
DEFINITION MODULE LowReal;

  (* Access to underlying properties of the type REAL *)

CONST
  radix      = __ATTRIBUTE__ __BUILTIN__ (( <REAL, radix> )) ;      (* ZType *)
  places     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, places> )) ;     (* ZType *)
  expoMin    = __ATTRIBUTE__ __BUILTIN__ (( <REAL, expoMin> )) ;    (* ZType *)
  expoMax    = __ATTRIBUTE__ __BUILTIN__ (( <REAL, expoMax> )) ;    (* ZType *)
  large      = __ATTRIBUTE__ __BUILTIN__ (( <REAL, large> )) ;      (* RType *)
  small      = __ATTRIBUTE__ __BUILTIN__ (( <REAL, small> )) ;      (* RType *)
  IEC559     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, IEC559> )) ;     (* BOOLEAN *)
  LIA1       = __ATTRIBUTE__ __BUILTIN__ (( <REAL, LIA1> )) ;       (* BOOLEAN *)
  ISO        = __ATTRIBUTE__ __BUILTIN__ (( <REAL, ISO> )) ;        (* BOOLEAN *)
  IEEE       = __ATTRIBUTE__ __BUILTIN__ (( <REAL, IEEE> )) ;       (* BOOLEAN *)
  rounds     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, rounds> )) ;     (* BOOLEAN *)
  gUnderflow = __ATTRIBUTE__ __BUILTIN__ (( <REAL, gUnderflow> )) ; (* BOOLEAN *)
  exception  = __ATTRIBUTE__ __BUILTIN__ (( <REAL, exception> )) ;  (* BOOLEAN *)
  extend     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, extend> )) ;     (* BOOLEAN *)
  nModes     = __ATTRIBUTE__ __BUILTIN__ (( <REAL, nModes> )) ;     (* ZType *)

TYPE
  Modes = PACKEDSET OF [0..nModes-1];

PROCEDURE exponent (x: REAL): INTEGER;
  (* Returns the exponent value of x *)

PROCEDURE fraction (x: REAL): REAL;
  (* Returns the significand (or significant part) of x *)

PROCEDURE sign (x: REAL): REAL;
  (* Returns the signum of x *)

PROCEDURE succ (x: REAL): REAL;
  (* Returns the next value of the type REAL greater than x *)

PROCEDURE ulp (x: REAL): REAL;
  (* Returns the value of a unit in the last place of x *)

PROCEDURE pred (x: REAL): REAL;
  (* Returns the previous value of the type REAL less than x *)

PROCEDURE intpart (x: REAL): REAL;
  (* Returns the integer part of x *)
```

```
PROCEDURE fractpart (x: REAL): REAL;
  (* Returns the fractional part of x *)

PROCEDURE scale (x: REAL; n: INTEGER): REAL;
  (* Returns the value of x * radix ** n *)

PROCEDURE trunc (x: REAL; n: INTEGER): REAL;
  (* Returns the value of the first n places of x *)

PROCEDURE round (x: REAL; n: INTEGER): REAL;
  (* Returns the value of x rounded to the first n places *)

PROCEDURE synthesize (expart: INTEGER; frapart: REAL): REAL;
  (* Returns a value of the type REAL constructed from the given expart and frapart

PROCEDURE setMode (m: Modes);
  (* Sets status flags appropriate to the underlying implementation of the type REA

PROCEDURE currentMode (): Modes;
  (* Returns the current status flags in the form set by setMode *)

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; otherwis
     returns FALSE.
  *)

END LowReal.
```

4.4.40 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.
*)

PROCEDURE InsertCipherLayer (cid: ChanId; key: INTEGER) ;


(*
   RemoveCipherLayer - removes a Caesar cipher below channel, cid.
*)

PROCEDURE RemoveCipherLayer (cid: ChanId) ;


END SimpleCipher.
```

4.4.41 gm2-libs-iso/StdChans

```
DEFINITION MODULE StdChans;

  (* Access to standard and default channels *)

IMPORT IOChan;

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.
  *)

PROCEDURE StdInChan (): ChanId;
  (* Returns the identity of the implementation-defined standard source for
program
     input.
  *)

PROCEDURE StdOutChan (): ChanId;
  (* Returns the identity of the implementation-defined standard source for program
     output.
  *)

PROCEDURE StdErrChan (): ChanId;
  (* Returns the identity of the implementation-defined standard destination for pr
     error messages.
  *)

PROCEDURE NullChan (): ChanId;
  (* Returns the identity of a channel open to the null device. *)

  (* The following functions return the default channel values *)

PROCEDURE InChan (): ChanId;
  (* Returns the identity of the current default input channel. *)

PROCEDURE OutChan (): ChanId;
  (* Returns the identity of the current default output channel. *)

PROCEDURE ErrChan (): ChanId;
  (* Returns the identity of the current default error message channel. *)

  (* The following procedures allow for redirection of the default channels *)
```

```
PROCEDURE SetInChan (cid: ChanId);
  (* Sets the current default input channel to that identified by cid. *)

PROCEDURE SetOutChan (cid: ChanId);
  (* Sets the current default output channel to that identified by cid. *)

PROCEDURE SetErrChan (cid: ChanId);
  (* Sets the current default error channel to that identified by cid. *)

END StdChans.
```

4.4.42 gm2-libs-iso/Semaphores

```
DEFINITION MODULE Semaphores;

  (* Provides mutual exclusion facilities for use by processes. *)

TYPE
  SEMAPHORE;

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.
  *)

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.
  *)

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.
  *)

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.
  *)

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.
```

4.4.43 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;

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.
  *)

PROCEDURE Write (cid: IOChan.ChanId; from: ARRAY OF SYSTEM.LOC);
  (* Writes storage units to cid from successive components
     of from. *)

END RawIO.
```

4.4.44 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 ;

TYPE
   GenDevIF ;
   readchar   = PROCEDURE (GenDevIF, DeviceTablePtr) : CHAR ;
   unreadchar = PROCEDURE (GenDevIF, DeviceTablePtr, CHAR) : CHAR ;
   geterrno   = PROCEDURE (GenDevIF, DeviceTablePtr) : INTEGER ;
   readbytes  = PROCEDURE (GenDevIF, DeviceTablePtr, ADDRESS, CARDINAL, VAR CARDINA
   writebytes = PROCEDURE (GenDevIF, DeviceTablePtr, ADDRESS, CARDINAL, VAR CARDINA
   writeln    = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;
   iseof      = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;
   iseoln     = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;
   iserror    = PROCEDURE (GenDevIF, DeviceTablePtr) : BOOLEAN ;


(*
   InitGenDevIF - initializes a generic device.
*)

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.
*)

PROCEDURE getDID (g: GenDevIF) : DeviceId ;
```

```
(*
   doReadChar - returns the next character from the generic
                device.
*)

PROCEDURE doReadChar (g: GenDevIF; d: DeviceTablePtr) : CHAR ;


(*
   doUnReadChar - pushes back a character to the generic device.
*)

PROCEDURE doUnReadChar (g: GenDevIF; d: DeviceTablePtr; ch: CHAR) : CHAR ;


(*
   doGetErrno - returns the errno relating to the generic device.
*)

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.
*)

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.
*)

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.
*)

PROCEDURE doWrLn (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


(*
   isEOF - returns true if the end of file was reached.
*)

PROCEDURE isEOF (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


(*
   isEOLN - returns true if the end of line was reached.
*)

PROCEDURE isEOLN (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


(*
   isError - returns true if an error was seen in the device.
*)

PROCEDURE isError (g: GenDevIF; d: DeviceTablePtr) : BOOLEAN ;


(*
   KillGenDevIF - deallocates a generic device.
*)

PROCEDURE KillGenDevIF (g: GenDevIF) : GenDevIF ;


END RTgenif.
```

4.4.45 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 ;
   FreeProcedure = PROCEDURE (ADDRESS) ;


(*
   MakeModuleId - creates a unique module Id.
*)

PROCEDURE MakeModuleId (VAR m: ModuleId) ;


(*
   InitData - adds, datum, to the device, d.  The datum
              is associated with ModuleID, m.
*)

PROCEDURE InitData (d: DeviceTablePtr; m: ModuleId;
                    datum: ADDRESS; f: FreeProcedure) ;


(*
   GetData - returns the datum assocated with ModuleId, m.
*)

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.
*)

PROCEDURE KillData (d: DeviceTablePtr; m: ModuleId) ;
```

```
END RTdata.
```

4.4.46 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.
  *)

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.
  *)

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.
  *)

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.
  *)
```

```
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.
  *)

PROCEDURE WriteLn (cid: ChanId);
  (* Writes a line mark over the channel cid. *)

PROCEDURE TextRead (cid: ChanId; to: SYSTEM.ADDRESS; maxChars: CARDINAL;
                    VAR charsRead: CARDINAL);
  (* Reads at most maxChars characters from the current line in cid,
     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.
  *)

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
  *)

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.
  *)

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 *)

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).
  *)

PROCEDURE Reset (cid: ChanId);
  (* Resets the channel cid to a state defined by the device module. *)

PROCEDURE Flush (cid: ChanId);
  (* Flushes any data buffered by the device module out to the channel cid. *)

  (* Access to read results *)

PROCEDURE SetReadResult (cid: ChanId; res: IOConsts.ReadResults);
  (* Sets the read result value for the channel cid to the value res. *)

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 *)

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 =
    (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 *)
    );

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.
  *)

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
     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 = INTEGER;

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.
```

4.4.47 gm2-libs-iso/SeqFile

```
DEFINITION MODULE SeqFile;

  (* Rewindable sequential files *)

IMPORT IOChan, ChanConsts;

TYPE
  ChanId = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  (* input operations are requested/available *)
  read = FlagSet{ChanConsts.readFlag};

  (* output operations are requested/available *)
  write = FlagSet{ChanConsts.writeFlag};

  (* a file may/must/did exist before the channel is opened *)
  old = FlagSet{ChanConsts.oldFlag};

  (* text operations are requested/available *)
  text = FlagSet{ChanConsts.textFlag};

  (* raw operations are requested/available *)
  raw = FlagSet{ChanConsts.rawFlag};

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
     channel.
  *)

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.
  *)

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.
  *)

PROCEDURE IsSeqFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a
     rewindable sequential file. *)

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.
  *)

PROCEDURE Rewrite (cid: ChanId);
  (* 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.
  *)

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.
```

4.4.48 gm2-libs-iso/ProgramArgs

```
DEFINITION MODULE ProgramArgs;

  (* Access to program arguments *)

IMPORT IOChan;

TYPE
  ChanId = IOChan.ChanId;

PROCEDURE ArgChan (): ChanId;
  (* Returns a value that identifies a channel for reading
     program arguments *)

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.
  *)

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.
```

4.4.49 gm2-libs-iso/IOConsts

```
DEFINITION MODULE IOConsts;

  (* Types and constants for input/output modules *)

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.
```

4.4.50 gm2-libs-iso/StreamFile

```
DEFINITION MODULE StreamFile;

  (* Independent sequential data streams *)

IMPORT IOChan, ChanConsts;

TYPE
  ChanId = IOChan.ChanId;
  FlagSet = ChanConsts.FlagSet;
  OpenResults = ChanConsts.OpenResults;

  (* Accepted singleton values of FlagSet *)

CONST
  read = FlagSet{ChanConsts.readFlag};   (* input operations are requested/availabl
  write = FlagSet{ChanConsts.writeFlag}; (* output operations are requested/availab
  old = FlagSet{ChanConsts.oldFlag};     (* a file may/must/did exist before the ch
                                            opened *)
  text = FlagSet{ChanConsts.textFlag};   (* text operations are requested/available
  raw = FlagSet{ChanConsts.rawFlag};     (* raw operations are requested/available


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.
  *)

PROCEDURE IsStreamFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a sequential stream. *)

PROCEDURE Close (VAR cid: ChanId);
  (* If the channel identified by cid is not open to a sequential stream, the excep
     wrongDevice is raised; otherwise closes the channel, and assigns the value ide
     the invalid channel to cid.
  *)

END StreamFile.
```

4.4.51 gm2-libs-iso/RndFile

```
DEFINITION MODULE RndFile;

  (* Random access files *)

IMPORT IOChan, ChanConsts, SYSTEM;

TYPE
   ChanId = IOChan.ChanId;
   FlagSet = ChanConsts.FlagSet;
   OpenResults = ChanConsts.OpenResults;

   (* Accepted singleton values of FlagSet *)

CONST
   (* input operations are requested/available *)
   read = FlagSet{ChanConsts.readFlag};
   (* output operations are requested/available *)
   write = FlagSet{ChanConsts.writeFlag};
   (* a file may/must/did exist before the channel is opened *)
   old = FlagSet{ChanConsts.oldFlag};
   (* text operations are requested/available *)
   text = FlagSet{ChanConsts.textFlag};
   (* raw operations are requested/available *)
   raw = FlagSet{ChanConsts.rawFlag};

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.
  *)

PROCEDURE OpenClean (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 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.
  *)

PROCEDURE IsRndFile (cid: ChanId): BOOLEAN;
  (* Tests if the channel identified by cid is open to a random access file. *)

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 = SIZE(LONGINT) ;
   (* <implementation-defined whole number greater than zero>; *)

TYPE
   FilePos = LONGINT ;  (* ARRAY [1 .. FilePosSize] OF SYSTEM.LOC; *)

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.
  *)

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.
  *)

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.
  *)

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.
  *)
```

```
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.
  *)

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.
```

4.4.52 gm2-libs-iso/WholeStr

```
DEFINITION MODULE WholeStr;

  (* Whole-number/string conversions *)

IMPORT
  ConvTypes;

TYPE
  ConvResults = ConvTypes.ConvResults;
  (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)

(* the string form of a signed whole number is
     ["+" | "-"], decimal digit, {decimal digit}
*)

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.
  *)

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}
*)

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.
  *)

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.
```

4.4.53 gm2-libs-iso/SysClock

```
DEFINITION MODULE SysClock;

(* Facilities for accessing a system clock that records the date
   and time of day *)

CONST
  maxSecondParts = 1000000 ;

TYPE
  Month    = [1 .. 12];
  Day      = [1 .. 31];
  Hour     = [0 .. 23];
  Min      = [0 .. 59];
  Sec      = [0 .. 59];
  Fraction = [0 .. maxSecondParts];
  UTCDiff  = [-780 .. 720];
  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;

PROCEDURE CanGetClock(): BOOLEAN;
(* Tests if the clock can be read *)

PROCEDURE CanSetClock(): BOOLEAN;
(* Tests if the clock can be set *)

PROCEDURE IsValidDateTime(userData: DateTime): BOOLEAN;
(* Tests if the value of userData is a valid *)

PROCEDURE GetClock(VAR userData: DateTime);
(* Assigns local date and time of the day to userData *)
```

```
PROCEDURE SetClock(userData: DateTime);
(* Sets the system time clock to the given local date and
   time *)

END SysClock.
```

4.4.54 gm2-libs-iso/COROUTINES

```
DEFINITION MODULE COROUTINES;

(* Facilities for coroutines and the handling of interrupts *)

IMPORT SYSTEM ;


CONST
  UnassignedPriority = 0 ;

TYPE
  COROUTINE ; (* Values of this type are created dynamically by NEWCOROUTINE
                 and identify the coroutine in subsequent operations *)
  INTERRUPTSOURCE = CARDINAL ;
  PROTECTION = [UnassignedPriority..7] ;


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.
  *)

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.
  *)

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.
  *)
```

```
PROCEDURE ATTACH (source: INTERRUPTSOURCE);
  (* Associates the specified source of interrupts with the calling
     coroutine. *)

PROCEDURE DETACH (source: INTERRUPTSOURCE);
  (* Dissociates the specified source of interrupts from the calling
     coroutine. *)

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.
  *)

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.
  *)

PROCEDURE CURRENT (): COROUTINE;
  (* Returns the identity of the calling coroutine. *)

PROCEDURE LISTEN (p: PROTECTION);
  (* Momentarily changes the protection of the calling coroutine to
     p. *)

PROCEDURE PROT (): PROTECTION;
  (* Returns the protection of the calling coroutine. *)

END COROUTINES.
```

4.4.55 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}
  *)

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.
  *)

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.
  *)

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.
  *)

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.
```

4.4.56 gm2-libs-iso/ConvStringLong

```
DEFINITION MODULE ConvStringLong ;

(*
    Description: provides a set of procedures which translate
                 floating point numbers to their String equivalent.
                 This is not part of the ISO standard, but it is
                 used by a number of ISO modules.
*)

FROM DynamicStrings IMPORT String ;


(*
   RealToFloatString - converts a real with, sigFigs, into a string
                       and returns the result as a string.
*)

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
                     that is a multiple of three.
*)

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.
*)

PROCEDURE RealToFixedString (real: LONGREAL; place: INTEGER) : String ;


END ConvStringLong.
```

4.4.57 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 ;


PROCEDURE InitGroup () : Group ;
PROCEDURE KillGroup (g: Group) : Group ;
PROCEDURE GetKey (g: Group; a: SYSTEM.ADDRESS) : CARDINAL ;
PROCEDURE PutKey (g: Group; a: SYSTEM.ADDRESS; key: CARDINAL) ;
PROCEDURE DelKey (g: Group; a: SYSTEM.ADDRESS) ;
PROCEDURE IsIn (g: Group; a: SYSTEM.ADDRESS) : BOOLEAN ;


END RTentity.
```

4.4.58 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}
  *)

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.
  *)

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.
  *)

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.
  *)

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.
```

```
  *)

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.
```

4.4.59 gm2-libs-iso/LongStr

```
DEFINITION MODULE LongStr;

  (* LONGREAL/string conversions *)

IMPORT
   ConvTypes;

TYPE
   (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)
   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}
*)

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.
  *)

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.
  *)

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
     in the whole number part and with an exponent that is a
     multiple of three.
  *)
```

```
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.
  *)

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.
```

4.4.60 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 = CARDINAL;

PROCEDURE AllocateSource(VAR newSource: ExceptionSource);
  (* Allocates a unique value of type ExceptionSource *)

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.
  *)

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.
  *)

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.
  *)

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.
  *)

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.
```

4.4.61 gm2-libs-iso/ErrnoCategory

```
DEFINITION MODULE ErrnoCategory ;

(*
    Description: 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.
*)

PROCEDURE IsErrnoHard (e: INTEGER) : BOOLEAN ;


(*
   IsErrnoSoft - returns TRUE if the value of errno is associated with
                 a soft device error.
*)

PROCEDURE IsErrnoSoft (e: INTEGER) : BOOLEAN ;


(*
   UnAvailable - returns TRUE if the value of errno indicates that
                 the resource or device is unavailable for some
                 reason.
*)

PROCEDURE UnAvailable (e: INTEGER) : BOOLEAN ;


(*
   GetOpenResults - maps errno onto the ISO Modula-2 enumerated
                    type, OpenResults.
*)

PROCEDURE GetOpenResults (e: INTEGER) : ChanConsts.OpenResults ;
```

```
END ErrnoCategory.
```

4.4.62 gm2-libs-iso/RealStr

```
DEFINITION MODULE RealStr;

  (* REAL/string conversions *)

IMPORT
  ConvTypes;

TYPE
  (* strAllRight, strOutOfRange, strWrongFormat, strEmpty *)
  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}
*)

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.
  *)

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.
  *)

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.
  *)
```

```
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.
  *)

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.
```

4.4.63 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     = SYSTEM.ADDRESS; (* Used to pass data between processes *)
  Body          = PROC;           (* Used as the type of a process body *)
  Urgency       = INTEGER;        (* Used by the internal scheduler *)
  Sources       = CARDINAL;       (* Used to identify event sources *)
  ProcessesExceptions =           (* Exceptions raised by this module *)
    (passiveProgram, processError);

(* The following procedures create processes and switch control between
   them. *)

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.
  *)

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.
  *)

PROCEDURE StopMe ();
  (* Terminates the calling process.
     The process must not be associated with a source of events.
  *)

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.
  *)

PROCEDURE Activate (procId: ProcessId);
  (* Causes the process identified by procId to enter the ready
     state, and thus to become eligible to run again.
  *)

PROCEDURE SuspendMeAndActivate (procId: ProcessId);
  (* Executes an atomic sequence of SuspendMe() and
     Activate(procId). *)

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).
  *)

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.
*)

PROCEDURE Attach (eventSource: Sources);
  (* Associates the specified eventSource with the calling
     process. *)

PROCEDURE Detach (eventSource: Sources);
  (* Dissociates the specified eventSource from the program. *)

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.
  *)
```

```
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.
*)

PROCEDURE Me (): ProcessId;
  (* Returns the identity of the calling process (as assigned
     when the process was first created).
  *)

PROCEDURE MyParam (): Parameter;
  (* Returns the value specified as procParams when the calling
     process was created. *)

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. *)

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.
  *)

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.
  *)

END Processes.
```

4.4.64 gm2-libs-iso/IOResult

```
DEFINITION MODULE IOResult;

  (* Read results for specified channels *)

IMPORT IOConsts, IOChan;

TYPE
  ReadResults = IOConsts.ReadResults;

  (*
    ReadResults =  (* This type is used to classify the result of an input operatio
    (
      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 *)
    );
  *)

PROCEDURE ReadResult (cid: IOChan.ChanId): ReadResults;
  (* Returns the result for the last read operation on the channel cid. *)

END IOResult.
```

4.4.65 gm2-libs-iso/M2RTS

```
DEFINITION MODULE M2RTS ;

(*
   Description: Implements the run time system facilities of Modula-2.
*)

FROM SYSTEM IMPORT ADDRESS ;


(*
   ExecuteTerminationProcedures - calls each installed termination
                                  procedure in reverse order.
*)

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.
*)

PROCEDURE InstallTerminationProcedure (p: PROC) : BOOLEAN ;


(*
   ExecuteInitialProcedures - executes the initial procedures installed
                              by InstallInitialProcedure.
*)

PROCEDURE ExecuteInitialProcedures ;


(*
   InstallInitialProcedure - installs a procedure to be executed just
                             before the BEGIN code section of the main
                             program module.
*)

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
          same effect as calling ExitOnHalt with the same code and
          then calling HALT with no parameter.
*)

PROCEDURE HALT ([exitcode: INTEGER = -1]) ;


(*
   Halt - provides a more user friendly version of HALT, which takes
          four parameters to aid debugging.
*)

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.
*)

PROCEDURE ExitOnHalt (e: INTEGER) ;


(*
   ErrorMessage - emits an error message to stderr and then calls exit (1).
*)

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.
*)

PROCEDURE IsTerminating () : BOOLEAN ;
```

```
(*
   HasHalted - Returns true if a call to HALT has been made and false
               otherwise.
*)

PROCEDURE HasHalted () : BOOLEAN ;


(*
   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.
*)

PROCEDURE Length (a: ARRAY OF CHAR) : CARDINAL ;


(*
   The following are the runtime exception handler routines.
*)

PROCEDURE AssignmentException (filename: ADDRESS; line, column: CARDINAL; scope: AD
PROCEDURE IncException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE DecException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE InclException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE ExclException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE ShiftException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS
PROCEDURE RotateException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRES
PROCEDURE StaticArraySubscriptException (filename: ADDRESS; line, column: CARDINAL;
PROCEDURE DynamicArraySubscriptException (filename: ADDRESS; line, column: CARDINAL
PROCEDURE ForLoopBeginException (filename: ADDRESS; line, column: CARDINAL; scope:
PROCEDURE ForLoopToException (filename: ADDRESS; line, column: CARDINAL; scope: ADD
PROCEDURE ForLoopEndException (filename: ADDRESS; line, column: CARDINAL; scope: AD
PROCEDURE PointerNilException (filename: ADDRESS; line, column: CARDINAL; scope: AD
PROCEDURE NoReturnException (filename: ADDRESS; line, column: CARDINAL; scope: ADDR
PROCEDURE CaseException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS)
PROCEDURE WholeNonPosDivException (filename: ADDRESS; line, column: CARDINAL; scope
PROCEDURE WholeNonPosModException (filename: ADDRESS; line, column: CARDINAL; scope
PROCEDURE WholeZeroDivException (filename: ADDRESS; line, column: CARDINAL; scope:
PROCEDURE WholeZeroRemException (filename: ADDRESS; line, column: CARDINAL; scope:
PROCEDURE NoException (filename: ADDRESS; line, column: CARDINAL; scope: ADDRESS) ;


END M2RTS.
```

4.4.66 gm2-libs-iso/ConvTypes

```
DEFINITION MODULE ConvTypes;

  (* Common types used in the string conversion modules *)

TYPE
  ConvResults =     (* Values of this type are used to express the format of a stri
  (
    strAllRight,    (* the string format is correct for the corresponding conversio
    strOutOfRange,  (* the string is well-formed but the value cannot be represente
    strWrongFormat, (* the string is in the wrong format for the conversion *)
    strEmpty        (* the given string is empty *)
  );

  ScanClass =  (* Values of this type are used to classify input to finite state sc
  (
    padding,   (* a leading or padding character at this point in the scan - ignore
    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 to
  );

  ScanState =  (* The type of lexical scanning control procedures *)
    PROCEDURE (CHAR, VAR ScanClass, VAR ScanState);

END ConvTypes.
```

4.5 ULM System Libraries

4.5.1 ulm-lib-gm2/sys/SysWrite

```
DEFINITION MODULE SysWrite;

   FROM SYSTEM IMPORT ADDRESS;

   PROCEDURE Write(fd: CARDINAL; ptr: ADDRESS;
                   VAR bytecount: CARDINAL) : BOOLEAN;

END SysWrite.
```

4.5.2 ulm-lib-gm2/sys/SysOpen

```
DEFINITION MODULE SysOpen;

   FROM SYSTEM IMPORT WORD;

   (* oflag: see SystemTypes *)

   PROCEDURE Open(VAR fd: CARDINAL; filename: ARRAY OF CHAR;
                  oflag: WORD) : BOOLEAN;

   PROCEDURE OpenCreat(VAR fd: CARDINAL; filename: ARRAY OF CHAR;
       oflag: WORD; mode: CARDINAL) : BOOLEAN;

END SysOpen.
```

4.5.3 ulm-lib-gm2/sys/SysExit

```
DEFINITION MODULE SysExit;

   PROCEDURE Exit(exitCode: CARDINAL);

   PROCEDURE EnterCleanup(p: PROC);

END SysExit.
```

4.5.4 ulm-lib-gm2/sys/SysAccess

```
DEFINITION MODULE SysAccess;

   PROCEDURE Access(filename: ARRAY OF CHAR; mode: CARDINAL) : BOOLEAN;

END SysAccess.
```

4.5.5 ulm-lib-gm2/sys/SysSignal

```
DEFINITION MODULE SysSignal;

   FROM SystemTypes IMPORT Sig;

   VAR
      default, ignore: PROC;
      old: PROC; (* will be set after each successfull Signal-call *)

   PROCEDURE Signal(sig: Sig; p: PROC) : BOOLEAN;

END SysSignal.
```

4.5.6 ulm-lib-gm2/sys/SysFork

```
DEFINITION MODULE SysFork;

   (* IF pid = 0 THEN son ELSE father END *)

   PROCEDURE Fork(VAR pid: CARDINAL) : BOOLEAN;

END SysFork.
```

4.5.7 ulm-lib-gm2/sys/SysPipe

```
DEFINITION MODULE SysPipe;

   PROCEDURE Pipe(VAR ReadFileDesc, WriteFileDesc: CARDINAL) : BOOLEAN;

END SysPipe.
```

4.5.8 ulm-lib-gm2/sys/SysRead

```
DEFINITION MODULE SysRead;

   FROM SYSTEM IMPORT ADDRESS;

   PROCEDURE Read(fd: CARDINAL; ptr: ADDRESS;
                  VAR bytecount: CARDINAL) : BOOLEAN;

END SysRead.
```

4.5.9 ulm-lib-gm2/sys/SysTime

```
DEFINITION MODULE SysTime; (* AFB 4/84 *)

   FROM SystemTypes IMPORT TIME;

   PROCEDURE Time(VAR t: TIME) : BOOLEAN;

END SysTime.
```

4.5.10 ulm-lib-gm2/sys/SysIoctl

```
DEFINITION MODULE SysIoctl;

   (* SUN version *)

   FROM SYSTEM IMPORT BYTE, BITSET;

   CONST
      shift = 0;
      Tandem = { shift + 15 };
      Cbreak = { shift + 14 };
      Lcase  = { shift + 13 };
      Echo   = { shift + 12 };
      Crmod  = { shift + 11 };
      Raw    = { shift + 10 };
      Oddp   = { shift + 9 };
      Evenp  = { shift + 8 };
      Anyp   = Oddp + Evenp;
      Nldelay = { shift + 6 , shift + 7 };
      Tbdelay = { shift + 4 , shift + 5 };
      Xtabs  = { shift + 4 , shift + 5 };
      Crdelay = { shift + 2 , 3 };
      Vtdelay = { shift + 1 };
      Bsdelay = { shift + 0 };
      Alldelay = Bsdelay + Vtdelay + Crdelay +
                 Xtabs + Tbdelay + Nldelay;

      IocVoid = { 2 };
      IocOut = { 1 };
      IocIn = { 0 };
      IocInOut = IocIn + IocOut;

      getd = BITSET(0) + IocOut;
      setd = BITSET(1) + IocIn;
      hpcl = BITSET(2) + IocVoid;
      modg = BITSET(3) + IocOut;
      mods = BITSET(4) + IocIn;
      getp = BITSET(8) + IocOut;
      setp = BITSET(9) + IocIn;
      setn = BITSET(10) + IocIn;
      excl = BITSET(13) + IocVoid;
      nxcl = BITSET(14) + IocVoid;
      flush = BITSET(16) + IocIn;
      setc = BITSET(17) + IocIn;
      getc = BITSET(18) + IocOut;
      (* BSD or SUN specific ioctl-calls *)
```

```
lbis = BITSET(127) + IocIn;
lbic = BITSET(126) + IocIn;
lset = BITSET(125) + IocIn;
lget = BITSET(124) + IocOut;
sbrk = BITSET(123) + IocVoid;
cbrk = BITSET(122) + IocVoid;
cdtr = BITSET(120) + IocVoid;
gprgp = BITSET(119) + IocOut;
sprgp = BITSET(118) + IocIn;
sltc = BITSET(117) + IocIn;
gltc = BITSET(116) + IocOut;
outq = BITSET(115) + IocOut;
sti = BITSET(114) + IocIn;
notty = BITSET(113) + IocVoid;
pkt = BITSET(112) + IocIn;
stop = BITSET(111) + IocVoid;
start = BITSET(110) + IocVoid;
mset = BITSET(109) + IocIn;
mbis = BITSET(108) + IocIn;
mbic = BITSET(107) + IocIn;
mget = BITSET(106) + IocOut;
remote = BITSET(105) + IocIn;
gwinsz = BITSET(104) + IocOut;
swinsz = BITSET(103) + IocIn;
ucntl = BITSET(102) + IocIn;

SizeOfSgttyb = 6; (* size of corresponding C-structures *)
SizeOfTchars = 6;
SizeOfWinsize = 8;

(* values of Sgttyb ispeed and ospeed *)
b0    =  0;
b50   =  1;
b75   =  2;
b110  =  3;
b134  =  4;
b150  =  5;
b200  =  6;
b300  =  7;
b600  =  8;
b1200 =  9;
b1800 = 10;
b2400 = 11;
b4800 = 12;
b9600 = 13;
exta  = 14;
extb  = 15;
```

```
  TYPE

     Sgttyb =
        RECORD
           ispeed: CHAR;
           ospeed: CHAR;
           erase: CHAR;
           kill: CHAR;
           flags: BITSET;
        END;

     Tchars =
        RECORD
           intrc: CHAR;
           quitc: CHAR;
           startc: CHAR;
           stopc: CHAR;
           eofc: CHAR;
           brkc: CHAR;
        END;

       Winsize =
 RECORD
    rows, cols: CARDINAL;
    xpixels, ypixels: CARDINAL; (* not used *)
 END;

  PROCEDURE Ioctl(fd: CARDINAL; request: BITSET;
                  VAR argp: ARRAY OF BYTE;
  argpsize: CARDINAL) : BOOLEAN;
     (* argpsize: size of corresponding C-structure *)

  PROCEDURE Stty(fd: CARDINAL; argp: Sgttyb) : BOOLEAN;

  PROCEDURE Gtty(fd: CARDINAL; VAR argp: Sgttyb) : BOOLEAN;

  PROCEDURE Isatty(fd: CARDINAL) : BOOLEAN;

  PROCEDURE GetWinsize(fd: CARDINAL; VAR winbuf: Winsize) : BOOLEAN;

  PROCEDURE Baudrate(speed: CHAR) : CARDINAL;

END SysIoctl.
```

4.5.11 ulm-lib-gm2/sys/SysExec

```
DEFINITION MODULE SysExec;

   FROM SYSTEM IMPORT ADDRESS;

   PROCEDURE Exec(name: ARRAY OF CHAR; argv: ADDRESS);

   PROCEDURE Exece(name: ARRAY OF CHAR; argv, envp: ADDRESS);

END SysExec.
```

4.5.12 ulm-lib-gm2/sys/SysLink

```
DEFINITION MODULE SysLink;

   PROCEDURE Link(name1, name2: ARRAY OF CHAR) : BOOLEAN;

END SysLink.
```

4.5.13 ulm-lib-gm2/sys/SysWait

```
DEFINITION MODULE SysWait;

   PROCEDURE Wait(VAR pid, status: CARDINAL) : BOOLEAN;

END SysWait.
```

4.5.14 ulm-lib-gm2/sys/SysFcntl

```
DEFINITION MODULE SysFcntl;

   FROM SYSTEM IMPORT WORD;

   TYPE
      FcntlRequest = (dupfd, getfd, setfd, getfl, setfl,getown,
setown,getlk,setlk,setlkw);

   PROCEDURE Fcntl(fd: CARDINAL; cmd: FcntlRequest; VAR arg: WORD) : BOOLEAN;

END SysFcntl.
```

4.5.15 ulm-lib-gm2/sys/SysUnlink

```
DEFINITION MODULE SysUnlink;

   PROCEDURE Unlink(name: ARRAY OF CHAR) : BOOLEAN;

END SysUnlink.
```

4.5.16 ulm-lib-gm2/sys/SysLseek

```
DEFINITION MODULE SysLseek;

   FROM SystemTypes IMPORT OFF;

   PROCEDURE Lseek(fd: CARDINAL; offset: OFF;
                   whence: CARDINAL) : BOOLEAN;

   PROCEDURE Tell(fd: CARDINAL; VAR offset: OFF) : BOOLEAN;

END SysLseek.
```

4.5.17 ulm-lib-gm2/sys/SysSetuid

```
DEFINITION MODULE SysSetuid;

   PROCEDURE Setuid(uid: CARDINAL) : BOOLEAN;

   PROCEDURE Setgid(gid: CARDINAL) : BOOLEAN;

END SysSetuid.
```

4.5.18 ulm-lib-gm2/sys/SysDup

```
DEFINITION MODULE SysDup;

   PROCEDURE Dup(fd: CARDINAL; VAR newfd: CARDINAL) : BOOLEAN;

   PROCEDURE Dup2(fd, newfd: CARDINAL) : BOOLEAN;

END SysDup.
```

4.5.19 ulm-lib-gm2/sys/SystemTypes

```
DEFINITION MODULE SystemTypes; (* and constants *)

   (* see...
/usr/include/fcntl.h
/usr/include/signal.h
/usr/include/sys/dir.h
/usr/include/sys/param.h
/usr/include/sys/types.h
   *)

   CONST
      DirSize = 255;
      MaxOpenFiles = 128;
      (* file control options; arguments of fcntl(2) and open(2) *)
      rdonly = {};
      wronly = { 31 };
      rdwr = { 30 };
      ndelay = { 29 };
      append = { 28 };
      async = { 25 };
      creat = { 22 };
      trunc = { 21 };
      excl = { 20 };
      nbio = { 19 };
      sync = { 18 };
   TYPE
      Sig = (SIG0,                                                    (*  0 *)
     SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGTRAP, SIGIOT, SIGEMT, (*  7 *)
     SIGFPE, SIGKILL, SIGBUS, SIGSEGV, SIGSYS, SIGPIPE,        (* 13 *)
     SIGALRM, SIGTERM, SIGUSR1, SIGUSR2, SIGCHLD, SIGPWR,      (* 19 *)
     SIGWINCH, SIGURG, SIGPOLL, SIGSTOP, SIGTSTP, SIGCONT,     (* 25 *)
     SIGTTIN, SIGTTOU, SIGVTALRM, SIGPROF, SIGXCPU, SIGXFSZ,   (* 31 *)
     SIGWAITING, SIGLWP, SIGFREEZE, SIGTHAW,                   (* 35 *)
     SIGRT36, SIGRT37, SIGRT38, SIGRT39, SIGRT40, SIGRT41,     (* 41 *)
     SIGRT42, SIGRT43);                                        (* 43 *)
   CONST
      (* aliases *)
      SIGABRT = SIGIOT;
      SIGCLD = SIGCHLD;
      SIGIO = SIGPOLL;
      SIGRTMIN = SIGRT36;
      SIGRTMAX = SIGRT43;

   TYPE
      SigSet = SET OF Sig;
```

```
        ProcessId = INTEGER; (* ProcessId may be -1 for kill *)
        TIME = LONGINT;
        OFF = LONGINT; (* offset/size of files *)

END SystemTypes.
```

4.5.20 ulm-lib-gm2/sys/SysPause

```
DEFINITION MODULE SysPause;

   PROCEDURE Pause;

END SysPause.
```

4.5.21 ulm-lib-gm2/sys/SysBreak

```
DEFINITION MODULE SysBreak;

   FROM SYSTEM IMPORT ADDRESS;

   PROCEDURE Break(addr: ADDRESS) : BOOLEAN;

   PROCEDURE Sbreak(incr: CARDINAL) : ADDRESS;

END SysBreak.
```

4.5.22 ulm-lib-gm2/sys/SysPanic

```
DEFINITION MODULE SysPanic;

   (* print text on stderr and abort with HALT *)

   PROCEDURE Panic(text: ARRAY OF CHAR);

END SysPanic.
```

4.5.23 ulm-lib-gm2/sys/SysTermIO

```
DEFINITION MODULE SysTermIO;

   IMPORT termios ;

   TYPE
      ControlChar = termios.ControlChar ;
      Flag = termios.Flag ;

      ControlCharRange = [MIN(ControlChar)..MAX(ControlChar)];

      Modes = SET OF Flag ;

      TermIO = RECORD
                  modes   : Modes ;
                  baud    : CARDINAL ;
                  cc      : ARRAY ControlCharRange OF CHAR ;
                  rows,
                  columns,
                  line    : CARDINAL ;
               END;

   PROCEDURE SetTermIO(fd: CARDINAL; termio: TermIO) : BOOLEAN;

   PROCEDURE GetTermIO(fd: CARDINAL; VAR termio: TermIO) : BOOLEAN;

   PROCEDURE Baudrate(termio: TermIO) : CARDINAL;

   PROCEDURE Isatty(fd: CARDINAL) : BOOLEAN;

END SysTermIO.
```

4.5.24 ulm-lib-gm2/sys/Sys

```
DEFINITION MODULE Sys;

(*
   The constants included here are those which are needed to get
   the ulm modules working. This is a much reduced list from the
   original ULM library. Please email <gm2 at glam.ac.uk> if you
   require more system calls than those presented below:
*)

CONST
    access = 0 ;
    brk = 1 ;
    close = 2 ;
    creat = 3 ;
    dup = 4 ;
    execve = 5 ;
    exit = 6 ;
    fcntl = 7 ;
    fstat = 8 ;
    getdents = 9 ;
    getgid = 10 ;
    getpid = 11 ;
    gettimeofday = 12 ;
    getuid = 13 ;
    ioctl = 14 ;
    kill = 15 ;
    link = 16 ;
    lseek = 17 ;
    open = 18 ;
    pause = 19 ;
    pipe = 20 ;
    read = 21 ;
    setitimer = 22 ;
    setgid = 23 ;
    setuid = 24 ;
    stat = 25 ;
    times = 26 ;
    unlink = 27 ;
    wait = 28 ;
    write = 29 ;

END Sys.
```

4.5.25 ulm-lib-gm2/sys/SysGetpid

```
DEFINITION MODULE SysGetpid; (* mh 11/1987 *)

   FROM SystemTypes IMPORT ProcessId;

   PROCEDURE Getpid(): ProcessId;

END SysGetpid.
```

4.5.26 ulm-lib-gm2/sys/SysGetuid

```
DEFINITION MODULE SysGetuid;

   PROCEDURE Getuid() : CARDINAL;

   PROCEDURE Geteuid() : CARDINAL;

   PROCEDURE Getgid() : CARDINAL;

   PROCEDURE Getegid() : CARDINAL;

END SysGetuid.
```

4.5.27 ulm-lib-gm2/sys/SysClose

```
DEFINITION MODULE SysClose;

   PROCEDURE Close(fd: CARDINAL) : BOOLEAN;

END SysClose.
```

4.5.28 ulm-lib-gm2/sys/SysAlarm

```
DEFINITION MODULE SysAlarm;

   VAR
      previous: CARDINAL; (* previous amount *)

   PROCEDURE Alarm(sec: CARDINAL) : BOOLEAN;

END SysAlarm.
```

4.5.29 ulm-lib-gm2/sys/Errno

```
DEFINITION MODULE Errno;

   (* following constants have been extracted from
         /usr/include/sys/errno.h
      on SunOS 5.5.1 at 1997/02/26
   *)

   CONST
      EPERM           =   1;  EL3RST          =  40;  ELIBSCN         =  85;
      ENOENT          =   2;  ELNRNG          =  41;  ELIBMAX         =  86;
      ESRCH           =   3;  EUNATCH         =  42;  ELIBEXEC        =  87;
      EINTR           =   4;  ENOCSI          =  43;  EILSEQ          =  88;
      EIO             =   5;  EL2HLT          =  44;  ENOSYS          =  89;
      ENXIO           =   6;  EDEADLK         =  45;  ELOOP           =  90;
      E2BIG           =   7;  ENOLCK          =  46;  ERESTART        =  91;
      ENOEXEC         =   8;  ECANCELED       =  47;  ESTRPIPE        =  92;
      EBADF           =   9;  ENOTSUP         =  48;  ENOTEMPTY       =  93;
      ECHILD          =  10;  EDQUOT          =  49;  EUSERS          =  94;
      EAGAIN          =  11;  EBADE           =  50;  ENOTSOCK        =  95;
      ENOMEM          =  12;  EBADR           =  51;  EDESTADDRREQ    =  96;
      EACCES          =  13;  EXFULL          =  52;  EMSGSIZE        =  97;
      EFAULT          =  14;  ENOANO          =  53;  EPROTOTYPE      =  98;
      ENOTBLK         =  15;  EBADRQC         =  54;  ENOPROTOOPT     =  99;
      EBUSY           =  16;  EBADSLT         =  55;  EPROTONOSUPPORT = 120;
      EEXIST          =  17;  EDEADLOCK       =  56;  ESOCKTNOSUPPORT = 121;
      EXDEV           =  18;  EBFONT          =  57;  EOPNOTSUPP      = 122;
      ENODEV          =  19;  ENOSTR          =  60;  EPFNOSUPPORT    = 123;
      ENOTDIR         =  20;  ENODATA         =  61;  EAFNOSUPPORT    = 124;
      EISDIR          =  21;  ETIME           =  62;  EADDRINUSE      = 125;
      EINVAL          =  22;  ENOSR           =  63;  EADDRNOTAVAIL   = 126;
      ENFILE          =  23;  ENONET          =  64;  ENETDOWN        = 127;
      EMFILE          =  24;  ENOPKG          =  65;  ENETUNREACH     = 128;
      ENOTTY          =  25;  EREMOTE         =  66;  ENETRESET       = 129;
      ETXTBSY         =  26;  ENOLINK         =  67;  ECONNABORTED    = 130;
      EFBIG           =  27;  EADV            =  68;  ECONNRESET      = 131;
      ENOSPC          =  28;  ESRMNT          =  69;  ENOBUFS         = 132;
      ESPIPE          =  29;  ECOMM           =  70;  EISCONN         = 133;
      EROFS           =  30;  EPROTO          =  71;  ENOTCONN        = 134;
      EMLINK          =  31;  EMULTIHOP       =  74;  ESHUTDOWN       = 143;
      EPIPE           =  32;  EBADMSG         =  77;  ETOOMANYREFS    = 144;
      EDOM            =  33;  ENAMETOOLONG    =  78;  ETIMEDOUT       = 145;
      ERANGE          =  34;  EOVERFLOW       =  79;  ECONNREFUSED    = 146;
      ENOMSG          =  35;  ENOTUNIQ        =  80;  EHOSTDOWN       = 147;
      EIDRM           =  36;  EBADFD          =  81;  EHOSTUNREACH    = 148;
      ECHRNG          =  37;  EREMCHG         =  82;  EALREADY        = 149;
```

```
      EL2NSYNC        =  38;  ELIBACC         =  83;  EINPROGRESS     = 150;
      EL3HLT          =  39;  ELIBBAD         =  84;  ESTALE          = 151;

      EWOULDBLOCK     = EAGAIN;

   CONST
      maxerror = 151;
      maxmsglen = 41;
      maxnamelen = 15;

   TYPE
      ErrorNumber = [0..maxerror];
      ErrorMessage = ARRAY [0..maxmsglen] OF CHAR;
      ErrorName = ARRAY [0..maxnamelen] OF CHAR;

   VAR
      message: ARRAY ErrorNumber OF ErrorMessage;
      name: ARRAY ErrorNumber OF ErrorName;

   VAR
      errno: CARDINAL;

END Errno.
```

4.5.30 ulm-lib-gm2/sys/SysStat

```
DEFINITION MODULE SysStat;

   FROM SystemTypes IMPORT TIME, OFF;
   FROM SYSTEM IMPORT BITSET ;

   TYPE
      StatBuf =
         RECORD
            dev: CARDINAL;
            ino: CARDINAL;
            mode: BITSET;
            nlink: CARDINAL;
            uid: CARDINAL;
            gid: CARDINAL;
            rdev: CARDINAL;
            size: OFF;
            atime: TIME;
    spare1 : CARDINAL;
            mtime: TIME;
    spare2 : CARDINAL;
            ctime: TIME;
    spare3 : CARDINAL;
    blksize : CARDINAL;
    blocks : CARDINAL;
    spare4 : ARRAY[0..1] OF CARDINAL;
         END;
   CONST
      (* bit masks for mode; bits 0..15 used *)
      FileType = { 0..3 };
      (* IF Ifxxx = mode * FileType *)
      IfDir = { 1 };       (* directory *)
      IfChr = { 2 };       (* character special *)
      IfBlk = { 1..2 };    (* block special *)
      IfReg = { 0 };       (* regular *)
      IfLnk = { 0,2 };    (* symbolic link *)
      IfSock = { 0..1 };   (* socket *)
      IfFifo = { 3 };      (* fifo *)
      (* IF Isxxx <= mode THEN *)
      IsUid =  { 4 };      (* set user id on execution *)
      IsGid =  { 5 };      (* set group id on execution *)
      IsVtx =  { 6 };      (* save swapped text even after use *)
      (* permissions on file: IF ... <= mode *)
      OwnerRead = { 7 };   (* read permission, owner *)
      OwnerWrite = { 8 }; (* write permission, owner *)
      OwnerExec = { 9 };   (* execute/search permission, owner *)
```

```
      GroupRead = { 10 };
      GroupWrite = { 11 };
      GroupExec = { 12 };
      WorldRead = { 13 };
      WorldWrite = { 14 };
      WorldExec = { 15 };

   PROCEDURE Stat(file: ARRAY OF CHAR; VAR buf: StatBuf) : BOOLEAN;

   PROCEDURE Fstat(fd: CARDINAL; VAR buf: StatBuf) : BOOLEAN;

END SysStat.
```

4.5.31 ulm-lib-gm2/sys/SysLocations

```
DEFINITION MODULE SysLocations;

   FROM SYSTEM IMPORT ADDRESS;

   VAR
      ProgramEnd, Etext, Edata, Break, Environment: ADDRESS;

END SysLocations.
```

4.5.32 ulm-lib-gm2/sys/UnixString

```
DEFINITION MODULE UnixString;

   CONST
      BufSiz = 512;
   TYPE
      Buffer = ARRAY[0..BufSiz-1] OF CHAR;

   PROCEDURE Copy(VAR buf: Buffer; str: ARRAY OF CHAR);
      (* copy str to Buf and guarantee 0C-termination;
         str is silently truncated on copying if necessary
      *)

END UnixString.
```

4.5.33 ulm-lib-gm2/sys/SysCreat

```
DEFINITION MODULE SysCreat;

   PROCEDURE Creat(VAR fd: CARDINAL; filename: ARRAY OF CHAR;
                   mode: CARDINAL) : BOOLEAN;

END SysCreat.
```

4.5.34 ulm-lib-gm2/sys/SYSTEM

```
DEFINITION MODULE FOR "C" SYSTEM ;

(*
    Description: provides a SYSTEM module for GNU Modula-2 so that the
                 ULM libraries can be built.
*)

EXPORT QUALIFIED (* the following are built into the compiler: *)
                 (* SIZE is exported if -fpim2 is enabled *)
                 ADDRESS, WORD, BYTE, BITSET, ADR, TSIZE, THROW,
                 (* the rest are implemented in SYSTEM.c *)
                 UNIXCALL, UNIXSIGNAL, UNIXFORK ;


(*
   UNIXCALL - returns TRUE if the syscall was successful, results from
              the system call are returned in r0 and r1.
*)

PROCEDURE UNIXCALL (syscall: CARDINAL; VAR r0, r1: INTEGER; ...) : BOOLEAN;


(*
   UNIXFORK - returns TRUE if successful and pid is set to the son pid
              if the parent is returning. If the child is returning pid=0.
              UNIXFORK returns FALSE if an error occurs and errno is held in pid.
*)

PROCEDURE UNIXFORK (VAR pid: CARDINAL) : BOOLEAN ;


(*
   UNIXSIGNAL -
*)

PROCEDURE UNIXSIGNAL (signo: CARDINAL; p: PROC;
                      VAR old: PROC; VAR result: CARDINAL) : BOOLEAN;


END SYSTEM.
```

4.5.35 ulm-lib-gm2/sys/SysKill

```
DEFINITION MODULE SysKill; (* AFB 9/88 *)

   FROM SystemTypes IMPORT Sig, ProcessId;

   PROCEDURE Kill(pid: ProcessId; sig: Sig) : BOOLEAN;

END SysKill.
```

4.6 ULM Standard Libraries

4.6.1 ulm-lib-gm2/std/Functions

```
DEFINITION MODULE Functions; (* AFB 12/88 *)

(*
   (C) Andreas Borchert, Universitaet Ulm, 1988

   The expression grammar is Modula-2 oriented with following exceptions:

   (1) The ?: operator has been added (semantic close to C).
?: has lowest priority and right-to-left associativity
   (2) Integer constants are real constants, too.
   (3)  Operator-keywords are not supported, so use
   &  instead of AND,
   |  instead of OR, and
   ~  instead of NOT.
   (4) TRUE is represented as 1.0 and FALSE as 0.0
Anything but 0.0 is taken to be TRUE (in conditions).

   The grammar (in EBNF):

      CondExpression = Expression [ "?" CondExpression ":" CondExpression ] .
      Expression = SimpleExpression [ RelOp SimpleExpression ] .
      SimpleExpression = ["+"|"-"] Term { AddOp Term } .
      Term = Factor { MulOp Factor } .
      Factor = Constant | IDENT | FunctionCall | "~" Factor |
       "(" CondExpression ")" .
      FunctionCall = IDENT "(" CondExpression [ "," CondExpression ] ")" .
      RelOp = "=" | "#" | "<" | ">" | "<=" | ">=" .
      AddOp = "+" | "-" | "|" .
      MulOp = "*" | "/" | "&" .

   The start symbol is CondExpression.
   Identifiers are sequences of letters and digits. The first character
   must be a letter.
   Syntax of constants:

      Digit { Digit } [ "." { Digit } ] [ ("E"|"e") Digit { Digit } ]

   On errors 'ParseFunction' returns FALSE after 'errpos' has been set
   to the error position.
   Errors can result from

      (1) Syntax errors.
      (2) Bad constants, e.g. exponent is too large
```

```
      (3) Bad number of arguments to a function

  Each identifier not declared as a constant or a function is
  considered to be a parameter. The value of a parameter is
  predefined to 0.0. 'FirstParam' and 'NextParam' give the parameter names
  in alphabetical order. Parameter values can be changed using
  'SetFuncParam'. 'EvalFunction' evaluates the function with the
  parameters set previously.

  Example:

     WriteString("func: "); ReadString(functxt);
     IF ParseFunction(func, functxt) THEN
FirstParam;
WHILE NextParam(func, parname) DO
   WriteString(parname); WriteString(" = ");
   ReadReal(val);
   SetFuncParam(func, parname, val);
END;
WriteReal(EvalFunction(func), 1); WriteLn;
     ELSE
(* error at errpos *)
     END;

  Warning:

  EvalFunction does not check for division by zero or any other
  operations which can result in a floating point exception.

  Hint:

  Import 'StdFuncs' for having a standard set of functions and
  constants.

*)

  TYPE
     Function;
     Real = REAL;
     StdFunc1 = PROCEDURE (Real) : Real;
     StdFunc2 = PROCEDURE (Real, Real) : Real;

  VAR
     errpos: CARDINAL; (* error position in expr of ParseFunction *)

  PROCEDURE InstallStdFunc1(funcname: ARRAY OF CHAR; stdfunc: StdFunc1);
```

```
   PROCEDURE InstallStdFunc2(funcname: ARRAY OF CHAR; stdfunc: StdFunc2);

   PROCEDURE InstallStdConst(constname: ARRAY OF CHAR; val: Real);

   PROCEDURE ParseFunction(expr: ARRAY OF CHAR; VAR func: Function) : BOOLEAN;

   PROCEDURE FirstParam(func: Function);

   PROCEDURE NextParam(func: Function; VAR symname: ARRAY OF CHAR) : BOOLEAN;

   PROCEDURE SetFuncParam(func: Function; parname: ARRAY OF CHAR;
  value: Real);

   PROCEDURE EvalFunction(func: Function) : Real;

   PROCEDURE DisposeFunction(VAR func: Function);
      (* release storage associated with 'func' *)

END Functions.
```

4.6.2 ulm-lib-gm2/std/Storage

```
DEFINITION MODULE Storage;              (* A. Borchert *)

   FROM SYSTEM IMPORT ADDRESS;

   PROCEDURE ALLOCATE(VAR ptr:ADDRESS; size: CARDINAL);

   PROCEDURE DEALLOCATE(VAR ptr:ADDRESS; size: CARDINAL);

   PROCEDURE Setmode(m: CARDINAL);

END Storage.
```

4.6.3 ulm-lib-gm2/std/Environment

```
DEFINITION MODULE Environment;

   PROCEDURE GetEnv(name: ARRAY OF CHAR; (* parameter name to be looked for *)
    VAR text: ARRAY OF CHAR; (* parameter contents *)
    VAR ok: BOOLEAN);

   PROCEDURE EnvPar(index: CARDINAL; (* ranging [0.. #parameters-1] *)
    VAR text: ARRAY OF CHAR; (* "name=contents" *)
    VAR ok: BOOLEAN);

END Environment.
```

4.6.4 ulm-lib-gm2/std/MathLib

```
DEFINITION MODULE MathLib;

   PROCEDURE arctan(x: REAL) : REAL;

   PROCEDURE exp(x: REAL) : REAL;

   PROCEDURE ln(x: REAL) : REAL;

   PROCEDURE sin(x: REAL) : REAL;

   PROCEDURE cos(x: REAL) : REAL;

   PROCEDURE sqrt(x: REAL) : REAL;

END MathLib.
```

4.6.5 ulm-lib-gm2/std/PipeIO

```
DEFINITION MODULE PipeIO;

   FROM StdIO IMPORT FILE, MODE;

   PROCEDURE Popen(VAR f: FILE; cmd: ARRAY OF CHAR; mode: MODE;
                   buffered: BOOLEAN) : BOOLEAN;

   PROCEDURE Pclose(f: FILE) : BOOLEAN;

END PipeIO.
```

4.6.6 ulm-lib-gm2/std/InOut

```
DEFINITION MODULE InOut; (* stripped version: AFB 4/84 *)

   CONST
      EOL = 12C;
   VAR
      Done: BOOLEAN; (* on eof true *)
      termCH: CHAR; (* set in ReadString and numeric input procs *)

   PROCEDURE Read(VAR ch: CHAR);

   PROCEDURE ReadString(VAR str: ARRAY OF CHAR);

   PROCEDURE ReadCard(VAR arg: CARDINAL);

   PROCEDURE ReadInt(VAR arg: INTEGER);

   PROCEDURE Write(ch: CHAR);

   PROCEDURE WriteLn;

   PROCEDURE WriteString(s: ARRAY OF CHAR);

   (* n: minimum field width *)

   PROCEDURE WriteInt(x: INTEGER; n: CARDINAL);

   PROCEDURE WriteCard(x: CARDINAL; n: CARDINAL);

   PROCEDURE WriteOct(x: CARDINAL; n: CARDINAL);

   PROCEDURE WriteHex(x: CARDINAL; n: CARDINAL);

END InOut.
```

4.6.7 ulm-lib-gm2/std/Calendar

```
DEFINITION MODULE Calendar;

   FROM SystemTypes IMPORT TIME;

(*
 * Date calculations with
 * (a) Julius Caesar's calendar since Jan 01, 0001
 * (b) the Gregorian calendar  since Oct 15, 1582
 * (c) Xelos system time.
 *
 * Martin Hasch, University of Ulm, Jan 1988
 *)

   TYPE
      Time         = TIME;     (* consecutive seconds *)
      Date         = LONGCARD; (* consecutive days *)

      Year         = CARDINAL;
      Month        = [1..12];
      Day          = [1..31];
      Hour         = [0..23];
      Minute       = [0..59];
      Second       = [0..59];
      Weekday      = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
      Week         = [1..53];
      Yearday      = [1..366];

      Daytime      = RECORD
hour:    Hour;
minute:  Minute;
second:  Second;
     END;
      Calendarday  = RECORD
year:    Year;
month:   Month;
day:     Day;
     END;
      CalendarInfo = RECORD
weekday: Weekday;
week:   Week;
yearday: Yearday;
     END;

   PROCEDURE CurrentTime(): Time;
   (*
```

```
    * returns actual system time = seconds since Jan 1, 1970, 00:00:00 GMT
    *)

   PROCEDURE ConvertTime(time: Time; VAR date: Date; VAR daytime: Daytime);
   PROCEDURE ConvertDate(date: Date; VAR calendarday: Calendarday);
   PROCEDURE ConvertCald(calendarday: Calendarday; VAR info: CalendarInfo);

   PROCEDURE CTime   (date: Date; daytime: Daytime):       Time;
   PROCEDURE CDate   (year: Year; month: Month; day: Day): Date;
   PROCEDURE CUltimo (year: Year; month: Month):     Date;
   PROCEDURE CWeekday(date: Date):     Weekday;

   PROCEDURE DateOK(year, month, day: CARDINAL): BOOLEAN;

   PROCEDURE TimeToString(time: Time; VAR string: ARRAY OF CHAR);
   (*
    * converts time to a string, e.g. "Sun Sep 16 01:03:52 1973 GMT"
    *)

   PROCEDURE SetFirstOfWeek(weekday: Weekday);
   (*
    * important for week of year calculation in ConvertCald; default is Mon.
    *)

   PROCEDURE GetTimezone (VAR tzName: ARRAY OF CHAR);
   PROCEDURE SetTimezone (    tzName: ARRAY OF CHAR);
   PROCEDURE GetLocaltime(VAR delay: Time);
   PROCEDURE SetLocaltime(    delay: Time);
   (*
    * important for CTime, ConvertTime and TimeToString.
    *)

END Calendar.
```

4.6.8 ulm-lib-gm2/std/Conversions

```
DEFINITION MODULE Conversions;                  (* LG *)

  PROCEDURE ConvertOctal(num, len: CARDINAL; VAR str: ARRAY OF CHAR);
    (* conversion of an octal number to a string *)

  PROCEDURE ConvertHex(num, len: CARDINAL; VAR str: ARRAY OF CHAR);
    (* conversion of a hexadecimal number to a string *)

  PROCEDURE ConvertCardinal(num, len: CARDINAL; VAR str: ARRAY OF CHAR);
    (* conversion of a cardinal decimal number to a string *)

  PROCEDURE ConvertInteger(num: INTEGER; len: CARDINAL;
                           VAR str: ARRAY OF CHAR);
    (* conversion of an integer decimal number to a string *)

END Conversions.
```

4.6.9 ulm-lib-gm2/std/RealConv

```
DEFINITION MODULE RealConv; (* AFB 6/84 * rev. wsc 2/85 *)

   TYPE
      ReadProc = PROCEDURE(VAR CHAR);
   VAR
      Done: BOOLEAN;
      termCH: CHAR;

   PROCEDURE ReadReal(Read: ReadProc; VAR x: REAL);
   (* convention: Read returns 0C on eof or error *)

   PROCEDURE WriteFloat(VAR field: ARRAY OF CHAR; x: REAL; cbase: CARDINAL;
                        dp: CARDINAL);
   PROCEDURE WriteFix(VAR field: ARRAY OF CHAR; x: REAL; cbase: CARDINAL;
                      VAR dp: CARDINAL);

END RealConv.
```

4.6.10 ulm-lib-gm2/std/Terminal

```
DEFINITION MODULE Terminal; (* A. Borchert *)

   (* read and write from/to standard input/output channel *)

   VAR Done: BOOLEAN;

   PROCEDURE Read(VAR ch: CHAR);

   PROCEDURE ReadAgain;

   PROCEDURE Write(ch: CHAR);

   PROCEDURE WriteLn;

   PROCEDURE WriteString(s: ARRAY OF CHAR);

END Terminal.
```

4.6.11 ulm-lib-gm2/std/StrSpec

```
DEFINITION MODULE StrSpec; (* gsk 1/85 *)

   PROCEDURE StrPartCpy ( VAR target           : ARRAY OF CHAR;
      source           : ARRAY OF CHAR;
      position, number : CARDINAL );

   PROCEDURE StrDel ( VAR target           : ARRAY OF CHAR ;
  position, number : CARDINAL );

   PROCEDURE StrIns ( VAR target      : ARRAY OF CHAR;
  insertion   : ARRAY OF CHAR;
  position    : CARDINAL );

   PROCEDURE StrPos ( source, search : ARRAY OF CHAR ) : CARDINAL;

END StrSpec.
```

4.6.12 ulm-lib-gm2/std/StrToReal

```
DEFINITION MODULE StrToReal;

   PROCEDURE StrToReal(str: ARRAY OF CHAR; VAR real: REAL) : BOOLEAN;
      (* converts str to the REAL real, leading white space is
 ignored, returns FALSE if str does not conform to following
 syntax:
    ["+" | "-"] digit { digit } ["." digit { digit } ]
    ["E" ["+" | "-"] digit [digit] ]
      *)

END StrToReal.
```

4.6.13 ulm-lib-gm2/std/SysPerror

```
DEFINITION MODULE SysPerror; (* AFB 2/84 *)

   PROCEDURE Perror(str: ARRAY OF CHAR);

   PROCEDURE GetErrorString(errno: CARDINAL; VAR str: ARRAY OF CHAR);

END SysPerror.
```

4.6.14 ulm-lib-gm2/std/Passwd

```
DEFINITION MODULE Passwd;
(*
 * scanning and searching the passord file
 *
 * Martin Hasch, University of Ulm, Nov-29-1988
 *)

   TYPE
      Pwent =
 RECORD
    logname:  ARRAY [0..7] OF CHAR;
    password: ARRAY [0..15] OF CHAR;
    uid:      CARDINAL;
    gid:      CARDINAL;
    fullname: ARRAY [0..31] OF CHAR;
    dir:      ARRAY [0..31] OF CHAR;
    shell:    ARRAY [0..31] OF CHAR;
 END;

   PROCEDURE OpenPw(filename: ARRAY OF CHAR): BOOLEAN;
   (* returns TRUE on success *)

   PROCEDURE GetPwent(VAR pwent: Pwent): BOOLEAN;

   PROCEDURE GetPwuid(uid: CARDINAL; VAR pwent: Pwent): BOOLEAN;

   PROCEDURE GetPwnam(logn: ARRAY OF CHAR; VAR pwent: Pwent): BOOLEAN;

   PROCEDURE ReopenPw(): BOOLEAN;
   (* returns TRUE if passwd file is open and seekable *)

   PROCEDURE ClosePw(): BOOLEAN;
   (* returns TRUE if passwd file was open *)


   PROCEDURE FetchPwuid(uid: CARDINAL; VAR pwent: Pwent): BOOLEAN;
   (* implies OpenPw("/etc/passwd"), and ClosePw() *)

   PROCEDURE FetchPwnam(logn: ARRAY OF CHAR; VAR pwent: Pwent): BOOLEAN;
   (* implies OpenPw("/etc/passwd"), and ClosePw() *)

END Passwd.
```

4.6.15 ulm-lib-gm2/std/Directories

```
DEFINITION MODULE Directories;

   FROM SystemTypes IMPORT DirSize, OFF;

   TYPE
      DIR;

      FileName = ARRAY [0..DirSize-1] OF CHAR;
      Direct =
         RECORD
            ino: CARDINAL;
            name: FileName;
         END;

   PROCEDURE OpenDir(VAR dirp: DIR; filename: ARRAY OF CHAR) : BOOLEAN;

   PROCEDURE ReadDir(dirp: DIR; VAR direct: Direct) : BOOLEAN;

   PROCEDURE TellDir(dirp: DIR; VAR offset: OFF) : BOOLEAN;

   PROCEDURE SeekDir(dirp: DIR; pos: OFF) : BOOLEAN;

   PROCEDURE RewindDir(dirp: DIR) : BOOLEAN;

   PROCEDURE CloseDir(VAR dirp: DIR);

END Directories.
```

4.6.16 ulm-lib-gm2/std/Strings

```
DEFINITION MODULE Strings; (* AFB 7/84 *)

   PROCEDURE StrLen(s: ARRAY OF CHAR) : CARDINAL;

   PROCEDURE StrCat(VAR s: ARRAY OF CHAR; s1: ARRAY OF CHAR);

   PROCEDURE StrCmp(s1, s2: ARRAY OF CHAR) : INTEGER;

   PROCEDURE StrCpy(VAR s: ARRAY OF CHAR; s1: ARRAY OF CHAR);

END Strings.
```

4.6.17 ulm-lib-gm2/std/TimeIO

```
DEFINITION MODULE TimeIO; (* AFB 9/88 *)

   FROM Calendar IMPORT Time, Date;
   FROM StdIO IMPORT FILE;

   TYPE
      Style = (date, (* date(1) and ctime(3) standard format *)
       ls, (* like the ls-command *)
       env); (* see for TIMEFMT in environment *)
   VAR
      Done: BOOLEAN;
      termCH: CHAR;

   PROCEDURE WriteTime(format: ARRAY OF CHAR; time: Time);
      (* the output format is very close to date(1): *)
      (* each field descriptor is preceded by % and will be *)
      (* replaced in the output by its corresponding value. *)
      (* WriteTime does not append a newline automatically  *)
      (* like date(1).                                      *)
      (* output is directed to StdIO.stdout                 *)

   PROCEDURE FwriteTime(file: FILE; format: ARRAY OF CHAR; time: Time);
      (* like WriteTime but output is directed to file      *)

   PROCEDURE SwriteTime(VAR string: ARRAY OF CHAR;
format: ARRAY OF CHAR;
time: Time);
      (* like WriteTime but output is put into string       *)

   PROCEDURE WriteTimeLike(style: Style; time: Time);
      (* write time to StdIO.stdout according to the given  *)
      (* style.                                             *)

   PROCEDURE FwriteTimeLike(file: FILE; style: Style; time: Time);

   PROCEDURE SwriteTimeLike(VAR string: ARRAY OF CHAR;
    style: Style; time: Time);

   PROCEDURE ReadTime(VAR time: Time);
      (* read time from StdIO.stdin *)

   PROCEDURE FreadTime(file: FILE; VAR time: Time);

   PROCEDURE SreadTime(string: ARRAY OF CHAR; VAR time: Time);
```

```
   PROCEDURE WriteDate(format: ARRAY OF CHAR; date: Date);

   PROCEDURE FwriteDate(file: FILE; format: ARRAY OF CHAR; date: Date);

   PROCEDURE SwriteDate(VAR string: ARRAY OF CHAR;
format: ARRAY OF CHAR; date: Date);

   PROCEDURE ReadDate(VAR date: Date);

   PROCEDURE FreadDate(file: FILE; VAR date: Date);

   PROCEDURE SreadDate(string: ARRAY OF CHAR; VAR date: Date);

   (* Reading  depends on a set of pattern describing valid      *)
   (* input formats. This formats are stored in an ordered list. *)
   (* If more than one pattern matches the input the first will  *)
   (* be chosen.                                                 *)
   (* Pattern consists of a sequence of letters and some special *)
   (* chars which must match the input. Whitespace (except nl)   *)
   (* is skipped by ReadTime and must not be given inside a      *)
   (* pattern.                                                   *)
   (* Legal Letters:                                             *)
   (*   'y': year, 'm': month,  'd': day                         *)
   (*   'H': hour, 'M': minute, 'S': second                      *)
   (* Examples:                                                  *)
   (*   m/d/yH:M:S        us-date, matches 10/22/86 13:12:14     *)
   (*   d.m.yH:M:S        german date, matches 22.10.86 13:12:14 *)
   (*   md,y              matches Oct 22, 1986                   *)

   PROCEDURE Append(pattern: ARRAY OF CHAR);
      (* appends a new pattern to the end of the list *)

   PROCEDURE Insert(pattern: ARRAY OF CHAR);
      (* inserts a pattern before the beginning of the list *)

   PROCEDURE ReleaseList;
      (* causes the list to be emptied *)

   PROCEDURE DefaultList;
      (* appends a list of standard patterns to the list *)
      (* this procedure is called during initialization of TimeIO *)

END TimeIO.
```

4.6.18 ulm-lib-gm2/std/Files

```
DEFINITION MODULE Files;

   FROM SystemTypes IMPORT OFF;

   (* high level module for file handling *)

   IMPORT StdIO;

   TYPE
      FILE = StdIO.FILE;

   VAR Done: BOOLEAN;

   PROCEDURE OpenRead(VAR f: FILE; filename: ARRAY OF CHAR);

   PROCEDURE OpenWrite(VAR f: FILE; filename: ARRAY OF CHAR);

   PROCEDURE Close(f: FILE);

   PROCEDURE SetPos(f: FILE; pos: OFF);

   PROCEDURE GetPos(f: FILE; VAR pos: OFF);

   PROCEDURE Reset(f: FILE);

   PROCEDURE Delete(filename: ARRAY OF CHAR);

   PROCEDURE Rename(oldname, newname: ARRAY OF CHAR);

END Files.
```

4.6.19 ulm-lib-gm2/std/CallShell

```
DEFINITION MODULE CallShell;

   PROCEDURE Shell(cmd: ARRAY OF CHAR; VAR status: CARDINAL) : BOOLEAN;

END CallShell.
```

4.6.20 ulm-lib-gm2/std/Archive

```
DEFINITION MODULE Archive; (* AFB 3/84 *)

   FROM SYSTEM IMPORT BITSET ;
   FROM SystemTypes IMPORT TIME, OFF;

   (* routines for reading an archive file *)

   CONST NameLength = 14;

   TYPE AFILE; (* hidden *)
      FileName = ARRAY[0..NameLength-1] OF CHAR;
      AStat =
         RECORD
            name: FileName;
            uid, gid: CARDINAL;
            date: TIME;
            size: OFF;
            mode: BITSET;
    offset: OFF; (* absolute offset in archive file *)
         END;

   PROCEDURE ArchiveOpen(VAR a: AFILE; archive: ARRAY OF CHAR;
 filename: ARRAY OF CHAR) : BOOLEAN;

   PROCEDURE ArchiveReopen(a: AFILE; filename: ARRAY OF CHAR) : BOOLEAN;
   (* in case of an error "a" will be closed *)

   PROCEDURE ArchiveClose(a: AFILE);

   PROCEDURE ArchiveRead(a: AFILE; VAR ch: CHAR) : BOOLEAN;

   PROCEDURE ArchiveStat(a: AFILE; VAR buf: AStat);

END Archive.
```

4.6.21 ulm-lib-gm2/std/StdIO

```
DEFINITION MODULE StdIO; (* AFB 1/84 *)

   FROM SYSTEM IMPORT ADDRESS;
   FROM SystemTypes IMPORT OFF;

   TYPE
      FILE; (* hidden *)
      MODE = (read, write, append);

   VAR
      stdin, stdout, stderr: FILE;

   (* all functions return FALSE in error case *)

   PROCEDURE Fopen(VAR f: FILE; name: ARRAY OF CHAR; mode: MODE;
   buffered: BOOLEAN) : BOOLEAN;

   PROCEDURE Fclose(f: FILE) : BOOLEAN;

   PROCEDURE Fread(ptr: ADDRESS; size: CARDINAL; VAR nitems: CARDINAL;
   f: FILE) : BOOLEAN;

   PROCEDURE Fwrite(ptr: ADDRESS; size: CARDINAL; VAR nitems: CARDINAL;
    f: FILE) : BOOLEAN;

   PROCEDURE Fseek(f: FILE; offset: OFF; whence: CARDINAL) : BOOLEAN;

   PROCEDURE Ftell(f: FILE; VAR pos: OFF) : BOOLEAN;

   PROCEDURE Feof(f: FILE) : BOOLEAN;

   PROCEDURE Ferror(f: FILE) : BOOLEAN;

   PROCEDURE Fgetc(VAR ch: CHAR; f: FILE) : BOOLEAN;

   PROCEDURE Fputc(ch: CHAR; f: FILE) : BOOLEAN;

   PROCEDURE Fungetc(ch: CHAR; f: FILE) : BOOLEAN;

   PROCEDURE CloseAll() : BOOLEAN;

   PROCEDURE Fflush(f: FILE) : BOOLEAN;

   PROCEDURE Fdopen(VAR f: FILE; filedesc: CARDINAL; mode: MODE;
                    buffered: BOOLEAN) : BOOLEAN;
```

```
   PROCEDURE FileNo(f: FILE) : CARDINAL;

END StdIO.
```

4.6.22 ulm-lib-gm2/std/FtdIO

```
DEFINITION MODULE FtdIO;

   FROM SYSTEM IMPORT WORD;
   FROM StdIO IMPORT FILE;

   VAR Done: BOOLEAN;
       termCH: CHAR;

   PROCEDURE FreadInt(f: FILE; VAR arg: INTEGER);

   PROCEDURE FwriteInt(f: FILE; arg: INTEGER; w: CARDINAL);

   PROCEDURE FreadCard(f: FILE; VAR arg: CARDINAL);

   PROCEDURE FwriteCard(f: FILE; arg: CARDINAL; w: CARDINAL);

   PROCEDURE FreadString(f: FILE; VAR str: ARRAY OF CHAR);

   PROCEDURE FwriteString(f: FILE; str: ARRAY OF CHAR);

   PROCEDURE FwriteLn(f: FILE);

   PROCEDURE Fread(f: FILE; VAR arr: ARRAY OF WORD);

   PROCEDURE Fwrite(f: FILE; arr: ARRAY OF WORD);

   PROCEDURE FreadWord(f: FILE; VAR w: WORD);

   PROCEDURE FwriteWord(f: FILE; w: WORD);

   PROCEDURE FreadChar(f: FILE; VAR ch: CHAR);

   PROCEDURE FwriteChar(f: FILE; ch: CHAR);

END FtdIO.
```

4.6.23 ulm-lib-gm2/std/StdFuncs

```
DEFINITION MODULE StdFuncs;

END StdFuncs.
```

4.6.24 ulm-lib-gm2/std/ASCII

```
DEFINITION MODULE ASCII;

   CONST

      (* control characters *)

      nul = 0C;   ack = 6C;   ff  = 14C;  dc2 = 22C;  can = 30C;  rs  = 36C;
      soh = 1C;   bel = 7C;   cr  = 15C;  dc3 = 23C;  em  = 31C;  us  = 37C;
      stx = 2C;   bs  = 10C;  so  = 16C;  dc4 = 24C;  sub = 32C;  sp  = 40C;
      etx = 3C;   ht  = 11C;  si  = 17C;  nak = 25C;  esc = 33C;
      eot = 4C;   lf  = 12C;  dle = 20C;  syn = 26C;  fs  = 34C;
      enq = 5C;   vt  = 13C;  dc1 = 21C;  etb = 27C;  gs  = 35C;

      (* other usual names *)

      null = nul;
      bell = bel;
      nl   = lf; (* new line *)
      tab  = ht;
      np   = ff; (* new page *)

      del  = 177C;

END ASCII.
```

4.6.25 ulm-lib-gm2/std/SysConf

```
DEFINITION MODULE SysConf; (* AFB 2/97 *)

   (* configuration parameters of the installation *)

   PROCEDURE GetLibDir(VAR libdirBuf: ARRAY OF CHAR);
      (* GetLibDir returns the directory where the Modula-2 library
         has been installed to; it does not honour the MODLIB
 environment variable
      *)

   PROCEDURE GetRelease(VAR releaseBuf: ARRAY OF CHAR);
      (* returns the release of the Modula-2 installation *)

END SysConf.
```

4.6.26 ulm-lib-gm2/std/RandomGenerator

```
DEFINITION MODULE RandomGenerator;

   (*    Anyone who considers arithmetical
methods of producing random digits
 is, of course, in a state of sin.
 - John von Neumann (1951)
   *)

   PROCEDURE IntVal() : INTEGER;
      (* get random 32-bit value *)

   PROCEDURE RealVal() : REAL;
      (* get a uniformly distributed real value in [0..1) *)

   PROCEDURE Random(low, high: INTEGER) : INTEGER;
      (* get a uniformly distributed integer in [low..high] *)

   PROCEDURE Flip() : BOOLEAN;
      (* return TRUE or FALSE *)

   PROCEDURE Init(seed: INTEGER);

END RandomGenerator.
```

4.6.27 ulm-lib-gm2/std/EtcGroup

```
DEFINITION MODULE EtcGroup;
(*
 * scanning and searching the etc/group file
 *
 * Martin Hasch, University of Ulm, Dec-06-1988
 *)

  TYPE
     MemberList = POINTER TO Member;
     Member =
 RECORD
    logname:  ARRAY [0..7] OF CHAR;
    nextmem:  MemberList;
 END;

     Grent =
 RECORD
    grname:   ARRAY [0..7] OF CHAR;
    password: ARRAY [0..15] OF CHAR;
    gid:      CARDINAL;
    members:  MemberList; (* NIL-terminated *)
 END;

  PROCEDURE OpenGr(filename: ARRAY OF CHAR): BOOLEAN;
  (* returns TRUE on success *)

  PROCEDURE GetGrent(VAR grent: Grent): BOOLEAN;

  PROCEDURE GetGrgid(gid: CARDINAL; VAR grent: Grent): BOOLEAN;

  PROCEDURE GetGrnam(grn: ARRAY OF CHAR; VAR grent: Grent): BOOLEAN;

  PROCEDURE ReopenGr(): BOOLEAN;
  (* returns TRUE if group file is open and seekable *)

  PROCEDURE CloseGr(): BOOLEAN;
  (* returns TRUE if group file was open *)


  PROCEDURE FetchGrgid(gid: CARDINAL; VAR grent: Grent): BOOLEAN;
  (* implies OpenGr("/etc/group"), and CloseGr() *)

  PROCEDURE FetchGrnam(grn: ARRAY OF CHAR; VAR grent: Grent): BOOLEAN;
  (* implies OpenGr("/etc/group"), and CloseGr() *)
```

```
END EtcGroup.
```

4.6.28 ulm-lib-gm2/std/GetPass

```
DEFINITION MODULE GetPass;

   PROCEDURE GetPass(prompt: ARRAY OF CHAR;
                     VAR passwd: ARRAY OF CHAR);

END GetPass.
```

4.6.29 ulm-lib-gm2/std/RTErrors

```
DEFINITION MODULE RTErrors;

   FROM SYSTEM IMPORT ADDRESS;

   TYPE
      Kind = (noError, halt, case, stack, crend, prio, fret, range);
      RangeCheck = (none, unsigned, signed, sign, dyn);

      Error = POINTER TO ErrorRec;
      ErrorRec =
 RECORD
    kind: Kind;
    module: ADDRESS; (* points to name of module *)
    line: CARDINAL; (* line number in source *)
    pc: ADDRESS; (* program counter *)
    CASE (* kind *) : Kind OF
    | range: CASE rtype: RangeCheck OF
     | unsigned, dyn:
   value, min, max: CARDINAL;
     | signed:
   ivalue, imin, imax: INTEGER;
                     ELSE
     END;
            ELSE
    END;
 END;

      Handler = PROCEDURE (Error);

   PROCEDURE Notify(error: Error);
      (* called by runtime system *)

   PROCEDURE SetHandler(newHandler: Handler);
      (* define alternative handler of runtime errors *)

END RTErrors.
```

4.6.30 ulm-lib-gm2/std/Clock

```
DEFINITION MODULE Clock;

   FROM SystemTypes IMPORT TIME;

   CONST UnitsPerSecond = 100;

   PROCEDURE RealTime(reset: BOOLEAN): TIME;
   PROCEDURE CPUTime (reset: BOOLEAN): TIME;
   (*
    * These functions return the time in units elapsed since the start
    * of the current process or since the last call with argument TRUE.
    *)

END Clock.
```

4.6.31 ulm-lib-gm2/std/Arguments

```
DEFINITION MODULE Arguments; (* mh 5/85 *)
(* rev mh 6/88 *)
(*
 * This module reads options and other arguments from the command
 * line. An argument "-" or "--" stops option reading. "-", how-
 * ever, will be delivered as argument then, whereas "--" will not.
 *)

(*
 * Example:
 *
 * xflag := FALSE;
 * string := defaultstring;
 * number := 1;
 * InitArgs("[-x] [-s string] [-nnn] [file]...");
 * WHILE GetFlag(flag) DO
 *    CASE flag OF
 *       "x":  xflag := TRUE;
 *    |  "s":  FetchString(string);
 *    |  "0".."9":
 *             UngetOpt;
 *             FetchCard(number);
 *       ELSE  Usage
 *    END;
 * END; (*WHILE GetFlag*)
 * WHILE GetArg(filename) DO
 *    IF StrCmp(filename,"-") = 0 THEN
 *       (* process stdin *)
 *    ELSE
 *       (* process filename *)
 *    END;
 * END; (*WHILE GetArg*)
 *)

   PROCEDURE InitArgs(is: ARRAY OF CHAR);
(* specifies infoString and (re)starts the reading cyclus *)

   PROCEDURE Usage;
(* prints 'Usage: command infoString' on stderr and aborts
 * program execution. FetchString, FetchCard and FetchInt call
 * this procedure automatically in case of errors.
 *)

   PROCEDURE GetFlag(VAR flag: CHAR): BOOLEAN;
(* tries to read one flag, i.e. a character within a string containing
```

```
 * a leading '-',from the argument list and returns TRUE if successful.
 *)

   PROCEDURE GetOpt( VAR flag: CHAR; VAR plus: BOOLEAN): BOOLEAN;
(* reads one character within a string starting in '+' or '-'.
 *)

   PROCEDURE FetchString(VAR string: ARRAY OF CHAR);
(* The procedures FetchXXX try to read data of type XXX from
 * the argument list.
 *)

   PROCEDURE FetchCard(  VAR number: CARDINAL);
(* syntax of cardinal arguments:  [+]{digit}  *)

   PROCEDURE FetchInt(   VAR number: INTEGER);
(* syntax of integer arguments:  [+|-]{digit}  *)

   PROCEDURE FetchOct(   VAR number: CARDINAL);
(* syntax of octal arguments:  [+]{octdigit}  *)

   PROCEDURE FetchHex(   VAR number: CARDINAL);
(* syntax of hexadecimal arguments:  [+]{hexdigit}  *)

   PROCEDURE GetArg(VAR arg: ARRAY OF CHAR): BOOLEAN;
(* reads one argument or returns FALSE if all are read. *)

   PROCEDURE UngetArg;
(* pushes the argument that has been read just before
 * back to the argument list.
 *)

   PROCEDURE UngetOpt;
(* pushes the flag or option that has been read just before
 * back to the argument list.
 *)

   PROCEDURE AllArgs;
(* calls 'Usage' if any arguments are not yet read. *)

END Arguments.
```

4.6.32 ulm-lib-gm2/std/RealInOut

```
DEFINITION MODULE RealInOut; (* AFB 6/84 * rev. wsc 2/85 *)

   FROM StdIO IMPORT FILE;

   VAR
      Done: BOOLEAN;

   (*
    * Read REAL number x according to syntax:
    *
    * ["+" | "-"] digit { digit } ["." digit { digit } ]
    * ["E" ["+" | "-"] digit [digit] ]
    *
    * Done := "a number was read".
    *
    * at most 16 digits are significant, leading zeroes not
    * counting. Maximum exponent is 76. Input terminates
    * with a blank or any control character.
    *)

   PROCEDURE ReadReal(VAR x: REAL);

   PROCEDURE FreadReal(f: FILE; VAR x: REAL);

   (*
    * Write x using n characters. If fewer than n characters
    * are needed, leading blanks are inserted.
    *)

   PROCEDURE WriteReal(x: REAL; n: CARDINAL);

   PROCEDURE FwriteReal(f: FILE; x: REAL; n: CARDINAL);

   (*
    * Write x in fixed point notation using pd digits in front
    * of decimal point and dp digits behind decial point. If
    * fewer than pd digits are needed, leading blanks are
    * inserted.
    *)

   PROCEDURE WriteFloat(x: REAL; pd: CARDINAL; dp: CARDINAL);

   PROCEDURE FwriteFloat(f: FILE; x: REAL; pd: CARDINAL; dp: CARDINAL);

   (*
```

```
    * Write x in octal/hexadecimal form with exponent and mantissa
    *)

   PROCEDURE WriteRealOct(x: REAL);

   PROCEDURE FwriteRealOct(f: FILE; x: REAL);

   PROCEDURE WriteRealHex(x: REAL);

   PROCEDURE FwriteRealHex(f: FILE; x: REAL);

END RealInOut.
```

4.6.33 ulm-lib-gm2/std/ScanPwfile

```
DEFINITION MODULE ScanPwfile;
(*
 * utility functions for modules Passwd and EtcGroup
 *
 * Martin Hasch, University of Ulm, Dec-06-1988
 *)

   FROM StdIO IMPORT FILE;
   FROM ASCII IMPORT nl;

   CONST
      fieldsep = ":";
      linesep = nl;

   PROCEDURE ReRead(pwfile: FILE): BOOLEAN;

   PROCEDURE GetText(pwfile: FILE; VAR text: ARRAY OF CHAR; sepchar: CHAR): BOOLEAN

   PROCEDURE GetNumber(pwfile: FILE; VAR number: CARDINAL; sepchar: CHAR): BOOLEAN;

END ScanPwfile.
```

4.6.34 ulm-lib-gm2/std/ReadIntCard

```
DEFINITION MODULE ReadIntCard;

   FROM SYSTEM IMPORT WORD;

   TYPE
      Type = (int, card);
      ReadProc = PROCEDURE(VAR CHAR);

   VAR Done: BOOLEAN;

   (* convention: ReadChar returns 0C on eof or error *)

   PROCEDURE Read(VAR w: WORD; t: Type; ReadChar: ReadProc);

END ReadIntCard.
```

4.6.35 ulm-lib-gm2/std/Plot

```
DEFINITION MODULE Plot;

   FROM StdIO IMPORT FILE;

   (* device independent plotter interface; see plot(3) and plot(5) *)

   PROCEDURE OpenPlot(f: FILE);

   PROCEDURE ClosePlot;

   PROCEDURE Move(xto, yto: INTEGER);

   PROCEDURE Cont(xto, yto: INTEGER);

   PROCEDURE Point(xpoint, ypoint: INTEGER);

   PROCEDURE Line(xfrom, yfrom, xto, yto: INTEGER);

   PROCEDURE String(str: ARRAY OF CHAR);

   PROCEDURE Arc(xcenter, ycenter, xstart, ystart, xend, yend: INTEGER);

   PROCEDURE Circle(xcenter, ycenter, radius: INTEGER);

   PROCEDURE Erase;

   PROCEDURE LineMod(style: ARRAY OF CHAR);

   PROCEDURE Space(xupleft, yupleft, xlowright, ylowright: INTEGER);

   PROCEDURE Reverse(xupleft, yupleft, xlowright, ylowright: INTEGER);

   PROCEDURE Polygon(xcenter, ycenter, xstart, ystart, edges: INTEGER);

   PROCEDURE CharMod(plotchar: CHAR);

END Plot.
```

4.6.36 ulm-lib-gm2/std/StrToNum

```
DEFINITION MODULE StrToNum; (* mh 5/85; rev afb 4/86: StrToOct/StrToHex *)

   PROCEDURE StrToCard(str: ARRAY OF CHAR; VAR card: CARDINAL): BOOLEAN;
(* converts str to the CARDINAL card. Leading spaces, tabs and new-
 * lines are ignored. Returns FALSE if str is not of the syntax:
 *   [+] {digit} , or if the resulting number exceeds CARDINAL range.
 *)

   PROCEDURE StrToInt(str: ARRAY OF CHAR; VAR integ: INTEGER): BOOLEAN;
(* converts str to the INTEGER integ in analogue manner.
 * Required syntax of str here:  [+|-] {digit} .
 *)

   PROCEDURE StrToOct(str: ARRAY OF CHAR; VAR card: CARDINAL) : BOOLEAN;

   PROCEDURE StrToHex(str: ARRAY OF CHAR; VAR card: CARDINAL) : BOOLEAN;

END StrToNum.
```

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Short Contents

Table of Contents