Mumps Programming Language Interpreter, Compiler, and C++ Class Library User's Guide Including Sqlite Global Array Database Storage Facility

Kevin C. O'Kane

kc.okane@gmail.com

http://threadsafebooks.com/
http://www.cs.uni.edu/~okane/

June 15, 2024

Table of Contents

Index of Figures

1 Installation

1.1 Installation Overview

1.2 Interpreter vs Compiler

The compiler supports most of the features supported by the interpreter. The compiler generates a C++ file which is subsequently compiled with Mumps run-time libraries. Compiler code is usually faster but since both use the same database engine, database intensive programs run at about the same speed.

The compiler translates Mumps to C++. Consequently, compiled programs can be edited as C++ programs and other code introduced which might not be available in original Mumps.

1.2.1 Interpreting a Program

To run a Mumps program (file suffix: *.mps*) with the interpreter, either:

1. Type

   mumps fileName.mps

   where *fileName.mps* is the name of the Mumps program, or,

2. Place on the first line, first column of the Mumps program the line:

   ```
   #!/usr/bin/mumps
   ```

   Make the Mumps program executable, and invoke the program by typing its name (including file extension):

   filename.mps

1.2.2 Compiling a Program

The script file *mumpsc* if given a Mumps program (file suffix: *.mps*) as an operand compiles the Mumps to C++ and then compiles the C++ resulting in an executable binary with the same name as the input Mumps program.

When compiling a Mumps program, *mumpsc* generates a C++ file (file suffix: *.cpp*) which is the C++ translation of the Mumps program. It is this file that is passed to the C++ (g++) compiler along with Mumps run-time libraries.

You may edit the C++ file and include calls to other routines. You may compile it (the C++) file) to a binary executable using *mumpsc*.

You should not pass the Mumps compiler generated C++ file directly to the C++ compiler due to required libraries which the *mumpsc* command automatically includes.

If you use the compiler, you should avoid using the **xecute** command and the indirection operator (**@**). These invoke the interpreter and increase overhead.

1.3 Required System Software

Building mumps requires that your system have certain software installed. For the most part, these are available through the Synaptic Package Manager or *apt-get*. The *Configure...* scripts automatically install these if they are not present on your system.

1. Linux, preferably a Debian based version such as Debian, Ubuntu or Mint. The Windows WSL (Windows Subsystem for Linux) implementation with Ubuntu may be used.

2. The *g++/gcc* compilers and related libraries.

3. The *pcre* (Perl Compatible Regular Expression) development libraries. The pcre libs should be in */usr/lib* and the include files in */usr/include*. Be certain to install the *pcre* **development** libraries.

4. The *bash shell* interpreter located in */bin*.

5. The *GNU readline* and *readline-dev* packages.

6. *autoconf*

7. The following libraries are needed for the extended precision mathematics. If they are not installed by default, you will need to do so. Be sure to install the **development** versions of the libraries:

   a) The GNU Multiple precision floating point computation library

   ```
   http://www.mpfr.org/
   libmpfr-dev
   ```

   b) The GNU Multiprecision arithmetic library development tools

   ```
   https://gmplib.org/
   libgmp-dev
   ```

Script files are provided containing the *apt* commands necessary to install required software.

1.4 Basic Software Installation

There are Bash script files (see below) that will install any needed software. You should use these to install the software packages required.

Do not run configure / make. There are script files that do these which correctly set the parameters.

The names of these script files all begin with the prefix ***Configure....*** The *configure...* files contain the *apt* commands to install needed software. There is one for the native database and one for the SQLite database.

Note: A related set of files to compile the various versions, begin with the prefix ***Compile....***

The *apt* tool installs required software used by Debian GNU/Linux and related distributions (such as Ubuntu and Mint). Other Linux systems use different but similar tools. You need to install these packages for all versions of Mumps that you intend to use. The script files here work only with Debian related systems.

1.5 Native Database

You must be **root** to run the following.

To configure the native Mumps database, type:

./ConfigureNativeMumps.script

This will install any needed software. Next type:

./CompileNativeSingleUserMumps.script

or

./CompileNativeSharedMumps.script

The first compiles the single user native database and is the one most often used. The second installs a shared version of the database which is slower. Only one database version may be installed at a time.

1.6 SQLite3 Software

You must be **root** to run the following.

The command to install neede software for the SQLite version are in:

./ConfigureSqlite.script

so you don’t need to run them manually.

Next type:

./CompileSqliteMumps.script

which will compile and install the SQLite version. Only one database version may be isntalled at a time.

1.7 Building the Software

The distribution consists of source code. The source code must be compiled and linked to create executable versions of the interpreter.

There are several options that must be set before compilation. These can be set manually. The script file *configure.ac*[1] contains all the options. The file *configure.ac* is used to create the file *configure.*

However, for the most part, you should use the *Compile...* script files:

CompileNativeSharedMumps.script

CompileNativeSingleUserMumps.script

CompileSqliteMumps.script

These will invoke *configure*, configure the source code, and build the resulting executables according to pre-set templates. You may change the configuration options by making changes to these files. These are discussed below. If you wish to change a configuration option, edit these files.

You must be *root* to run the configuration and compile scripts.

1.7.1 Quick Start

If you want to build the most basic version of the Mumps interpreter, see the following. Compile time options are shown in section 1.11.2 .

To build the simplest and fastest version, the Native Database Single User version, as **root**, type:

./ConfigureNativeMumps.script

./Compile NativeSingleUserMumps.script

The first script file installs any necessary software and the second compiles and builds the most basic version of the interpreter. If you have already installed the necessary software, the first step is not needed. You must be **root** to run these scripts.

The resulting interpreter is named *mumps* and is located in /usr/bin/mumps.

The single user native data base is fastest but only one user may use a set of database files at a time. The Shared Native is next fastest and permits multiple users to share the same data base files. The slowest is based on Sqlite3 but it provides for the greatest data base integrity and permits the data base to be accessed/viewed in a relational context.

1. *configure* is a program that edits the source files to set paramater, limits, file names, and so forth before the files are compiled. The file *configure.ac* becomes *configure* when you run the program *autoconf.*

1.7.1.1 Single User Native Database

```
ConfigureNativeMumps.script
```

followed by:

```
CompileNativeSingleUserMumps.script
```

1.7.1.2 Shared (Multi-User) Native Database

```
ConfigureNativeMumps.script
```

followed by:

```
CompileNativeSharedMumps.script
```

1.7.1.3 Multi-User Sqlite3 Database

```
ConfigureSqliteMumps.script
```

followed by:

```
CompileSqliteMumps.script
```

1.7.2 Native Database Options

The native database option is fast with a minimum of overhead and it can efficiently manage very large databases however it lacks a number of features normally found on modern database systems:

1. It is sensitive to system crashes and programming errors.
2. It does a minimum of checkpointing.
3. It maintains part of the global array tree in volatile memory.

If the host system crashes or the program using the global arrays terminates unexpectedly, the contents of the entire global array database are likely to be lost.

However, in applications where speed is important and, in the event of a crash, the program can be re-run, the native database is a good choice.

The native database has two configurations:

1. The first of these is a *single user* global array facility where the global arrays are stored in one directory, usually the one in which the Mumps program is itself running. In this mode, only one *read-write*[2] Mumps program may access the global arrays in a given directory at a time although other Mumps programs may run concurrently in other directories operating on other global array data sets. This is the fastest but most restrictive option. The single user version also contains a *read-only* version that permits multiple instances of Mumps to access the database concurrently provided no version concurrent version is *read-write.*

2. The native database also has a *shared* option. In this version, multiple instances of Mumps may concurrently access the database in read-write mode. This option is slower than the single user version.

The native database is stored in the current directory in files named *key.dat* and *data.dat.* Database files created by the single user version may be used by the shared version (but not concurrently) and vice versa.

1.7.3 Sqlite3 Database Option

If data integrity, remote and multi-user access are important, the Sqlite is better. In this option, the interpreter and compiler use Sqlite3 to store the global arrays.

2 The native database Mumps comes in two versions: a *read-write* version which may both read and write global arrays and an *read-only* version where each Mumps program may only read the global arrays. Multiple *read-only* instances may operate concurrently on the same global array data sets.

While option 2 is slower than option 1, due to relational data base system overhead, using a relational database has *significant advantages* with regard to reliability and flexibility. These include:

1. All database transactions are ACID (*Atomicity, Consistency, Isolation, Durability*) compliant.
2. SQL commands such as Begin Transaction, Commit and Rollback are available.
3. The Mumps global arrays can be queried with SQL commands from non-Mumps environments.
4. SQL views of the Mumps database may be constructed.
5. The Mumps global array database can be remote and distributed.
6. Mumps programs can execute SQL commands on the server on any accessible database table.
7. Multiple concurrent Mumps programs may run at the same time.

The distribution contains scripts that will build various versions of the system. These are detailed next. You must be *root* to run these.

The scripts assume a Debian (*apt-get*) based Linux installation. If you are using a version of Linux not based on Debian, you will need to manually install and configure the required system software manually according to the procedures on your system.

Some of the scripts provided with the distribution may install system software as needed. Consequently, when using these scripts, your machine needs to have a reliable Internet connection. Also, due to Internet load factors, it is possible that software installations may take a long time or, in some cases, fail in the unlikely event that the servers from which the software to be downloaded are unavailable.

The Mumps interpreters and libraries built as a result of the scripts will be stored in */usr/bin, /usr/lib,* and */usr/include*.

1.7.4 Sqlite3 Database Server Stored Global Arrays

The Mumps global arrays my be stored in the Sqlite3 relational database system. With simple code changes, other servers could also be accommodated.

To build the Sqlite3 versions, use the scripts:

ConfigureSqliteMumps.script
CompileSqliteMumps.script

There are advantages and disadvantages to storing global arrays in a relational database. The primary disadvantage is that the hierarchical nature of the Mumps database is not well suited to the tabular structure of a relational database and overall access is slower.

On the other hand, relational databases provide flexible multi-user, robust, fully ACID (*Atomicity, Consistency, Isolation, Durability)* compliant data storage along with a complete suite of transaction processing functions not otherwise available in the Mumps language definition.

A further advantage is that global array data may be interrogated and manipulated by ordinary, standard SQL commands.

By default, the Mumps interpreter maps global array references to a multi-column relational database table normally with the name *mumps* (this can be changed by *configure*). The columns of the table are named *a1*, *a2*, … *a10* and so forth. The values in the columns are the name of the Mumps global array (in a1) and indices from a global array reference (in *a2* through *a9*)..

The final column (a10) contains the value stored at the reference, if any. For example, the code:

```
set ^birds(1,2,3,4,5)="ducks"
```

would map to a table named *mumps* in the relational database as follows[3]:

3 Columns sizes can be set to other values by *configure*. Smaller values may improve performance.

birds

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
birds	2	3	4	5					ducks

Where the values for *a6* through *a9* are *null*.

If your program instantiates array elements like the following:

```
set ^birds(1)=”all”
set ^birds(1,2)=”flying”
set ^birds(1,2,3)=”water”
set ^birds(1,2,3,4)=”large”
set ^birds(1,2,3,4,5)=”ducks”
set ^birds(1,3)=”flightless”
set ^birds(1,3,3)=”water”
set ^birds(1,3,3,4)=”large”
set ^birds(1,3,3,4,5)=”penguins”
```

The relational table will look like[4]:

a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
birds	1								all
birds	1	2							flying
birds	1	2	3						water
birds	1	2	3	3					large
birds	1	3	3	4	5				ducks
birds	1	3							flighhtless
birds	1	3	3						water
birds	1	3	3						large
birds	1	3	3	5					penguins

Mumps access requests produce the expected results:

```
write ^birds(1)              => all
write ^birds(1,2)            => flying
write ^birds(1,2,3)          => water
write ^birds(1,2,3,4)        => large
write ^birds(1,2,3,4,5)      => ducks

write $order(^birds(1,2))     => 3
write $order(^birds(1,2,"")) => 3
```

The row-wise duplication in the above is also present in many other Mumps systems and the empty columns (*nulls*) have little real effect on overall performance.

An advantage, as mentioned above, is that data stored in such a table may be queried by an ordinary SQL command such as:

```
select a10 from mumps where a1='birds' and a2=’1’ and a3='2';
```

which yields *flying*.

4 Table row order may differ but this is not important.

Similarly, SQL *views* may be established on the *Mumps* table to facilitate access in other ways by other users.

1.7.5 Basic Sqlite3 Database Configuration

By default, in order for Mumps to store and retrieve global arrays Sqlite3 there must be a database file named *mumps.sqlite* accessible to the instance of Mumps being executed (links may be used if the database file is in another directory).

You may create *mumps.sqlite* with the file *CreateSqliteDB.script* which is produced by the *configure* procedure. Options to *configure* can be sued to set the maximum number of characters per Mumps global array index and the maximum number of characters stored at the node. The defaults are 64 and 128, respectively.

1.8 Optimal Compilation Configure options

The following are the optimal recommended compile configuration options as included in the appropriate script files.

1.8.1 Single User Native Database

```
./configure prefix=/usr \
--with-cache=262145 \
--with-hardware-math \
--with-int-32 \
--with-float-digits=6 \
--with-block=2048 \
--with-slice=0 \
--with-alarm=0
```

1.8.2 Shared Native Database

```
./configure prefix=/usr \
--with-hardware-math \
--with-cache=5 \
--with-slice=10 \
--with-alarm=1 \
--with-shared \
--with-block=4096
```

1.8.3 Sqlite3 Database

```
./configure prefix=/usr \
--with-sqlite --with-dbname=mumps \
--with-slice=0 \
--with-alarm=0
```

1.9 Math Options

Arithmetic in this Mumps distribution can be performed either by hardware or by a library of extended precision software.

In extended precision mode, the precision of both floating point and integer numbers can be significantly larger than is the case with standard hardware arithmetic with minimal performance penalty.

The several Build scripts look for files *gmp.h* and *mpfr.h*. If these are found, they permit the use of the extended math packages. If not found, the builds will use hardware arithmetic.

You may override this and force hardware arithmetic by modifying the scripts to add the *--with-hardware-math* option.

1.10 Numeric Configuration Options

Both extended precision and basic hardware precision are available as noted above.

In this version of Mumps, as is the case with many others, numeric values are stored in variables as character strings. When a variable participates in an arithmetic operation, the value is converted to a numeric format, the operation performed (for example, addition), and the result converted back to character string. Not only are numeric values stored in variables as strings, but also, intermediate results are in string format.

In this version of Mumps, there are several options with regard to handling numeric data. As an option, you may process numeric data either by means of builtin hardware operations or by means of extended precision software. Hardware is quicker while extended precision permits a greater range of values. The following discusses the *configure* options available.

1.10.1 Hardware Math

In hardware math mode, integer and floating point numbers are processed by your machine's arithmetic processing hardware. Floating point numbers are treated as either *long double* or *double* values and integers are treated as either signed 64-bit *long long* or signed 32-bit *long* integer values.

To enable hardware math, you must specify the following as a *configure* option:

--with-hardware-math

Integer arithmetic may be performed in *int* (32 bit) or *long long* (64 bits in the gcc compiler) mode. The default is *long long*. The *int* mode may be turned on with the *configure* option:

--with-int-32

If the above is not specified, *long long* is used. The gcc compiler implements *long long* as 64 bits. The data type *int* is implemented as 32 bits.

Floating point arithmetic may be performed in either *long double* or *double* mode. The *long double* mode may be enabled with the *configure* option:

--with-long-double

If the above is not specified, floating point arithmetic will be performed in *double* mode.

All numeric values are stored internally as strings. They are converted to binary numeric integer or floating point format just prior to an arithmetic operation and then converted back to strings.

By default, the string format of a floating point number will have with 8 digits of precision. This can be altered by *configure* using the *–with-float_digits* option (default is 8). For example, if you want 16 digits of precision, add

--with-float-digits=16

to the *configure* parameters. The number of digits specified should be consistent with the hardware data type (*double* or *long double*).

On x86 architectures, *long double* is usually implemented as an 80 bit number with a sign bit, an 15 bit exponent and 63 bit fractional part with a range of approximately 3.65×10^{-4951} to 1.18×10^{4932} while *double* is implemented as a 64 bit number.

1.10.2 Extended Precision Math

Extended precision is available through use of the GNU multiple precision arithmetic library[5] and the GNU MPFR library[6]. For integers, this means effectively unlimited precision. For floating point numbers, the exponent is 64 bits and the fraction is user specified (default of 72 bits in Mumps - this option may be set by *configure*).

Hardware arithmetic will be selected during compilation of the interpreter if (1) *configure* does ***not*** find the extended precision libraries or (2) the user affirmatively specifies the configuration option:

--with-hardware-math

5 http://www.mpfr.org/
6 http://gmplib.org/manual/index.html

If extended precision is used, the number of bits in the fraction of a floating point number can be set with:

--with-float-bits=value

where *value* is the number of bits. The default value is 72. The number of decimal digits for a given number of bits (nbits) is approximately:

$$\log_{10}(2^{nbits})$$

Thus, 72 bits corresponds to approximately 21 decimal digits.

For extended precision floating point numbers, the number of digits of precision to print is controlled by:

--with-float-digits=value

where *value* is the number of digits. The default is 8.

The number of digits specified should be consistent with the number of bits in the fraction. If the number of digits specified is too large, random low-order digits will appear in numbers.

If extended precision mode is in effect, integer numbers have no upper or lower bound.

1.11 All Configure Options

The basic install sequence, as is the case with many Linux based packages is to run something similar to the following as *root*:

```
./configure prefix=/usr
./make
./make install
```

You should NOT do this.

The configure step, however, as is typical, contains many options. Specifying these causes modification to the source code and changes the final product. These must be set correctly.

The distribution, as noted above, contains several *bash* script files with pre-configured *configure* commands. For the most part, you probably don't want to write your own *configure* options except in limited cases. You may, however, want to edit the files provided to set details such as passwords and so on. This is discussed below.

The full set of options to *configure* are:

1.11.1 configure prefix=/usr

The directory where the runtime modules will be stored. If this is not specified, the default location is in a directory named *mumps_compiler* in the user's home directory. Normally, if you want Mumps available to all users, you will specify the option as shown and run *make* and *make install* as *root*. If you specify */usr* as shown, the Mumps routines will be placed in */usr/bin/mumps*.

1.11.2 General Relational Database Options

1.11.2.1 --with-dbname=name

Default name of the Sqlite3 mumps database table name [default: *mumps*].

1.11.2.2 --with-index_size=number

Maximum number of characters in each Sqlite3 global array index [default: 64]

1.11.2.3 --with-data_size=nbr

Maximum number of data characters stored for an Sqlite3 global array reference (final column) [default: 128]

1.11.2.4 --with-dbfile=name

Name of Sqlite's database file stored in the users directory [default: mumps.sqlite]

1.11.2.5 --with-slice=value

When using Sqlite3 or the single user native database, this number should be zero.

For the shared native database, a value of zero will cause the database files to be finalized after each global array transaction. This results in slower but safer operation.

For the shared native database, if this number is a positive integer, it is the number of milliseconds for the database to sleep when a time slice has expired (see *--with-alarm*). This allows other pending instances of mumps to gain access to the database. [default: 10]

1.11.2.6 --with-server

Compile the native database in shared (server) mode. This value should be zero for single user native and Sqlite databases. It is used with in connection with a pipe version of the global database not currently supported.

1.11.2.7 --with-alarm=value

The time interval of a database time slice in seconds. During a time slice, parts of the native database are maintained in memory. [default: 1]

If *--with-slice* is zero, this value should be set to zero.

This value should be zero for Sqlite3 and single user native modes.

1.11.2.8 --with-cache=VAL

Native global database in-memory cache size. The number is the number of blocks (see: *--with-block*) to maintain in memory.

The **only** legal values for this parameter are:

```
9
17
33
65
129
257
513
1025
2049
4097
8193
16385
32769
65537
131073
262145
524289
1048577
```

1.11.2.9 --with-block=blksize

Native global Btree block size.

The native Btree database consists of two files: the tree file (*key.dat*) containing the actual Btree and the data file (*data.dat*) containing stored data. The maximum size of the Btree file is dependent on the block size. The block sizes listed below each have a PAGE_SHIFT value and this ultimately determines the maximum file size as shown. The basic internal disk address is effectively 31 bits (signed 32 bit quantity) but, depending upon the block size, some number of bits at the low-order end of a block address are always zero. For example, if the block size is 1024, the final 10 bits of an address are always zeros. As only the significant 31 bits are stored, the true address is not 31 bits but 41 bits thus a file size of 2 terabytes is possible.

The only legal values for this parameter are:

```
1024
2048
4096
8192
16384
32768
65536
131072
262144
```

The block size determines the internal PAGE_SHIFT factor:

```
1024      →    PAGE_SHIFT 10
2048      →    PAGE_SHIFT 11
4096      →    PAGE_SHIFT 12
8192      →    PAGE_SHIFT 13
16384     →    PAGE_SHIFT 14
32768     →    PAGE_SHIFT 15
65536     →    PAGE_SHIFT 16
131072    →    PAGE_SHIFT 17
262144    →    PAGE_SHIFT 18
524288    →    PAGE_SHIFT 19
1048576   →    PAGE_SHIFT 20
2097152   →    PAGE_SHIFT 21

PAGE_SHIFT 10 corresponds to MBLOCK 1024 and a max Btree file size of 2 TB
PAGE_SHIFT 11 corresponds to MBLOCK 2048 and a max Btree file size of 4 TB
PAGE_SHIFT 12 corresponds to MBLOCK 4096 and a max Btree file size of 8 TB
PAGE_SHIFT 13 corresponds to MBLOCK 8192 and a max Btree file size of 16 TB
PAGE_SHIFT 14 corresponds to MBLOCK 16384 and a max Btree file size of 32 TB
PAGE_SHIFT 15 corresponds to MBLOCK 32768 and a max Btree file size of 64 TB
PAGE_SHIFT 16 corresponds to MBLOCK 65536 and a max Btree file size of 128 TB
```

The separate data file may grow to a max of 2**64 bytes for all settings.

1.11.2.10 --with-readonly

Native database will be read-only – only applies to the native global array facility. A multiple instances of a read-only version can be run concurrently.

1.11.3 --with-ibuf=

Maximum size of an interpreted program [default: 32000].

1.11.4 --with-strmax=

Maximum internal string size [default: 4096].

1.11.5 --with-locale=locale

Locale information [default: en_US.UTF-8].

1.11.6 --with-terminate-on-error

Halt interpreter on error [default: off]

1.11.7 --with-float-bits=val

Number of bits in an extended precision floating point fractional part (72).

1.11.8 --with-float-digits=val

Number of decimal digits to print in an extended precision floating point number (20).

1.11.9 --with-hardware-math

Use hardware arithmetic facilities.

1.11.10 --with-no-inline

Do not use inline functions.

1.11.11 --with-profile

Enable profiler (run *gprof mumps gmon.out > stats*).

1.11.12 --with-maxglobal=val

Maximum length of a global array reference in the native database. A global reference length includes the array name, all indices, plus parentheses and commas.

2 Running a Mumps Program

2.1 Format the Global Array Sqlite3 Server

If you are using Sqlite3, be sure you have created *mumps.sqlite* using the *CreatSqliteDB.script* file. This creates the base Sqlite database.

2.2 Mumps CLI Interpreter

To run the command line interpreter from a terminal window, type:

```
mumps
```

Any Mumps commands you enter will be executed immediately. To exit the interpreter, type H[alt] or control-d.

In interactive mode, you will be presented with a prompt (>). Any Mumps command may be typed for immediate execution (including a **goto** or **do** command with a file name reference pointing to a file to be loaded and executed).

The keyboard *up arrow* and *down arrow* keys may be used to cycle through and display commands previously entered during this session. Your may edit previously entered commands.

A previously entered command my be re-executed by using the keyboard up arrow key to locate and display the command and then typing <enter>.

Input to the Mumps CLI follows GNU *readline* conventions.

2.2.1 Mumps CLI Special Commands

2.2.1.1 halt, ^d

Exit the Mumps CLI. use the **halt** (**h**) command or **^d**.

2.3 Mumps Programs (scripts)

Mumps programs are ASCII files that can be created by any ASCII text editor. Do not use word processing editors that may embed hidden formatting characters into the text.

A Mumps script will have the following as their first line:

```
#!/usr/bin/mumps
```

The file extension of a Mumps program *.mps* is preferred but not required.

The Mumps source file must be made executable:

```
chmod u+x prog.mps
```

where *prog.mps* is the name of your mumps source file.

Example:

```
#!/usr/bin/mumps
      for i=1:1:10 do
      . write "Hello World ",i,!
      halt
```

You may execute the program by typing *prog.mps* to your terminal prompt. The program above will write *Hello World,* followed by a number ten times.

2.4 Source Code Format

C++ and C code were formatted using:

```
astyle -A6 -s6 *.cpp
```

3 Relational Database Commands & Variables

If Sqlite3 relational database storage of globals is enabled, the following functions and builtin variables are available in the Mumps interpreter. If the native database is in use, these, with the exception of **$zNative**, are ignored.

3.1 $zSqlite

$zsqlite with no arguments returns true (1) if globals are being stored in Sqlite3, false (0) otherwise.

3.2 $zSqlite("begin transaction")

Sends a *BEGIN TRANSACTION;* command to Sqlite.

3.3 $zSqlite("commit transaction")

Sends a *COMMIT TRANSACTION*; command to Sqlite.

3.4 $zSqlite("savepoint"[,*savepoint_name])*

If the second argument is omitted, send a *SAVEPOINT default;* command to Sqlite.
If the second argument is present, send a *SAVEPOINT savepoint;* command to Sqlite where '*savepoint'* is the value passed as the second argument. See Sqlite3 documentation for details.

3.5 $zSqlite("rollback"[,*savepoint*])

If the second argument is omitted, send a *ROLLBACK TRANSACTION* to default; command to Sqlite. If the second argument is present, send a *ROLLBACK TRANSACTION to savepoint;* command to Sqlite where '*savepoint'* is the value passed as the second argument.

3.6 $zSqlite("SQL",*sql_command)*

The SQL *command* will be passed to the Sqlite3 server. The result, if a single value, will be returned.

3.7 $zSqlite("pragma",*option)*

A PRAGMA command will be sent to Sqlite with *option* as its argument. If the *PRAGMA* results in a returned value, it will be the returned result of the function. Otherwise, the function will return 0 (success) or 1 (failure).

Some example *PRAGMA* commands:

```
s i=$zsqlite("pragma","mmap_size=20000000")
s i=$zsqlite("pragma","cache_size=-1000000")
s i=$zsqlite("pragma","journal_mode=off")
```

3.8 $zsqlOpen

Returns *true* if a connection to the SQL server is open, *false* otherwise.

3.9 $zNative

$znative returns true (1) if globals are being stored in the native global array. False (0) otherwise.

4 Implementation Notes

4.1 Modulo Operator

The modulo operator (#) returns results that are the same as the C/C++ modulo operator (%). Some Mumps documentation shows the Mumps modulo returning results that are different than what would be expected from C/C++.

4.2 Goto Command

If you use a **goto** command, all **do** command pending returns are canceled. That is if you invoke a section of code by means of a **do** and the section of code executes a **goto** command, the return to the line the **do** was on is canceled as well as any other pending returns.

You may not use a **goto** in a compiled program block.

4.3 Notes on Arithmetic Precision

See section 1.9 on page 13 for additional details.

4.3.1 $fnumber()

The builtin function **$fnumber()** only works on numbers that can be represented in a 64 bit floating point variable.

4.3.2 Exponential format numbers

All numbers represented in exponential format are treated as floating point numbers. If exponential format constants are used in expressions, they must be enclosed in quotes:

```
set i="1.23e3"*5
```

4.3.3 Arithmetic Precision

If found, Mumps will use the GNU *bignum* integer and MPFR floating point packages (this can be disabled by a *configure* option).

4.3.3.1 Floating Point Precision

When using extended precision MPFR numbers, floating point values have a default fractional precision of 72 bits. This can be changed with the *--with-float-bits=val* configure option. The maximum number of printed decimal digits is, by default, 20. This can be changed with the *--with-float-digits=val* configure option. The number of meaningful decimal digits that can be printed depends upon the number of bits in the fractional part of the floating point number. More bits mean more decimal digits can be printed.

If MPFR is not present, standard hardware *double* precision is used.

4.3.3.2 Integer Precision

There is no effective limit to integer precision except string length and memory when the extended precision *bignum* package is in use. Otherwise, precision is the same as the hardware *long.*

4.3.3.3 Performance

Extended precision arithmetic results in slower performance. The amount is dependent on how much arithmetic a program does, whether it is mainly integer or floating point (floating point is slower), and, in the case of fixed length numbers, how large the numbers are. Larger numbers result in slower computations.

4.3.4 Rounding

The *$justify()* function is useful to round lengthy repeating decimal floating point numbers to a more reasonable value.

4.4 New Command

The **new** command functions differently than in the 1995 standard. The following details its behavior.

4.4.1 Runtime Symbol Table

The **new** command controls the internal run time symbol table. Upon entering a block by means of a **do** command, a new layer of the symbol table is created. Upon exit, the layer is discarded and the previous layer becomes the current layer.

When a program begins, an initial or base layer is created in the symbol table. In the absence of any **new** commands, newly created variables are stored at this base or initial layer.

When a variable is retrieved, all layers are searched beginning with the most recently created layer and progressing through to older layers until the initial layer is reached.

In the absence of any **new** commands, only the initial or base layer will contain variables.

4.4.2 Forms of the New Command

There are three forms of the **new** command based on the arguments provided. The first has no arguments, the second has a list of arguments consisting of variable names separated from one another by commas, and, finally, the third has an argument consisting of a parenthesized comma separated list of variable names. For example:

```
new
new a,b,c
new (a,b,c)
```

4.4.2.1 New Command with No Arguments

A **new** command with no arguments cause the system to copy all variables from all layers to the current layer.

Until the current block is exited, all access to any variable known at the time of the **new** command will access the copy of the variable, not the original. Upon exit from the block, the copies are deleted[7].

Any variable created whose name was not known when the **new** command was executed, will be created and stored at the lowest base layer of the symbol table and, consequently, not deleted upon exit from the block that contained the **new** command.

If a **new** command is executed in a block that invokes a block which itself executes a **new** command, the **new** command in the second block makes of copy of the invoking block's variables along with any variables created by the invoking block after executing its **new** command. If, in the symbol table stack, a vriable appears at several layers, only the most recent version will be copied.

An example is given in Figure 1. In this example, variables *i, j,* and *k* are created at the beginning of the program. The function *test1* is then called.

Initially, in *test1*, the variables have the same values that they did in the main function. The variable *i* is changed. The **new** command is executed and a copy of all the variable currently known (*i,j,k*) is made to the current layer. The values of *i*, *j*, and *k* are altered the function *test2* is called.

The values of the variables on entry to *test2* are the same as they were in *test1*. Anothe **new** command is executed making another copy of the variables. These are altered and a new variable, *y*, not previously known at any level (and thus stored at the base level) is created. Return is made to *test1.*

In *test1* the values of the variable are printed and it can be seen that they have reverted to the values they had prior to entering *test2*. Return is made to the main function.

7 A block is any sequence of code entered as a result of a **do** command.

In the main function the variables have reverted to the values they had prior to the invocation of *test1* with the exception of *i* which was altered in *test1* prior to execution of the **new** command. It retains the value it received in *test1*.

Note also that the variable *y* now exists at the main function level since, when it was created in *test1*, it was not in the group of variables copied to the symbol table level for *test1*. Thus, it was created at the base level of the symbol table.

However, when *y* was altered in *test2*, only the copy made by the **new** command in *test2* was altered, not the original.

```
#!/usr/bin/mumps
        set i=10
        set j=20
        set k=30
        do test1
        write "Main: expect 100 20 30 50: ",i," ",j," ",k," ",y,!
        halt

test1 write "test1: expect 10 20 30: ",i," ",j," ",k,!
      set i=100
      new
      set i=11,j=22,k=33,y=50
      do test2
      write "test1: expect 11 22 33 50 : ",i," ",j," ",k," ",y,!
      quit

test2 write "test2: expect 11 22 33 50: ",i," ",j," ",k," ",y,!
      new
      set i=12,j=23,k=34,y=55
      write "test2: expect 12 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

root@AMD6 validate new01.mps

test1: expect 10 20 30: 10 20 30
test2: expect 11 22 33 50: 11 22 33 50
test2: expect 12 23 34 55 : 12 23 34 55
test1: expect 11 22 33 50 : 11 22 33 50
Main: expect 100 20 30 50: 100 20 30 50
```

Figure 1 **new** Command without Arguments

4.4.2.2 New Command with Arguments

There are two forms of the **new** command that take arguments.

The first has a list of arguments consisting of variable names separated from one another by commas:

```
new a,b,c
```

The second has an argument consisting of a parenthesized, comma separated list of variable names:

```
new (a,b,c)
```

If a variable is named in the list that does not exist, it is created in the current symbol table layer with a value of the empty string.

4.4.2.2.1 New Command with Comma List of Variable Names

If the **new** command argument is a list of one or more variable names, it means that the variables listed will be copied to the current symbol table level and, eventually, discarded when the current block is exited[8].

If a variable whose name appears in the list exists at several layers in the symbol table stack, only the most recent will be copied.

Any reference to any variable not in the argument list will be satisfied by searching through the symbol table stack for the most recent instance of it. See Figure 2.

If a variable is mentioned in the argument list that does not exist, it is ignored.

```
#!/usr/bin/mumps
        set i=10
        set j=20
        set k=30
        do test1
        write "Main: expect 100 20 30 50: ",i," ",j," ",k," ",y,!
        halt

test1 write "test1: expect 10 20 30: ",i," ",j," ",k,!
      set i=100
      new i,j
      set i=11,j=22,k=33,y=50
      do test2
      write "test1: expect 11 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

test2 write "test2: expect 11 22 33 50: ",i," ",j," ",k," ",y,!
      new i
      set i=12,j=23,k=34,y=55
      write "test2: expect 12 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

root@AMD6 validate # new02.mps

test1: expect 10 20 30: 10 20 30
test2: expect 11 22 33 50: 11 22 33 50
test2: expect 12 23 34 55 : 12 23 34 55
test1: expect 11 23 34 55 : 11 23 34 55
Main: expect 100 20 30 50: 100 20 34 55
```

Figure 2 **new** Command with Comma List

4.4.2.2.2 New Command with Parenthesized List of Variable Names

If the **new** command argument list consists of a parenthesized list of one or more variable names, it means to make a copy of the most recent versions of all known variables except for the variable named in the list. This is similar to the no-argument version except the one or more variables known at the time of command execution will not be copied to the current symbol table layer.

When the block containing the **new** command is exited, the copies of the variables are discarded but any changes to this variables given in the argument list are not[9].

See Figure 3.

```
#!/usr/bin/mumps
        set i=10
        set j=20
```

8 A block is any sequence of code entered as a result of a **do** command.

9 Note: if one or more of the variables in the argument list are themselves copies from a lower layer but not the base layer, they will eventually be discarded.

```
        set k=30
        do test1
        write "Main: expect 11 22 30 50: ",i," ",j," ",k," ",y,!
        halt

test1 write "test1: expect 10 20 30: ",i," ",j," ",k,!
      new (i,j)
        set i=11,j=22,k=33,y=50
      do test2
      write "test1: expect 11 23 34 55 : ",i," ",j," ",k," ",y,!
        quit

test2 write "test2: expect 11 22 33 50: ",i," ",j," ",k," ",y,!
      new i
      set i=12,j=23,k=34,y=55
      write "test2: expect 12 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

root@AMD6 validate # new03.mps

test1: expect 10 20 30: 10 20 30
test2: expect 11 22 33 50: 11 22 33 50
test2: expect 12 23 34 55 : 12 23 34 55
test1: expect 11 23 34 55 : 11
```

Figure 3 **new** Command with Parenthesized List

4.5 Kill Command

The **kill** command operates only on the current symbol table level.

4.6 For Command Extensions

The **for** command accepts extensions such as the following:

```
for i=$order(^a(i)) ...
for i=init:$order(^a(i)):final ...
```

In the first example, the variable *i* will assume all the index values of the global array in collating sequence order.

In the second, the first value of *i* will be the next higher collating sequence value of the index above *init* and subsequent values will be the values in collating sequence order of the global array up to but not including *final*.

4.7 Break and Quit

In this version, the **break** command has a non-standard use. Originally intended as a means of interrupting a program for debugging purposes, in this implementation is is used in loop control.

A **quit** in a single line **for** terminates processing of the **for**. If there are multiple **for** commands, it terminates the nearest:

```
for i=1:1:10 write i,! if i>5 quit
     writes 1 through 6 only.


for i=1:1:10 for j=1:1:10 write j,! if j>5 quit
     writes 1 through 6 ten times.
```

A **break** may *NOT* be used in a single line **for** command. It may *ONLY* be used in an indented block that was introduced by a **do** command.

In an indented block, **quit** and **break** have special meanings:

A **quit** ends further processing of the block in which it appears and returns control to the line containing the invoking **do** at a point just after the **do**. Processing of the line containing the invoking **do** resumes. If there are more commands on the line, they are executed.

A **break** ends further processing of the block in which it appears but does not return the line containing the invoking **do**. Instead, execution moves to the line following the block which the **do** invoked.

Examples:

```
for i=1:1:10 do  write " continuing"
. write !,i
. if i>5 quit
. write " ",i
write !,"done",!

writes

      1 1 continuing
      2 2 continuing
      3 3 continuing
      4 4 continuing
      5 5 continuing
      6 continuing
      7 continuing
      8 continuing
      9 continuing
      10 continuing
      done
```

In this example, the block is invoked 10 times. After each invocation, the remainder of the line containing the **for** is executed producing the instances of the word "continuing". Each block invocation prints the value of "i". When the value of "i" is greater than 5, the block executes the **quit** command thus returning to the invoking line early. When the value if "i" is 5 or less, the full block is executed and return is made to the invoking line at block end. When the **for** command finishes execution, control is passed to the line following the **for** and "done" is printed.

```
set i=9
if i>0 do  write " continuing"
. write !,i
. if i>5 quit
. write " ",i
write !,"done",!

writes:

      9 continuing
      done
```

In this example, the block is entered, the value of "i" is printed but, because "i" is greater than 5, the **quit** is executed and control is returned to the invoking **do** and the word "continuing" is printed. Now, the line being completely executed, control passes to the line following the block and "done" is printed.

```
for i=1:1:10 do  write " mark " do  write " end of line",!
. write i
. if i>5 quit
. write "X"

writes:

      1X mark 1X end of line
      2X mark 2X end of line
      3X mark 3X end of line
```

```
4X mark 4X end of line
5X mark 5X end of line
6 mark 6 end of line
7 mark 7 end of line
8 mark 8 end of line
9 mark 9 end of line
10 mark 10 end of line
```

In this example, multiple **do** commands are shown. Note the two blanks following each. Each **do** invokes the block following the line containing the **do**

On the other hand, the **break** command terminates the the block in which it is contained but execution does not return to the line containing the invoking **do** but, instead, continues with the line following the block:

```
for i=1:1:10 do  write " continuing"

. write !,i
. if i>5 break
. write " ",i
write !,"done",!

writes:

      1 1 continuing
      2 2 continuing
      3 3 continuing
      4 4 continuing
      5 5 continuing
      6
      done

set i=9
if i>0 do  write " continuing"
. write !,i
. if i>5 break
. write " ",i
write !,"done",!

writes:
9
done

for i=1:1:10 do  write " mark " do  write " end of line",!
. write i
. if i>5 break
. write "X"
write !

writes:

      1X mark 1X end of line
      2X mark 2X end of line
      3X mark 3X end of line
      4X mark 4X end of line
      5X mark 5X end of line
      6
```

In these examples, execution of the **break** can be seen to terminate the current block and move to the line following the block.

```
for i=1:1:10 do
. for j=1:1:5 do
.. write j,!
```

```
.. if j>3 break
```

The above write 1 through 4 ten times.

Note: the contents of **$test** revert to their former value when exiting an indented block by means of **break** or **quit**:

```
if 1=1 do
. write "test 1: ",$test,!
. if 1=2 write "wow",!
. else  write "not wow",!
. write "test 2: ",$test,!
write "test 3: ",$test,!

writes:

      test 1: 1
      not wow
      test 2: 0
      test 3: 1
```

If you exit a block with a **goto**, the value of **$test** is not restored.

4.8 Lock Command with SQL

Locks are not needed if you are using Sqlite3 for global array storage as SQL transaction commands can achieve the same or better effect.

When using SQL for the backend global array stores, the Lock should not be used. Instead, use the more modern native SQL transaction processing commands (*BEGIN, COMMIT, ROLLBACK*, etc.) to achieve the same effect with far greater integrity (see Section 3 on page 21).

4.9 Lock Command in Shared Native Database Mode

In native B-tree mode, the Lock command creates a file named *Mumps.Locks* in */tmp* where lock information for the system is stored. If this file becomes corrupted due to abnormal terminations, it should be deleted. It will be rebuilt as needed.

4.10 Naked indicator

This version of Mumps does not support the naked indicator.

It was originally included in early versions of Mumps because of the inefficient binary mapping of an n-way tree which was used at the time to store the global arrays. The naked indicator was a short-hand to the interpreter to allow it to search for a global without stating at the top of the tree each time thus resulting in faster access. That is no longer the case with B-tree based access methods.

The main issue is the ambiguity of determining what exactly the naked indicator is after certain Mumps operations. Unfortunately, some legacy applications use it. These should be re-written.

4.11 Job command

The **JOB** command results in a C/C++ *fork()* function to be executed thus creating a child process. The child process will attempt to execute the argument to the **JOB** command. The **JOB** command may be used in the native B-tree user mode but only one process may access the globals.

The child process must end with a **HALT** command or the child process will hang.

4.12 File Names Containing Directory Information

When invoking a file name containing directory information (forward slash in Linux) with the **DO** or **GOTO** commands, the file name itself **must** be enclosed in quotes. For example:

```
set x="""^/home/user/xxx.mps""" goto @y
goto @"""^/home/user/xxx.mps"""
```

Note the extra quotes. These are required.

4.13 File Names

File names should conform to variable naming conventions except that the first character of a file name may not be the percent sign (%) character. The first character must be alphabetic. File names may only contain letters, digits and the percent sign.

4.14 Array Index Collating Sequence

Array index collating sequences for both global and local array is ASCII. That is, for the *$query()* and *$order()* functions, all array indices will be presented in the same order as ASCII strings. Thus, in an array with 15 elements whose indices range from 1 to 15, the indices will be presented as:

```
1 10 11 12 13 14 15 2 3 4 5 6 7 8 9
```

Other versions of Mumps may present numeric indices in numeric order. This, however, leads to considerable inefficiencies in the data base.

You may achieve numeric ordering by storing the indices padded to left with blanks such as:

```
for i=1:1:15 set ^a($justify(i,8))=i
set i=”” for  set i=$order(^a(i)) quit:i='' write +i,” “
```

the indices will now be presented as:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Note the the *+i* in the **write** command has the effect of converting the string to a number with no leading blanks.

4.15 Subroutine & Function Calls

Subroutines and functions may be performed in several ways as shown in Figure 4. Values returned from functions invoked by a **do** command are ignored. In standard Mumps, the *$$* form is used only with function invocations.

Caution: be certain to include a **halt** or other exit in your program *prior* to any functions that may appear at the end of your code. If the **halt** is not present, function code will be entered and any passed variables will be undefined.

```
#!/usr/bin/mumps
# calls.mps

      set i=10
      do fcn(i)
      do fcn(5)
      do $$fcn(i)
      do $$fcn(5)
      set k=$$fcn(5)
      write "returned k=",k,!

      set i=10
      do fcn^ext.mps(i)
      do fcn^ext.mps(5)
      do $$fcn^ext.mps(i)
      do $$fcn^ext.mps(5)
      set k=$$fcn^ext.mps(5)
      write "returned k=",k,!

      do fcn^ext1.mps
      do fcn^ext1.mps
```

```
      do $$fcn^ext1.mps
      do $$fcn^ext1.mps
      set k=$$fcn^ext1.mps
      write "returned k=",k,!

      halt

fcn(x) write "in fcn(x) value passed is ",x,!
       quit x

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

#!/usr/bin/mumps
# ext.mps

fcn(x) write "in fcn(x) value passed is ",x,!
       quit x

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

#!/usr/bin/mumps
# ext1.mps

fcn    write "in fcn ext1.mps",!
       set x=22
       quit x
------------------------------

output results:

in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
returned k=22
```

Figure 4 Subroutine/Function Calls

4.16 $Fnumber() Function

The **$fnumber()** is implemented via the C function *strfmon()* which provides much greater flexibility when dealing with differing locales and, especially, currencies. The default locale is *en_US.UTF-8* but this can be set with the *configure* option:

`--with-locale=`*`location-information`*

You may use **$fnumber()** with the legacy Mumps parameters or use it with a pattern parameter designed for *strfmon()*.

If you use the *strfmon()* parameter option, the function takes two arguments. The first must be a number consisting of only numeric characters. The second is a character string conforming to a *strfmon()* pattern but preceded by an asterisk to distinguish the pattern from those used by the legacy Mumps function of the same name. The *strfmon()* function is well documented but here are some examples:

```
set x=12345.6789
write $fn(x,”*%!n”)     ==> 12,345.68
write $fn(x,”*%n”)      ==> $12,345.68
write $fn(x,”*%i”)      ==> USD 12,345.68
write $fn(x,”*%n3”)     ==> $12,345.683
write $fn(x,"*%20n")    ==>           $12,345.68
```

4.17 $Select() Function

All arguments of the **$select()** function are evaluated. In standard Mumps, they are evaluated individualy until one is true or all are false.

4.18 Compiling Large Programs

When compiling[10] large programs, especially if SQL is enabled, there may be a warning about *variable tracking* from the gcc/g++ compiler. You may ignore this.

4.19 Embedded Expressions

In several extended Mumps commands, the figure *&~exp~* may appear. The expression *exp* is evaluated and the result replaces the figure. For example:

```
set x=”ls -lh”
shell &~x~
```

4.20 Inline C++ Code (Compiler Only)

Lines that begin with a plus (+) sign in column 1 are inserted as-is into C++ programs. Usually, these will be lines of C++ code. For example, if you have a line of Mumps code you want to execute 1000 times, the Mumps code would be:

```
for i=1:1:1000 do
. write “abc”,!
```

This could be written as:

```
+     for(int i=0; i < 1000; i++) {
      write “abc”,!
+     }
```

The C++ *for* loop is considerably faster.

Mumps code will not have access to the C++ variable unless you use a *declared* variable into which you place the C++ variables value:

```
      declare val
+     for(int i=0; i < 1000; i++) {
+     sprintf(val, “%s”, i);
      write val,!
+     }
```

4.21 Functions

This is the form of subroutine was originally used in Mumps. There are no parameters passed to the subroutine and the subroutine shares the same *namespace* as the calling program hence, as seen

10 Using the compiler is not presently recommended.

in the example in Figure 5, the values of the variables *i*, *j*, and *k* are accessible to the subroutine and any changes to them are available in the calling program.

Variables created in the subroutine in the normal manner by a **set** or **read** command, unless the subject of a **kill** command, are available to the calling routine.

Variables created in the subroutine as a result of a **new** command are destroyed upon return and are not available to the calling routine.

```
        zmain
        set i=10
        set j=20
        set k=30
        write "main program: ",i," ",j," ",k,!
        do test
        write "main program: ",i," ",j," ",k,!
        write "main program x=",x,!
        write "main program $data(y)=",$data(y),!
        halt

test
        write "sub-program: ",i," ",j," ",k,!
        set i=11
        set j=22
        set k=33
        set x=22
        new y
        set y=33
        quit

which produces the following output:

main program: 10 20 30
sub-program: 10 20 30
main program: 11 22 33
main program x=22
main program $data(y)=0
```

Figure 5 Inline Functions

4.21.1 Call by Value

This form of subroutine call was introduced later in the evolution of Mumps. It permits parameters to be passed to the subroutine but the subroutine maintains a separate name space for values passed to it as parameters. Variables from the calling program are visible to the called program. Variables created by the called program become available to the calling program upon return (except if the are **kill**ed prior to return or created by a **new** command). and variables created in the called program are deallocated upon return and are thus not visible to the calling program. Changes to parameters passed to the called program do not change the corresponding arguments in the calling program.

```
        zmain
        set i=10
        set j=20
        set k=30
        write "main program: ",i," ",j," ",k,!
        do test(i,j,k)
        write "main program: ",i," ",j," ",k,!
        halt

test(a,b,c)
        write "sub-program: ",a," ",b," ",c,!
        set a=11
        set b=22
```

```
        set c=33
        quit

which produces the following output:

main program: 10 20 30
sub-program: 10 20 30
main program: 10 20 30
```

Figure 6 Call by Value Functions

4.21.2 Call by Reference.

Same as the above but 'call be reference' permitted. That is, changes to parameters made by the called program cause changes to the corresponding arguments in the calling program. Note the "." in front of the variables in the 'do' command that are to be passed by reference. Both call by reference and call by value arguments may be mixed in the same 'do' statement.

```
#!/usr/bin/mumps
        zmain
        set i=10
        set j=20
        set k=30
        write "main program: ",i," ",j," ",k,!
        do test(.i,.j,.k)
        write "main program: ",i," ",j," ",k,!
        halt

test(a,b,c)
        write "sub-program: ",a," ",b," ",c,!
        set a=11
        set b=22
        set c=33
        quit

which produces the following output:

main program: 10 20 30
sub-program: 10 20 30
main program: 11 22 33
```

Figure 7 Call by Reference Functions

In each of the examples, the subroutine and calling program are actually part of the same C++ function. In effect, subroutines of the type shown above as similar to the old Basic ***gosub*** facility. Functions such as shown above may also return values:

An example recursive factorial computation is shown in Figure 8.

```
#!/usr/bin/mumps
      zmain
      set i=$$factorial(5)
      write "factorial=",i,!
      halt

factorial(a)
      write "sub-program: a=",a,!
      if a<2 quit 1
      set b=$$factorial(a-1)
      write "a=",a," b=",b,!
      quit a*b
```

```
sub-program: a=5
sub-program: a=4
sub-program: a=3
sub-program: a=2
sub-program: a=1
a=2 b=1
a=3 b=2
a=4 b=6
a=5 b=24
factorial=120
```

Figure 8 Function Return Values

5 Shell Commands

The **shell** command passes the remainder of the line to a shell for execution (**sh** in Linux). Shell output will appear on **stdout**. The command sets **$test** to false if the *fork()* fails, true otherwise.

5.1 shell/p

The **shell/p** form passes the remainder of the line to a shell for execution but opens a pipe **from** the shell **to** Mumps unit number 6. All **stdout** output from the shell is directed to unit number 6 and can be read with any of the input commands or functions in association with the **use** command.

5.2 shell/g

The **shell/g** form passes the remainder of the line to a shell for execution (**sh** in Linux) and opens an output pipe **from** the Mumps program **to** the shell as Mumps unit number 6. Data **written** to this unit becomes **stdin** to the shell. Output from the shell is written to **stdout**. Remember to **close** unit number 6 to signal end-of-file to the shell.

5.3 shell

With no qualifier, the **shell** command passes the remainder of the command line to a shell. Input or output from the shell come from or go to **stdin** or **stdout**, respectively.

5.4 Epression Substitution

In all cases, the remainder of the command line is scanned for **&~...~** expressions. The expression between **&~** and **~** is evaluated and the result replaces the **&~...~** expression.

For example:

```
shell sort dictionary.tmp | uniq -c | sort -nr > dictionary.s
```

The Linux shell created will do the following:

1. The file *dictionary.tmp,* a collection of words, will be sorted by **sort** and the output piped to **uniq**
2. **uniq** counts duplicate entries and pipes its output consisting of a count and a word to **sort**
3. **sort** sorts the result numerically by number of duplicates in reverse order and writes its output to *dictionary.s*.

```
shell/p sort dictionary.tmp | uniq -c | sort -nr
open 1:"dictionary.s,new"
for  do
. use 6
. read line
. if '$test break
. use 1
. write line,!
close 1
```

Figure 9 Shell Command Example

The above does the same but the output will be presented to Mumps unit 6 which reads and writes the result to the file named dictionary.s

6 Added Commands

6.1 Database *expr*

By default, Native databse file *key.dat* and *data.dat* are stored in the directory current when a program is invoked.

The **database** command may be used to set the name of the files to be used to store the native global arrays. The expression will be evaluated and the resulting name will become the name, suffixed *.key* and *.dat,* of the files in which the native global arrays are stored. The expression may contain directory information. For example:

```
database "/home/user/data/mumps"
```

will cause the system to access files:

```
/home/user/data/mumps.key
/home/user/data/mumps.dat
```

This command ***must*** be issued prior to any attempt to access the global arrays. It only works with the native B-tree database option.

6.2 Zhalt return_code

The **zhalt** command will terminate the current program with a return error code given by its argument. Example:

```
if a=0 zhalt 99
```

The value of **$?** in the BASH environment will be 99.

6.3 Declare

The Declare command is ignored by the interpreter.

In the compiler, is can be used to establish one or more variables as declared C++ variables rather than variables stored in the Mumps run time symbol table. Consequently, access to these variables is about twice as fast.

Declared variables may only be scaler variables. The may not, at present, be subscripted.

Declared variable names may conflict with exiting internal compiler variables. In which case, select a different name for your scaler vairable.

Example:

```
zmain
declare dclx,dclx1
for dclx=0:1:1000 set dclx1=dclx write dclx1,!
```

7 Z Functions and System Variables

$zfunctions are extensions added by the implementor and not covered by the standard. Thus, many if not all of the following M2 extensions may not be supported or supported differently in other implementations. Likewise, there are implementer defined system variables which may be queried and, in some cases, set.

M2 implementation note: you may add new **$z** functions by modifying the function **zfcn()** located in the source file *bifs.cpp.in*

7.1 System Variables

7.1.1 $zProgram

Returns a string with the name of the currently executing program.

7.2 Bash Functions

7.2.1 $zbasename(arg1[,arg2])

Returns a result equivalent of the Bash function *basename*

```
$zbasename("/home/jsmith/base.wiki") yields base.wiki
$zbasename("/home/jsmith/") yields jsmith
$zbasename("/") yields /

$zbasename("/home/jsmith/base.wiki",".wiki") yields base
$zbasename("/home/jsmith/base.wikia","ki") yields base.wi
$zbasename("/home/jsmith/base.wiki","base.wiki") yields base.wiki
```

7.2.2 $zfiletest(arg1,arg2)

Performs a Bash style check on a file name. The first argument is the name of a file and the second is a parameter that determines the type for file check. If the check condition is *true*, a one (1) is returned, zero (0) otherwise. The following are legal values for the second argument:

```
-a True if FILE exists.
-b True if FILE exists and is a block-special file.
-c True if FILE exists and is a character-special file.
-d True if FILE exists and is a directory.
-e True if FILE exists.
-f True if FILE exists and is a regular file.
-g True if FILE exists and its SGID bit is set.
-h True if FILE exists and is a symbolic link.
-k True if FILE exists and its sticky bit is set.
-p True if FILE exists and is a named pipe (FIFO).
-r True if FILE exists and is readable.
-s True if FILE exists and has a size greater than zero.
-t True if file descriptor FD is open and refers to a terminal.
-u True if FILE exists and its SUID (set user ID) bit is set.
-w True if FILE exists and is writable.
-x True if FILE exists and is executable.
-O True if FILE exists and is owned by the effective user ID.
-G True if FILE exists and is owned by the effective group ID.
-L True if FILE exists and is a symbolic link.
-N True if FILE exists and has been modified since it was last read.
-S True if FILE exists and is a socket.
```

7.3 Math Functions

The following C/C++ math functions are available in M2. Their arguments and return values are the same as the correspondingly named C++ functions.

7.3.1 **$zabs(arg)** absolute value

Function returns the absolute value of its numeric argument.

7.3.2 **$zacos(arg)** arc cosine

Computes the inverse cosine (arc cosine) of the input value. Arguments must be in the range -1 to 1.

7.3.3 **$zasin(arg)** Arc sine

Computes the inverse sine (arc sine) of the argument **arg**. Arguments must be in the range -1 to 1.

7.3.4 **$atan(arg)** Arc tangent

Computes the inverse tangent (arc tangent) of the input value.

7.3.5 **$zcos(arg)** Cosine

Computes the cosine of the argument **arg**. Angles are specified in radians.

7.3.6 **$zexp(arg)** Exponential

Calculates the exponential of **arg**, that is, ***e*** raised to the power *arg* (where ***e*** is the base of the natural system of logarithms, approximately 2.71828).

7.3.7 **$zexp2(arg)** Exponential base 2

Calculates 2 raised to the power *arg*.

7.3.8 **$zexp10(arg)** Exponential base 10

Calculates 10 raised to the power *arg*.

7.3.9 **$zlog(arg)** Natural log

Returns the natural logarithm of **arg**, that is, its logarithm base ***e*** (where e is the base of the natural system of logarithms, 2.71828...).

7.3.10 **$zlog2(arg)** Base 2 log

Returns the base 2 logarithm of ***arg***.

7.3.11 **$zlog10(arg)** Base 10 log

Returns the base 10 logarithm of **arg**.

7.3.12 **$zpow(arg1,arg2)** Power function

Calculates **arg1** raised to the exponent **arg2**.

7.3.13 **$zsqrt(*arg*)** Square root

Function returns the square root of its numeric argument.

7.3.14 **$zsin(arg)** Sine function

Computes the sine of the argument **arg**. Angles are specified in radians.

7.3.15 **$ztan(arg)** Tangent function

Computes the tangent of **arg**.

7.4 Date functions

7.4.1 **$zdate(or $zd)** formatted date string

Function returns the system date and time in standard system printable format. This includes: day of week, month, day of month, time (hour:minute:second), and year (4 digits).

7.4.2 **$zd1** numeric internal date

Returns the number of seconds since January 1, 1970 - a standard used in Linux. This number may be used to accurately correlate events.

7.4.3 **$zd2(*InternalDate*)** date conversion

Translates the Linux time from $ZD1 into standard system printable format. The argument is a Linux format time value.

7.4.4 **$zd3(*Year,Month,Day*)** Julian date

Returns the day of the year (Julian date) for the Gregorian date argument.

7.4.5 **$zd4(*Year,DayOfYear*)** Julian to Gregorian

Returns the Gregorian date for the Julian date argument.

7.4.6 **$zd5(*Year, Month, Day*)** comma listed date

Returns a string consisting of the year, a comma, the day of year, and the number of days since Sunday (Monday is 1).

7.4.7 **$zd6** hour:minute

Returns a string consisting of the hour, a colon, and the minute.

7.4.8 **$zd7** hyphenated date

Returns a string consisting of the year, hyphen, month, hyphen, and day of month. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

7.4.9 **$zd8** hyphenated date with time

Returns a string consisting of the year, hyphen, month, hyphen, and day of month, comma, and time in HH:MM format. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

7.5 Special Purpose Functions

The following special purpose functions are available:

7.5.1 **$zb(arg)** remove blanks

Function returns a string in which all leading blanks have been removed and all multiple blanks have been replaced by single blanks. See also **$zNoBlanks()**. Figure 10 gives examples.

```
1 #!/usr/bin/mumps
2  set a="   abc   xyz     123    "
3  write $zb(a),"***",!

output:

abc xyz 123 ***
```

Figure 10 $Zb() Examples

7.5.2 **$zchdir(directory_path)** change directory

Function changes the current directory to the path specified. If the operation succeeds, a zero is returned. If it fails, -1 is returned.

7.5.3 $zCurrentFile Current Mumps File

Returns the name of the currently executing Mumps program file (if any) ar blank.

7.5.4 $zdump[(filename)] dump global arrays

Function dumps the globals to a sequential ASCII file in the current directory. If an argument is given, it is taken as the name of the file to which the globals will be written. If the argument is omitted, a file name is constructed from the system date of the form **number.dmp** where **number** is the value of the C++ **time()** function at the time of the dump.

The dump file is a pure ASCII text file. Each entry in the global array is represented by two lines. The first line is the global array reference and the second line is the store value. In the global array reference, parentheses and commas are replaced by the "~" character. Thus, if you wish to use this facility, you may not include the "~" character in a global array index.

The function **$zrestore()** reloads the global arrays from a dump file (see below).

$zdump and **$zrestore** do not work when SQL is used for the global array store.

7.5.5 $zrestore[(arg)] restore globals

Function restores the globals from a dump file produced by **$zdump**. If an argument is given, it is taken as the name of the dump file otherwise, the default name **dump** is used.

$zdump and **$zrestore** do not work when SQL is used for the global array store.

7.5.6 $zfile(*arg*) file exists test

Function returns a zero or one indicating if the file given as the argument exists.

7.5.7 $zflush flush Btree buffers

Function flushes all modified native global array handler buffers to disk. The function should only be used with the native globals. After flushing, all updates to the btree file system have been committed. In cases where the internal buffers are very large, this function may take several seconds to execute. The function returns the empty string. Flushing the buffers is a precaution against system failure which would otherwise result in corruption of the global arrays.

7.5.8 $zgetenv(*arg*) get environment variable

Returns the contents of the environment variable specified as *arg* or the empty string if the variable is not found.

7.5.9 $zhtml(*arg*) encode HTML string

Function encodes its argument in the form necessary to be a cgi-bin parameter. That is, alphabetics remain unchanged, blanks become plus signs and all other characters become hexadecimal values, preceded by a percent sign.

7.5.10 $zhit global array cache hit ratio

Function calculates and returns the native global array cache hit ratio. This number ranges between zero and one. A value of one indicates all requests were satisfied from the cache while a value of zero indicates no requests were satisfied from the cache. Calling this function resets the hit ratio to zero. A higher value for the hit ratio indicates better database performance.

7.5.11 $zlower(string) convert to lower case

Function returns the input string with alphabetics converted to lower case.

7.5.12 $znormal(*arg1*[,arg2]) word normalization

Function converts the word passed as argument 1 to lower case and removes any embedded punctuation. If a second argument is given, the word is truncated to the length specified by this argument. If no second argument is given, words are truncated to 25 characters if their length exceeds 25 characters.

7.5.13 $zNoBlanks(arg*)* remove all blanks

Returns ***arg*** with all blanks removed. See also: **$zb**.

7.5.14 $zpad(arg1,arg2) left justify with padding

Function left justifies the first argument in a string whose length is given by the second argument, padding to the right with blanks.

7.5.15 $zseek(arg)

Function takes one argument (a positive integer) which is a byte offset in the currently active (use) file. The command moves the file pointer to that location in the file. **$zseek()** may only be used on files opened with **old** attribute. Figure 11 gives examples.

```
#!/usr/bin/mumps
 open 1:"tdb,new"
 for j=1:1:1000 do
 . use 1
 . set i=$ztell
 . set ^a(j)=i
 . write "**** ",j,!

 close 1
 open 1:"tdb,old"
 for j="":$order(^a(j)):"" do
 . use 1
 . set i=$zseek(^a(j))
 . read a
 . use 5
 . write a,!

output:

**** 1
**** 10
**** 100
**** 1000
**** 101
**** 102
**** 103
**** 104
**** 105
**** 106
**** 107
**** 108
**** 109
**** 11
**** 110
**** 111
...
```

Figure 11 $Zseek() Examples

7.5.16 $zsrand(arg)

Seed the random number generator. The value passed as the argument will seed the internal random number generator. If the random number generator is re-seeded with the same seed, the sequence of random numbers produced by **$random** will be the same. The value passed must be a positive integer.

7.5.17 $zstem(arg)

Returns an word English word stem of the argument. This function attempts to remove common endings from words and return a root stem.

7.5.18 $zsystem(arg)

Executes "arg" in a system shell. Returns -1 (fork failed) or the return code of the execution of the argument. See also the **shell** command.

7.5.19 $ztell

Function returns the byte offset in the currently open file. Similar to the C++ **ftello** function. Note: The offset returned is for the file most recently made the default i/o file by the **use** command. **$ztell** may be used on either a file opened as **new, old** or **append**. (See example under **$zseek** above)

7.5.20 $zu(expression)

Function returns 1 if the expression is numeric, 0 otherwise.

7.5.21 $zwi(arg)

Function loads an internal buffer with the string given as the argument. The alphabetic characters of the argument are converted to lower case. The contents of this buffer are returned by the **$zwn** and **$zwp** functions. Figure 12 gives examples.

7.5.22 $zwn extract words from buffer

Function returns successive words from the internal buffer delimited by blanks. When no more words remain, it returns an empty string (string of length zero). Returned words are converted to lower case. See **$zwi**.

7.5.23 $zwp extract words from buffer

Function returns successive words from an internal buffer delimited by blanks and punctuation characters. When no more words remain, it returns an empty string (string of length 0). Returned words are converted to lower case. See **$zwi**.

7.5.24 $zws(string) initialize internal buffer

Initializes the parse buffer but does not convert "string" to lower case as is the case with **$zwi**

```
#!/usr/bin/mumps
 set i="now, is the time, for all good"
 set %=$zwi(i)
 for w=$zwp write w,!
 write "-------",!
 set %=$zwi(i)
 for w=$zwn write w,!

output:

    now
    ,
    is
    the
    time
    ,
    for
    all
    good
    -------
    now,
    is
    the
    time,
    for
```

```
    all
    good
```

Figure 12 $Zwi() Examples

7.5.25 Scan Functions

7.5.25.1.1 $zzScan

7.5.25.1.2 $zzScanAlnum

7.5.25.1.3 $zzInput(var)

The functions return the next word in the current input stream delimited by white space. Words are restricted to a maximum length of 1023. Successive calls return successive words. When there are no more input words, an empty string is returned and **$test** is set to *false*.

If only part of a line is scanned as a result of these functions, a subsequent **read** command will begin at the white space following the last word returned.

If scanning input from stdin (i/o unit 5), you may signal end of file with a *control-d* on a separate line by itself. This will result terminate the scan and **$test** will be set to false.

$zzScan returns all words delimited by whitespace with no conversion. Words may contain any *printable* ASCII character.

$zzScanAlnum processes words before returning them according to the following rules:

- If a word begins with a number or punctuation, it is not returned.
- Non alpha-numeric characters are removed.
- Alpha characters are converted to lower case.

Both functions will advance to additional lines as needed. If a word exceeds 1023 bytes, the results are undefined. See Figure 13 for an example.

```
for the input line:

now -- __ ?? !@#$%^&*()_+= IS 2for the time for


      for  set i=$zzScan quit:'$test  write i,!

output:

      now
      --
      __
      ??
      !@#$%^&*()_+=
      IS
      2for
      the
      time
      for

      for  set i=$zzScanAlnum quit:'$test  write i,!

output:

      now
      the
      time
      for
```

Figure 13 Scan Functions Examples

$zzInput(var) reads an entire input line, converts all characters to lower case, separates the words, removes punctuation (as defined by the C *ispunct()* function except hyphen), and stores the words into a numerically indexed array whose name is the value of the variable or constant passed as the argument. The function returns the number of elements in the array. A return of zero indicates no input was obtained (end of file). As the array created by the function could be quite large, you should probably **kill** it when no it is longer needed. The maximum line length permitted is twice the system parameter *MAX_STR* (9,000 bytes by default).

7.6 Vector and Matrix Functions

7.6.1 $zzAvg(vector)

Computes and returns the average of the numeric values in the vector. For example, see Figure 14.

```
#!/usr/bin/mumps
 for i=1:1:10 set ^a(99,i)=i
 set i=$zzAvg(^a(99))
 write "average=",i,!
```

Figure 14 $zzAvg() Example

The above writes 5.5

7.6.2 $zzCentroid(gblMatrix,gblRef)

A centroid vector *gblRef* is calculated for the invoking two dimensional global array *gblMatrix*. The centroid vector is the average value for each for each column of the matrix. Any previous contents of the global array named to receive the centroid vector are lost. The global array *gblMatrix* must contain at least two dimensions. See Figure 15 for an example. The matrix must be a top level global array.

```
#!/usr/bin/mumps
 for i=0:1:10 do
 . for j=1:1:10 do
 .. set ^A(i,j)=5
 set %=$zzCentroid(^A,^B)
 for i=1:1:10 write ^B(i),!
```

output:

```
5
5
5
5
5
5
5
5
5
5
```

Figure 15 $zzCentroid() Example

7.6.3 $zzCount(gblVector)

Counts the number of nodes that contain a value in the global array reference and any descendants. For example, see Figure 9.

```
#!/usr/bin/mumps
 kill ^a
```

```
for i=1:1:10 set ^a(99,i)=i
set i=$zzCount(^a(99))
write "count=",i,!

writes: count=10
```

Figure 16 $zzCount() Example

7.6.4 $zzMax(gbl)

Computes and returns the maximum numeric value in the vector and any descendants. See Figure 17 for an example.

```
#!/usr/bin/mumps
for i=1:1:10 set ^a(99,i)=i
set i=$zzMax(^a(99))
write "max=",i,!

output:

10
```

Figure 17 $zzMax() Example

The above writes the largest value stored in the vector.

7.6.5 $zzMin(gbl)

Returns the minimum numeric value stored in the vector and any descendants. See Figure 18 for an example.

```
#/usr/bin/mumps
for i=1:1:10 set ^a(99,i)=i*2
set i=$zzMin(^a(99))
write "min=",i,!

output:

2
```

Figure 18 $zzMin() Example

7.6.6 $zzMultiply(gbl1,gbl2,gbl3)

Multiplies the first and second matrix leaving the result in the third. The ordinary rules of algebra apply. Figure 22 gives an example. The arguments *gbl1* and *gbl2* must be top level, two dimensional arrays.

7.6.7 $zzSum(gblVector)

Computes and returns the sum of the numeric values stored in the vector. For example, see Figure 23.

7.6.8 $zzTranspose(gblMatrix1,gblMatrix2)

Transposes the first global array matrix leaving the result in the second. For example, see Figure 24. the argument *gblMatrix1* must be a top level, two dimensional array.

7.7 Text Processing Functions

The following functions are used in connection with experiments in information storage and retrieval.

7.7.1 Similarity Functions

7.7.1.1 $zzCosine(gbl1,gbl2)

7.7.1.2 $zzSim1(gbl1,gbl2)

7.7.1.3 $zzDice(gbl1,gbl2)

7.7.1.4 $zzJaccard(gbl1,gbl2)

These compute the Cosine, Sim1, Dice and Jaccard similarity coefficients between document vectors given as the first and second arguments. Both arguments are numeric global array vectors. The formulae are given in Figure 19 and an example in code is given in Figure 20. The formulae calculate the similarities between two global array vector *gbl1* and global array vector *gbl2*. The vectors need not be of equal length. Missing elements are interpreted as zero. The vectors should be top level vectors.

$$Simularity_{Dice}(i,j)=\frac{2\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}}{\sum_{k=1}^{k=t} Term_{ik}+\sum Term_{jk}}$$

$$Simularity_{Jaccard}(i,j)=\frac{\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}}{\sum_{k=1}^{k=t} Term_{ik}+\sum Term_{jk}-\sum_{k=1}^{k=t}\left(Term_{ik}\cdot Term_{jk}\right)}$$

$$Simularity_{Cosine}(i,j)=\frac{\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}}{\sqrt{\sum_{k=1}^{k=t} Term_{ik}^{2}\cdot\sum Term_{jk}^{2}}}$$

$$Simularity_{¿1}(i,j)=\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}$$

Figure 19 Similarity Formulae

```
#!/usr/bin/mumps
 kill ^A
 kill ^B

 set ^A("1")=3
 set ^A("2")=2
 set ^A("3")=1
 set ^A("4")=0
 set ^A("5")=0
 set ^A("6")=0
 set ^A("7")=1
 set ^A("8")=1

 set ^B("1")=1
 set ^B("2")=1
 set ^B("3")=1
 set ^B("4")=0
```

```
set ^B("5")=0
set ^B("6")=1
set ^B("7")=0
set ^B("8")=0

write "Cosine=",$zzCosine(^A,^B),!
write "Sim1=",$zzSim1(^A,^B),!
write "Dice=",$zzDice(^A,^B),!
write "Jaccard=",$zzJaccard(^A,^B),!

output:

Cosine=0.75
Sim1=6
Dice=1
Jaccard=1
```

Figure 20 Similarity Functions

7.7.2 $zzBMGSearch(arg1,arg2)

Boyer-Moore-Gosper Function returns the number of non-overlapping occurrences of *arg1* in *arg2*.

These functions, were obtained from

`ftp://ftp.uu.net/usenet/comp.sources.unix/volume5/bmgsubs.Z`

and were written by Jeffrey Mogul (Stanford University), based on code written by James A. Woods (NASA Ames, an agency of the U.S. Government) and are thus believed to be in the public domain. Figure 21 gives an example.

```
#!/usr/bin/mumps
 set key=”now”
 set str=”now is the now of the now in the know”
 write $zBMGSearch(key,str),!

output:

4
```

Figure 21 $zzBMGSearch() Example

7.7.3 $zPerlMatch(string,pattern)

Applies the Perl **pattern** to **string** and returns 1 if the pattern fits and 0 otherwise. The **$zPerlMatch** function has the side effect of creating variables in the local symbol table to hold backreferences, the equivalent concept of **$1**, **$2**, **$3**, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming scheme as Perl (**$1** through **$9**). These variables remain defined up to a subsequent call to **$zPerlMatch**, at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6 through $9 will contain the empty string. Figure 25 contains examples (long lines wrapped).

```
#/usr/bin/mumps
 set ^d("1","1")=2
 set ^d("1","2")=3
 set ^d("2","1")=1
 set ^d("2","2")=-1
 set ^d("3","1")=0
 set ^d("3","2")=4

 set ^e("1","1")=5
```

```
 set ^e("1","2")=-2
 set ^e("1","3")=4
 set ^e("1","4")=7
 set ^e("2","1")=-6
 set ^e("2","2")=1
 set ^e("2","3")=-3
 set ^e("2","4")=0

 set %=$zzMultiply(^d,^e,^f)

 for i="":$order(^f(i)):"" do
 . for j="":$order(^f(i,j)):"" do
 .. write i," ",j," ",^f(i,j),!
```

output:

```
1 1 -8
1 2 -1
1 3 -1
1 4 14
2 1 11
2 2 -3
2 3 7
2 4 7
3 1 -24
3 2 4
3 3 -12
3 4 0
```

Figure 22 $zzMultiply() Example

```
#!/usr/bin/mumps
 for i=1:1:10 set ^a(99,i)=i
 set i=$zzSum(^a(99))
 write "sum=",i,!
```

output:

```
55
```

Figure 23 $zzSum() Example

```
#!/usr/bin/mumps
 kill ^f

 set ^d("1","1")=2
 set ^d("1","2")=3
 set ^d("2","1")=4
 set ^d("2","2")=0

 set %=$zzTranspose(^d,^f)

 for i="":$order(^f(i)):"" do
 . for j="":$order(^f(i,j)):"" do
 .. write i," ",j," ",^f(i,j),!
```

output:

```
    1 1 2
    1 2 4
    2 1 3
    2 2 0
```

Figure 24 $zzTranspose() Example

```
#!/usr/bin/mumps
 write "Please enter a telephone number:",!
  read phonenum

  set p="^(1-)?(\(?\d{3}\)?)?(-| )?\d{3}-?\d{4}$"
  if $zperlmatch(phonenum,p) do
  . write "+++ This looks like a phone number.",!
  . write "The area code is: ",$2,!
  else  do
  . write "--- This didn't look like a phone number.",!

output:

Please enter a telephone number:
(123) 456-7890
+++ This looks like a phone number.
The area code is: (123)

Please enter a telephone number:
(123) 456-7890
+++ This looks like a phone number.
```

Figure 25 $zPerlMatch() Example

7.7.4 $zReplace(string,pattern,replacement)

The regular expression in *pattern* is evaluated on *string* and, if there is a match, the matching section is replaced by *replacement*. Figure 26 contains an example. In the first part, the word 'is' is replaced by 'IS'. In the second part, a match is sought for any content between two sets of matching brackets ([[...]]). The matched section is in back reference **$2**. This is then used as a pattern to be replaced.

7.7.5 $zShred(string,length)

7.7.6 $zShredQuery(string,length)

The **$zShred()** function segments the input argument **string** into fragments of **length** size upon successive calls. The function returns a string of length zero when there are no more fragments of size **length** remaining (thus, short fragments at the end of a string are not returned).

$zShred copies the input string to an internal buffer upon the first call. Subsequent calls retrieve from this buffer. When the buffer is consumed, the function will copy the contents of the next string submitted to the buffer. Figure 27 contains an example.

```
#!/usr/bin/mumps
 set a="now is the time for all"
 set a=$zReplace(a,"is","IS")
 write a,!

 set a="[[now is the time]]"
 if $zPerlMatch(a,"(\[\[)(.*)(\]\])") do
 . set a=$zReplace(a,$2,"ABC")
 . write a,!

output:

now IS the time for all
[[ABC]]
```

Figure 26 $zReplace() Example

```
#!/usr/bin/mumps
 set a="now is the time for all good men to "
 set a=a_"come to the aid of the party"
 for  do  quit:j=""
 . set j=$zShred(a,5)
 . if j="" quit
 . write j,!

output:

    nowis
    theti
    mefor
    allgo
    odmen
    tocom
    etoth
    eaido
    fthep
```

Figure 27 $zShred() Example

The **$zShredQuery** function segments **length** shifted copies of the input **string** into fragments of size **length** upon successive calls. That is, the function first returns all the fragments of size length of the **string** in the same manner as **$zShred**. However, it then shifts the starting point of the input string to the right by one and returns all the fragments of size length relative to the shifted starting point. If repeatedly called, it repeats this process a total of **length** times. When there are no more combinations, the empty string is returned as shown in Figure 28.

```
#!/usr/bin/mumps
 set a="now is the time for all good men to come to "
 set a=a_"the aid of the party"
 for  do  quit:j=""
 . set j=$zShredQuery(a,5)
 . if j="" quit
 . write j,!

output:

    nowis                tothe                goodm
    theti                aidof                entoc
    mefor                thepa                ometo
    allgo                wisth                theai
    odmen                etime                dofth
    tocom                foral                epart
    etoth                lgood                isthe
    eaido                mento                timef
    fthep                comet                orall
    owist                othea                goodm
    hetim                idoft                entoc
    efora                hepar                ometo
    llgoo                isthe                theai
    dment                timef                dofth
    ocome                orall                epart
```

Figure 28 $ShredQuery() Example

7.7.7 $zzSoundex(s1)

Returns the Soundex code for the argument string as follows:

1. All letters are converted to lower case;
2. Non-alphabetic characters are removed;
3. Adjacent duplicate letters are replaced by a single occurrence;
4. The first letter is retained;
5. The letters b, f, p, and v are replaced by the number 1;
6. The letters c, g, j, k, q, s, x, and z are replaced by the number 2;
7. The letters d and t are replaced by the number 3;
8. The letter l is replaced by the number 4;
9. The letters m and n are replaced by the letter 5;
10. the letter r is replaced by the number 6;
11. The is truncated to four characters.

7.7.8 $zSmithWaterman(s1,s2,algn,mat,gap,noMatch,match)

Computes the Smith Waterman score between two strings. Result returned is the highest alignment score achieved. String lengths are limited by **STR_MAX** in the interpreter. If you compare very long strings (>100,000 characters), you may exceed stack space. This can be increased under Linux with the command:

```
ulimit -s unlimited
```

Figure 29 gives an example.

```
#!/usr/bin/mumps
 set s1="now is the time"
 set s2="now  i th  time"
 set i=$zSmithWaterman(s1,s2,1,0,-1,-1,2)
 write "score=",i,!

output:

   1 now- is the time 16
     ::: :: ::: :::::
   1 now  i- th  time 16

score=23
```

Figure 29 $zSmithWaterman() Example

Parameters:

If *algn* is zero, no printout of alignments is produced. If *algn* is not zero, a summary of the alternative alignments will be printed.

If *mat* is zero, intermediate matrices will not be printed.

The parameters *gap*, *noMatch* and *match* are the gap and mismatch penalties (negative integers) and the match reward (a positive integer).

If insufficient memory is available, a segmentation violation will be raised. Try increasing your stack size.

7.7.9 $zzIDF(global,doccount)

Calculates the Inverse Document Frequency score of words contained in the argument *global*. The parameter *doccount* is the total number of documents. The index of each element of the *global* vector is a word and the value stored is the number of times the word occurs in the collection. Figure 30 gives and example. The vector argument *global* must be a top level array.

```
#!/usr/bin/mumps
 set ^a("now")=2
 set ^a("is")=5
 set ^a("the")=6
```

```
 set ^a("time")=3
 set j=4
 set %=$zzIDF(^a,j)
 for i="":$order(^a(i)):"" write i," ",^a(i),!

output:

is 0.7
now 2.0
the 0.4
time 1.4
```

Figure 30 $zzIDF() Example

7.7.10 Correlation Functions

7.7.10.1 $zzTermCorrelate(global1,global2)

Calculates the Term-Term co-occurrence matrix for the Document-Term matrix in *global1*. The result is placed in *global2*.

A Term-Term matrix has terms (words) as the indices of its rows and columns. A Term-Term matrix gives, for each position, the degree to which the term corresponding to the row is similar to the term corresponding to the column. The diagonal, which is the degree a term is related to itself, is ignored. Both operands must be top level arrays.

In both the doc-doc and term-term matrices, the upper and lower diagonal matrices are mirror images of one another. Figure 31 gives an example. The order of words in the output will depend upon which data base facility is in use and what it's collating settings are. The Native global array handler collates according to ASCII-7.

```
#!/usr/bin/mumps
 kill ^A,^B

 set ^A("1","computer")=5
 set ^A("1","data")=2
 set ^A("1","program")=6
 set ^A("1","disk")=3
 set ^A("1","laptop")=7
 set ^A("1","monitor")=1

 set ^A("2","computer")=5
 set ^A("2","printer")=2
 set ^A("2","program")=6
 set ^A("2","memory")=3
 set ^A("2","laptop")=7
 set ^A("2","language")=1

 set ^A("3","computer")=5
 set ^A("3","printer")=2
 set ^A("3","disk")=6
 set ^A("3","memory")=3
 set ^A("3","laptop")=7
 set ^A("3","USB")=1

 set %=$zzTermCorrelate(^A,^B)

 for i="":$order(^B(i)):"" do
 . write i,!
 . for j="":$order(^B(i,j)):"" do
 .. write ?10,j," ",^B(i,j),!
```

output:

```
USB
        computer 1
        disk 1
        laptop 1
        memory 1
        printer 1
computer
        USB 1
        data 1
        disk 2
        language 1
        laptop 3
        memory 2
        monitor 1
        printer 2
        program 2
data
        computer 1
        disk 1
        laptop 1
        monitor 1
        program 1
disk
        USB 1
        computer 2
        data 1
        laptop 2
        memory 1
        monitor 1
        printer 1
        program 1
language
        computer 1
        laptop 1
        memory 1
        printer 1
        program 1
laptop
        USB 1
        computer 3
        data 1
        disk 2
        language 1
        memory 2
        monitor 1
        printer 2
        program 2
memory
        USB 1
        computer 2
        disk 1
        language 1
        laptop 2
        printer 2
        program 1
monitor
        computer 1
        data 1
        disk 1
        laptop 1
        program 1
printer
        USB 1
        computer 2
        disk 1
        language 1
        laptop 2
        memory 2
        program 1
program
        computer 2
        data 1
        disk 1
        language 1
        laptop 2
        memory 1
        monitor 1
        printer 1
```

Figure 31 $zTermCorrelate() Example

7.7.10.2 $zzDocCorrelate(gblref1,gblref2,mthd,thrshld)

A square Document-Document matrix *gblref2* is calculated from the Document-Term matrix *gblref1* according to method *mthd* (Cosine, Sim1, Dice, Jaccard). The value of elements in the Document-Document matrix will not exceed threshold (*thrshld*) and the cells associated with coresponding document numbers will not exist.

A Document-Document matrix has document id's as its row and column indices. A cell in the matrix indicates the degree to which the row document is related to the column document. The diagonal is ignored. Figure 32 gives an example.

7.7.11 Stop and Synonym Functions

7.7.11.1 $zStopInit(arg)

7.7.11.2 $zStopLookup(word)

7.7.11.3 $zSynInit(fileName)

7.7.11.4 $zSynLookup(word)

A call to **$zStopInit(file_name)** will open and load a file of stop words into a C++ container. The file should consist of one word per line. If the file cannot be opened or there is insufficient memory to hold the list of words, the program will halt with an error message. **$zStopInit()** converts all words to lower case.

```
#!/usr/bin/mumps
 kill ^A,^B

 set ^A("1","computer")=5
 set ^A("1","data")=2
 set ^A("1","program")=6
 set ^A("1","disk")=3
 set ^A("1","laptop")=7
 set ^A("1","monitor")=1
```

```
set ^A("2","computer")=5
set ^A("2","printer")=2
set ^A("2","program")=6
set ^A("2","memory")=3
set ^A("2","laptop")=7
set ^A("2","language")=1

set ^A("3","computer")=5
set ^A("3","printer")=2
set ^A("3","disk")=6
set ^A("3","memory")=3
set ^A("3","laptop")=7
set ^A("3","USB")=1

set %=$zzDocCorrelate(^A,^B,"Cosine",.5)

for i="":$order(^B(i)):"" do
. write i,!
. for j="":$order(^B(i,j)):"" do
.. write ?10,j," ",^B(i,j),!
```

output:

```
1
         2 0.887096774193548
         3 0.741935483870968
2
         1 0.887096774193548
         3 0.701612903225806
3
         1 0.741935483870968
         2 0.701612903225806
```

Figure 32 $zDocCorrelate()Example

A call to **$zStopLookup(word)** will return 1 if *word* is in the stop list, 0 otherwise. Words presented to **$zStopLookup(word)** should be in lower case.

$SynInit() opens a synonym file. The file should consist of two or more words per line separated by from one another by one blank. The words are treated as synonyms with the first word on each line as the primary synonym. The primary synonym may be a code or category number. This word or code will be returned if any of the remaining words are passed as arguments to **$SynLookup()**. Figure 33 gives an example.

7.8 SQL functions

These functions are peculiar to this implementation..'

```
Assume that the file “stop” contains the word “and”

 set %=$zStopInit("stop")
 if $zStopLookup("and") write "yes",!

Writes yes

Assume that the file “synonyms” contains a line with the text:

compression compressions compress compressed compresses

 set %=$zSynInit("synonyms")
 write $zSynLookup("compressions"),!
```

```
output:

compression
```

Figure 33 Stop List Functions

7.8.1 $zsqlOpen

Returns *true* if a connection to the SQL server is open, *false* otherwise.

7.8.2 $zNative

$znative returns true if globals are being stored in the native global array.

7.8.3 $zSqlite[command[,option]]

$zsqlite with no arguments returns 1 if globals are being stored in Sqlite3, 0 otherwise.

7.8.3.1 $zSqlite("begin transaction")

Send a *BEGIN TRANSACTION;* command to Sqlite.

7.8.3.2 $zSqlite("commit transaction")

Send a *COMMIT TRANSACTION* ;command to Sqlite.

7.8.3.3 $zSqlite("savepoint"[,*savepoint*])

If the second argument is omitted, send a *SAVEPOINT default;* command to Sqlite.
If the second argument is present, send a *SAVEPOINT savepoint;* command to Sqlite where '*savepoint'* is the value passed as the second argument.

7.8.3.4 $zSqlite("rollback"[,*savepoint*])

If the second argument is omitted, send a *ROLLBACK TRANSACTION to default;* command to Sqlite.
If the second argument is present, send a *ROLLBACK TRANSACTION to savepoint;* command to Sqlite where '*savepoint'* is the value passed as the second argument.

7.8.3.5 $zSqlite("pragma",*option)*

A PRAGMA command will be sent to Sqlite with *option* as its argument. If the *PRAGMA* results in a returned value, it will be the returned result of the finction. Otherwise, the function will return 1 (success) or 1 (failure).

8 GTK Desktop GUI Apps

Several simplified GTK functions are included. These will alow you to create desktop GUI applications. These are functions that control GTK widgets in a graphical application.

8.1 Glade GUI Design Tool

The open source program *Glade* allows the user to design the layout of a desktop GUI app by dragging and dropping GUI widgets (buttons, text boxes, etc.) onto a canvas. Figure 34 gives an example that includes several widget types.

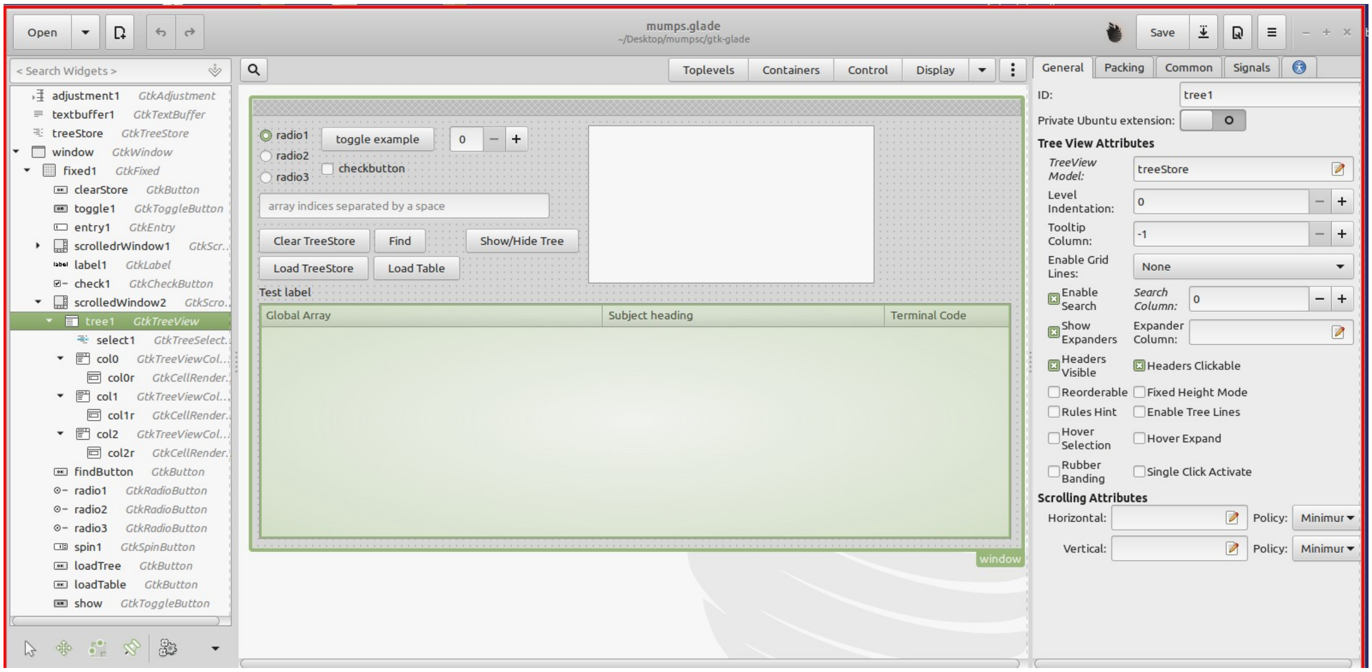

Figure 34 Glade Canvas

When you save a Glade canvas it appears in your directory as a file with the *.glade* extension. This is an XML file giving the details on your design.

Included with the Mumps distribution in the directory *gtk-glade* is a script file named *appBuild,script* and a Mumps program named *extractWidgets.mps*. The script file:

1. runs the Mumps file which reads the file *.glade* file from above and builds several files;
2. compiles (using the Mumps compiler) the file *gtk.mps* which includes the files from the previous step and creates an executable named *gtk* which will render the GUI application on the screen.

Among the files created by *extractWidgets.mps* are several files containing Mumps programs to service the actions to be performed by interacting with the on-screen GUI. There will be a file for each signal defined for each widget. The files will have names of the form:

on.widgetName.clicked.mps

where *widgetName* is the name of the widget as given in the *ID* field in the glade app and *clicked* is a signal established for that widget. The file will be invoked if the action associated with the signal is detected (for example, a button is clicked).

8.2 GTK Example

8.2.1 Glade Design Tool

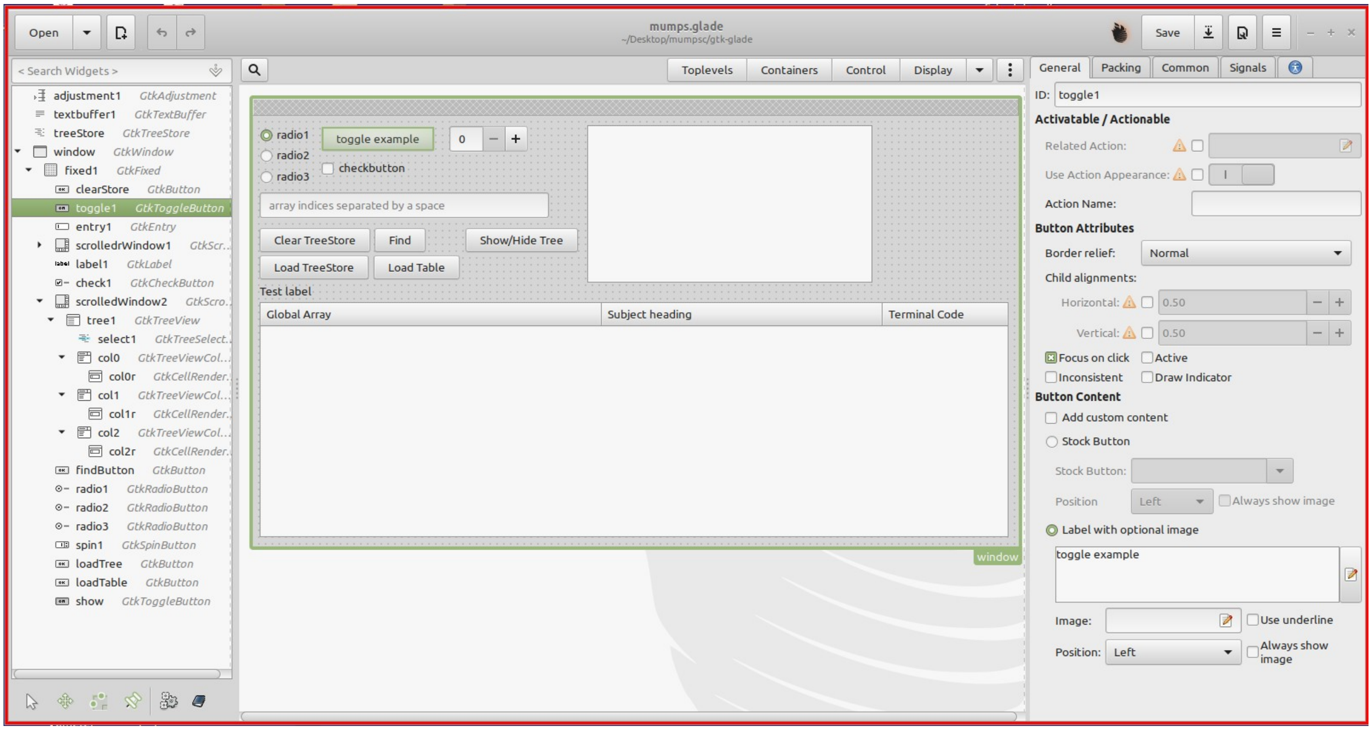

Figure 35 Toggle Button Screen 1

In Figure 35 you see the a Glade layout page. The center panel is the layout for the on-screen app that is being built. The various entities (widgets) have been dragged and dropped into their positions from widgets available in dropdown menus shown at the top named Toplevel, Containers, Control, and Display.

The leftmost panel contains the user assigned names (IDs) of the widgets along with an indication of their data types.

Some widgets are nested within others according to the display hierarchy. This, the GtkToggleButton named toggle1 is contained within the GtkFixed container named fixed1 which in turn is contained within the GtkWindow named window.

The rightmost panel contains tabs which show options for a selected widget. In this case, the selected widget is the toggle1 button which is highlighted in green in upper left of center panel and also as a row in panel one.

As can be seen in panels 1 and 3, the ID of the widget is toggle1 (user assigned), The widget is a GtkToggleButton (as seen in panel 1).

The text displayed in the button is set in panel 3 under *Label with Optional Image*. No image is assigned in this case.

Except for assigning the ID name of the widget and entering the text to appear in the button, the remainder of the options are defaults which are suitable for most ordinary applications.

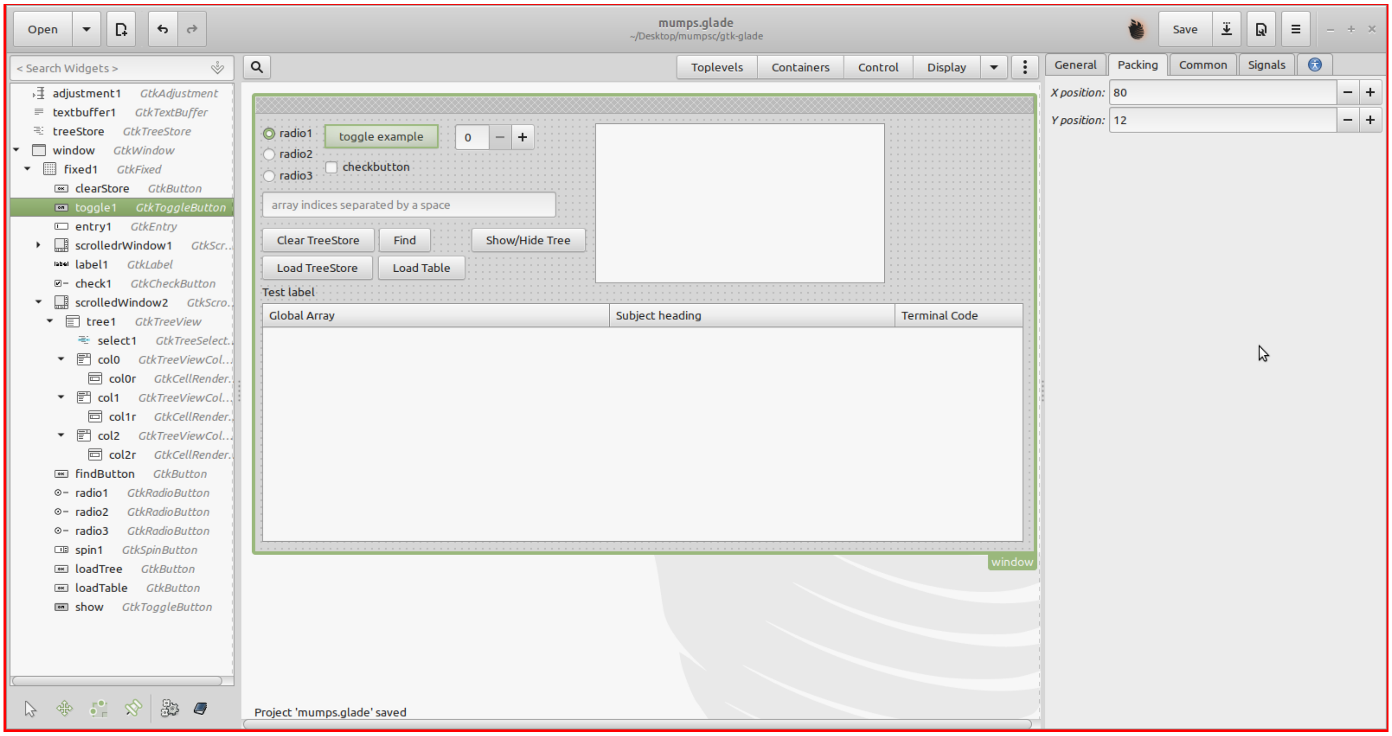

Figure 36 Toggle Button Screen 2

In Figure 36 the second tab of panel 3 has been selected. This panel determines the location of the widget within the window. Changing these numbers moves the widget accordingly.

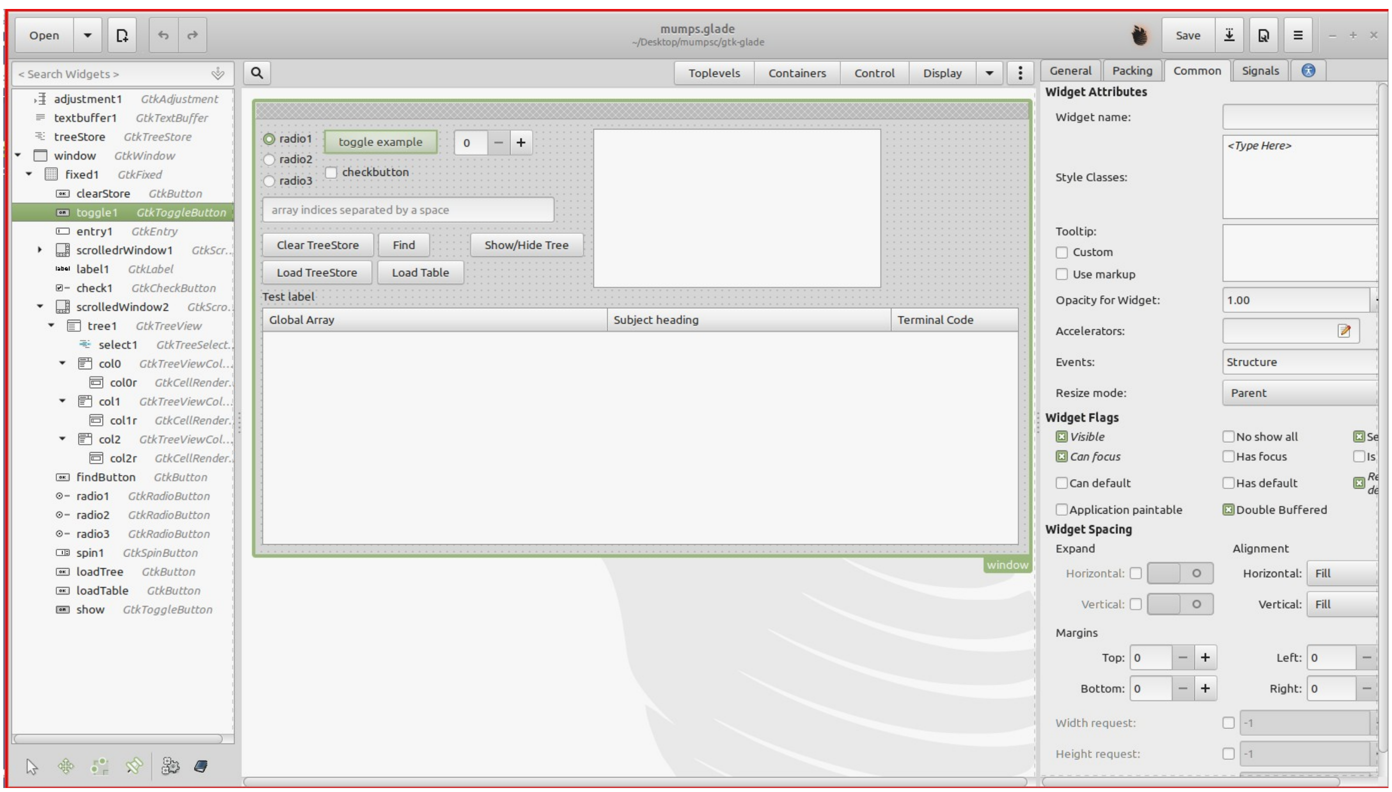

Figure 37 Toggle Button Screen 3

In the third tab of panel 3 are many adjustments all of which are defaults except for the height and width settings. These determine the size of the button. The height and width request boxes have been unchecked which causes the button to be sized to fit the contained text.

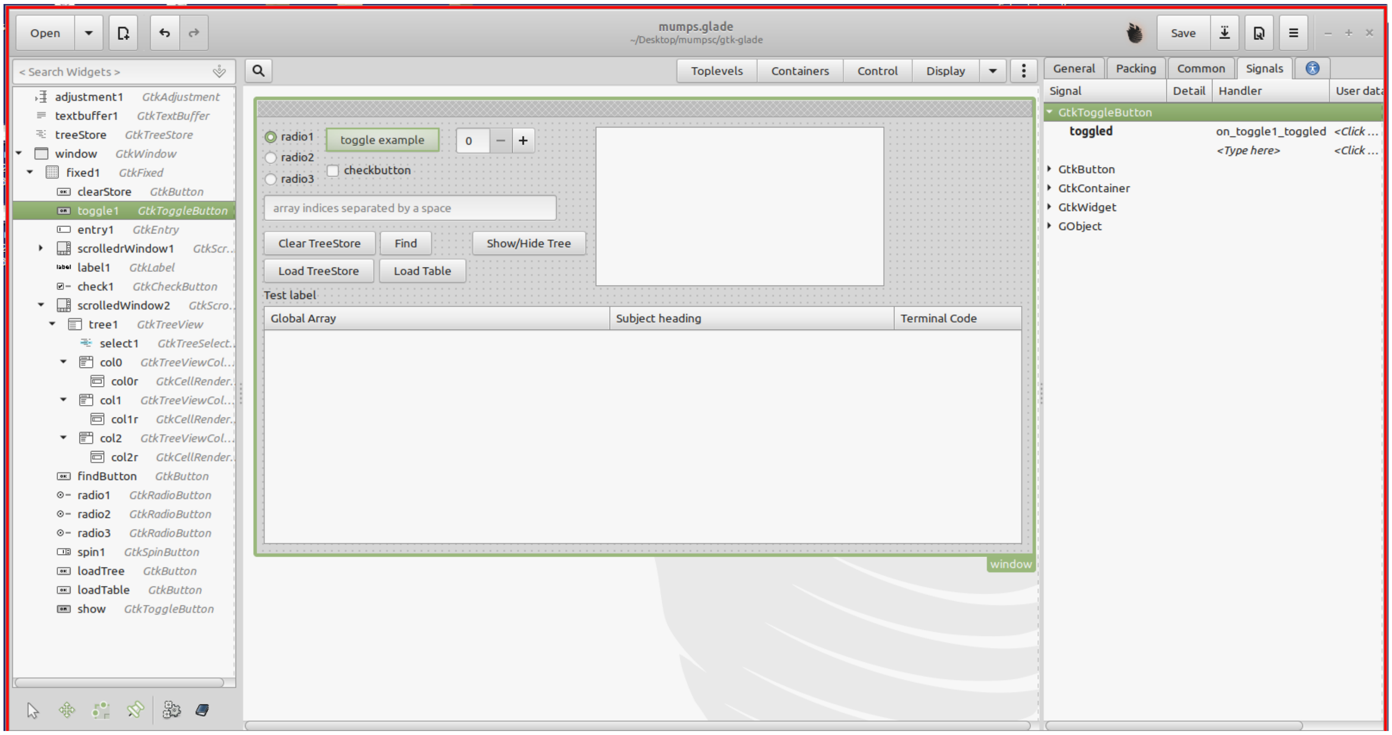

Figure 38 Toggle Button Screen 4

In Figure 38 we see the last tab of panel 3. This is the panel where you select the signals to be emitted for actions on the widget. Since this is a toggle button, the primary action is to click the button using the left button on your mouse. This action can emit a toggled signal.

If you want your program to process this signal, you enter the name of the routine to be called should the signal emit. In this case, the function named *on_toggle1_toggled* will be called if the button is clicked. The GTK GUI manager will cause the button to appear depressed or not depressed after successive clicks. Your function can determine the state of the button by using a system function.

When you save a Glade layout, it is saved as an XML file with the extension *.glade*.

8.2.2 Building A Mumps App from The Glade XML File

The disk representation of a Glade design is a XML file. For purposes of building a Mumps program from this file, the file needs to be named *mumps.glade*.

In the above we highlighted the toggl1 toggle button. The Glade XML for that button looks like:

```
<child>
  <object class="GtkToggleButton" id="toggle1">
    <property name="label" translatable="yes">toggle example</property>
    <property name="visible">True</property>
    <property name="can_focus">True</property>
    <property name="receives_default">True</property>
    <signal name="toggled" handler="on_toggle1_toggled" swapped="no"/>
  </object>
  <packing>
    <property name="x">80</property>
    <property name="y">12</property>
  </packing>
</child>
```

The above is a fragment of the larger Glade file which is 299 lines in length. The XML tells us that the name of the widget (toggle1), its data type (GtkToggleButton), its label contents (toggle example), any signals it emits (toggled) and the name of the signal handlers (on_toggle1_toggled). It also gives the location of the button on the app window and other information concerning its appearance and performance.

The distro program *extractWidgets.mps* reads the XML file and generates files that are used to compile and service an application. These are:

8.2.2.1 gtk1.h

This file contains C declarations for all the widgets defined in the XML file. It also includes the relevant GTK header files. In the case of the toggle one widget, the line:

```
GtkToggleButton *toggle1;
```

appears, among others.

8.2.2.2 gtk2.h

This file contains code that will invoke a Mumps signal handler (see below) for each signal emitted for a widget. In the case of the toggle1 widget, this code looks like:

```
toggle1=GTK_TOGGLE_BUTTON(gtk_builder_get_object(builder,"toggle1"));
{ char tmp[128]; sprintf(tmp,"%p", toggle1);
 SymPut("toggle1",tmp); fprintf(f," set toggle1=\"%s\"\n",tmp); }
```

The above code fragment which will be compiled into the base program *gtk.mps* builds the internal data structure and screen representation associated with the widget by means of *gtk_builder_object().* This function reads the *mumps.glade* XML file information for the parameter *toggle1.* The function returns a pointer to the object which is stored in the GtkToggleButton pointer *toggle1* (the names of the widgets and the internal pointers as usually the same, both are *toggle1* in this case).

The string value of the pointer is stored in the Mumps symbol table (*SymPut()*) and a string containing the Mumps command or the form: *set toggle1=0x123456* is written to the file gtk4.mps.

8.2.2.3 gtk3.h

This file contains the basic signal handlers (written in C) which are used to invoke the corresponding Mumps programs which will actuall handle the signal. The code for the toggle1 widget looks like:

```
extern "C" void on_toggle1_toggled(GtkWidget *w)
{struct MSV * Ptr = AllocSV(); char tmp[512];
sprintf(tmp,"set widget=\"%p\" g ^on.toggle1.toggled.mps",w);
Interpret((const char *) tmp, Ptr); free(Ptr);}
```

This fragment establishes the signal handler (*on_toggle1_toggled()*), creates an instance of the Mumps state vector (MSV *Ptr), creates a string consisting of Mumps *set* and *goto* (g) *commands* with the string value of the widget *w* as the right hand side of the *set* command.

The subject of the goto command is a file named *^on.toggle1.clicked.mps* which will contain the Mumps code to process the signal.

Next, it then invokes the mumps interpreter (*Interpret()*) which executes the commands in *tmp*.

The first line specifies that the calling conventions for this function will follow C language rules. This is because the Mumps interpreter is actually a collection of C++ programs and the basic GTK library is written in C.

8.2.2.4 gtk4.h

This file is created when the actual application is run. It writes, for each widget, a Mumps set command that establishes the address of the data structure for the widget. In the case of the toggle1 example, this looks like:

```
set toggle1="0x55ab6337e230"
```

When the Mumps signal handler is invoked, the file containing this information will be run by the signal handler thus giving the signal handler the memory references of all widgets in the application.

8.2.2.5 gtk.mps

This is the main routine that is compiled by the Mumps compiler. It will start the GTK GUI system. It looks like:

```
# Jan 30, 2022
+ #include "gtk1.h"
        zmain
+ #include "gtk2.h"
        do ^gtk4.h
+ gtk_main();
        write "Goodbye!",!
        zexit
+ #include "gtk3.h"
```

The lines that begin with a plus sign are passed directly to the C++ compiler. The function *gtk_main()* passes control to the GTK runtime routines. Return is only made upon program termination.

The first #include brings in the global widget declarations (in C++). The second #include incorporates all the builder calls which create the widgets on the screen and their associated data structured. The third #include brings in the C++ signal handlers for all signals used by the widgets.

8.2.2.6 on.toggle1.toggled.mps

The actual Mumps signal handler created by *extractWidgets.mps*, named *on.toggle1.toggled.mps* looks like:

```
#!/usr/bin/mumps

#       Mumps GTK Signal Handler

 do ^gtk4.h
 write "on.toggle1.toggled.mps"," ",widget,!
 write $z~mdh~toggle~button~get~active(toggle1),!
```

The function *$z~mdh~toggle~button~get~active(toggle1)* returns 0 or 1 depending if the button is not depressed or depressed. In this case of the function, it's Mumps reference (toggle1) was used but the variable *widget* is also present which contains a pointer to the data structure of the widget (toggle1 in this case) which emitted the signal.

You're on your own from here.

8.3 MDH Functions

8.3.1 $z~mdh~toggle~button~get~active(ToggleButtonReference)

Returns 0 if the button is inactive, 1 if active

8.3.2 $z~mdh~toggle~button~set~active(ToggleButtonReference,intVal)

Sets the button to active if intVal is 1, inactive if the value is 0.

8.3.3 $z~mdh~dialog~new~with~buttons(ParentWindowRef,dialog)

Raises a Gtk Dialog window displaying the contents of *dialog* with buttons **Yes** and **No**. Returns 1 if **Yes** is clicked; 0 if **No** is clicked; and -1 if the box is dismissed.

8.3.4 $z~mdh~entry~get~text(EntryReference)

Returns the current string contents of the referenced Entry box.

8.3.5 $z~mdh~entry~set~text(EntryReference,value)

Sets the contents of the named entry box to *value*.

8.3.6 $z~mdh~text~buffer~set~text(TextBufferReference,string)

Sets the contents of the referenced text buffer to the value of string.

8.3.7 $z~mdh~label~set~text(LabelReference,string)

Sets the text contents of the label referenced to string. Triggers a value changed signal.

8.3.8 $z~mdh~tree~selection~get~selected(TreeModelReference,column)

Returns value in designated column of referenced TreeModel.

8.3.9 $z~mdh~tree~store~clear(TreeStoreReference)

Clears (deletes) the contents of the referenced TreeStore.

8.3.10 $z~mdh~tree~level~add(TreeStoreReference,treeDepth,index,data[,...])

Add index at tree level treeeDepth to column 1 of TreeStore. Add additional data items in successive columns.

8.3.11 $z~mdh~spin~button~get~value(SpinButtonReference)

Returns the current value of the referenced SpinButton.

8.3.12 $z~mdh~spin~button~set~value(SpinButtonReference,number)

Sets the current value of the referenced spin button to number.

8.3.13 $z~mdh~widget~hide(widgetReference)

Hides the widget from view.

8.3.14 $z~mdh~widget~show(widgetReference)

Displays (un-hides) the widget.

9 Pattern Matching

9.1 Mumps 95 Pattern Matching

Author: Matthew Lockner

Mumps 95 compliant pattern matching (the '?' operator) is implemented in this compiler/interpreter as given by the following grammar:

```
pattern         ::= {pattern_atom}
pattern_atom    ::= count pattern_element
count           ::= int | '.' | '.' int | int '.' | int '.' int
pattern_element ::= pattern_code {pattern_code} | string | alternation
pattern_code    ::= 'A' | 'C' | 'E' | 'L' | 'N' | 'P' | 'U'
alternation     ::= '(' pattern_atom {',' pattern_atom} ')'
```

The largest difference between the current and previous standard is the introduction of the alternation construct, an extension that works as in other popular regular expressions implementations. It allows for one of many possible pattern fragments to match a given portion of subject text.

A string literal must be quoted. Also note that alternations are only allowed to contain pattern atoms and not full patterns; while this is a possible shortcoming, it is in accordance with the standard. It is a trivial matter to extend alternations to the ability to contain full patterns, and this may be implemented upon sufficient demand.

Pattern matching is supported by the Perl-Compatible Regular Expressions library (PCRE). Mumps patterns are translated via a recursive-descent parser in the Mumps library into a form consistent with Perl regular expressions, where PCRE then does the actual work of matching. Internally, much of this translation is simple character-level transliteration (substituting '|' for the comma in alternation lists, for example). Pattern code sequences are supported using the POSIX character classes supported in PCRE and are mostly intuitive, with the possible exception of 'E', which is substituted with `[[:print][:cntrl:]]`. Currently, this construct should cover the ASCII 7-bit character set (lower ASCII).

Due to the heavy string-handling requirements of the pattern translation process, this module uses a separate set of string-handling functions built on top of the C standard string functions, using no dynamic memory allocation and fixed-length buffers for all operations whose length is given by the constant STR_MAX in *sysparms.h*. If an operation overflows during the execution of a Mumps compiled binary, a diagnostic is output to *stderr* and the program terminates. If such termination occurs too frequently, simply increase the value of STR_MAX.

9.2 Using Perl Regular Expressions

Author: Matthew Lockner

In addition to Mumps 95 pattern matching using the '?' operator, it is also possible to perform pattern matching against Perl regular expressions via the *perlmatch* function. Support for this functionality is provided by the Perl-Compatible Regular Expressions library (PCRE), which supports a majority of the functionality found in Perl's regular expression engine.

The *perlmatch* function works in a somewhat similar fashion to the '?' operator. It is provided with a subject string and a Perl pattern against which to match the subject. The result of the function is boolean and may be used in boolean expression contexts such as the "If" statement.

Some subtleties that differ significantly from Mumps pattern matching should be noted:

1. A Mumps match expects that the pattern will match against the entire subject string, in that successful matching implies that no characters are left unmatched even if the pattern matched against an initial segment of the subject string. Using *perlmatch*, it is sufficient that the entire Perl pattern matches an initial segment of the subject string to return a successful match.
2. The *perlmatch* function has the side effect of creating variables in the local symbol table to hold *backreferences*, the equivalent concept of $1, $2, $3, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming

scheme as Perl ($1 through $9). These variables remain defined up to a subsequent call to *perlmatch* , at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6 through $9 will contain the null string.

Examples

This program asks the user to input a telephone number. If the data entered looks like a valid telephone number, it extracts and prints the area code portion using a backreference; otherwise, it prints a failure message and exits.

```
Write "Please enter a telephone number:",!
Read phonenum

If $$^perlmatch(phonenum,"^(1-)?(\(?\d{3}\)?)?(-| )?\d{3}-?\d{4}$") Do
. Write "+++ This looks like a phone number.",!
. Write "The area code is: ",$2,!
Else  Do
. Write "--- This didn't look like a phone number.",!
```

The output of several sample runs of the program follows:

```
Please enter a telephone number:
1-123-555-4567
+++ This looks like a phone number.
The area code is: 123


Please enter a telephone number:
(123)-555-1234
+++ This looks like a phone number.
The area code is: (123)


Please enter a telephone number:
(123) 555-0987
+++ This looks like a phone number.
The area code is: (123)
```

As in Perl, sections of the regular expression contained in parentheses define what is contained in the backreferences following a match operation. The backreference variables are named in a left-to-right order with respect to the expression, meaning that $1 is assigned the portion matched against the leftmost parenthesized section of the regular expression, with further references assigned names in increasing order. For a much more in-depth treatment of the subject of Perl regular expressions, refer to the *perlre* manpage distributed with the Perl language (also widely available online).

10 Mumps Compiler

Included in the distribution package is (1) a beta version compiler for the Mumps language and (2) the Multi-Dimensional and Hierarchical library (MDH). At present, not all Mumps language features are implemented but many are. There is a companion document entitled *MDH.pdf* which provides additional details on the MDH package.

The Mumps Compiler translates Mumps source code to C++ and then compiles the resulting C++ programs into executable binaries.

The MDH package consists of a C++ class library which permits C++ programs to be written using many of the database and string handling features of Mumps.

10.1 Compiling Programs

The Mumps programs described in this document can be run in either of two ways: either as interpreted code using the Mumps interpreter or as binary executables resulting from the Mumps Compiler.

Binary programs run faster than interpreted programs but the difference can be small if the programs rely heavily on input/output operations.

10.2 How to Compile and Run a Mumps or MDH Program.

Programs written in Mumps must have the extension *.mps* when used with the compiler. Programs written for the interpreter, however, may have any extension however *.mps* is preferred. MDH programs written in C++ must have the ".cpp" extension.

When you compile a Mumps program, a C++ translation of your program is created and resides on the disk with the same name but with the *.cpp* extension. The C++ translation is then compiled and linked with run-time libraries to build an executable binary.

On MS Windows, the binary will have the same name as your original program but with the .exe extension. On Linux, the binary will have the same name as your original program but with no extension. Depending on which system you are using, there will be other, intermediate files generated by the Mumps and C++ compilers. These are not important and can be deleted.

You may compile a Mumps program of an MDH C++ program by using the executable script *mumpsc*. To compile a Mumps or MDH C++ program using the script, type:

```
mumpsc myprog.mps
```

If the name of the file presented as an argument to *mumpsc* has the extension *.mps,* the script will first translate the Mumps to C++ and then compile the result and link the output of the C++ compiler with MDH and standard Mumps libraries.

If the name of the file presented as an argument to *mumpsc* has the extension *.cpp,* the script will compile the C++ program and make available the MDH class library.

As noted above, the script *mumpsc* first translates a Mumps program to C++ and then compiles the result. The program that translates Mumps to C++ is named *mumps2c*. You may run this program standalone:

```
mumps2c myprog.mps
```

The result will be a file named *myprog.cpp.* You may edit or modify this file and then compile it to binary executable with the *mumpsc* script. Since the output of *mumps2c* requires access not only to the MDH object libraries but also some uncommon system libraries, usage of the *mumpsc* script is required (*i.e.* don't use *g++*).

10.3 Compiler Error Messages

Generally speaking, in most cases you will receive syntax error messages from the Mumps compiler which will identify the error and the line number in the original Mumps program containing the error.

However, in some cases, an error may not be detected by the Mumps compiler but, instead, by the C++ compiler.

Consequently, if you get C++ error messages, the line number on the error message will refer to the line number in the C++ translation of your Mumps program. To reference this to a line number in your Mumps program, look into the generated *.cpp* file at the line number given by the C++ error message and then back track to the nearest prior commented Mumps source line - this shows the original in your Mumps programs that caused the problem.

For example, if you get a message from the C++ compiler saying that you have an error at line 1234 in the C++ module, open the C++ file and move to line 1234. At that location you may see something like:

```
/*===================================================================================*
svPtr->LineNumber=4; //       write "the sum is: ",total,!
/*===================================================================================*/
       if (svPtr->out_file[svPtr->io]==NULL) ErrorMessage("Write to input file",svPtr->LineNumber);
       svPtr->hor[svPtr->io]+=fprintf(svPtr->out_file[svPtr->io],"%s","the sum is: ");
       if (sym_(SYMGET,(unsigned char *) "total",(unsigned char *) tmp0,svPtr)==NULL)
            VariableNotFound(svPtr->LineNumber);
       svPtr->hor[svPtr->io]+=fprintf(svPtr->out_file[svPtr->io],"%s",tmp0);
       fprintf(svPtr->out_file[svPtr->io],"\n"); svPtr->hor[svPtr->io]=0; svPtr->ver[svPtr->io]++;
```

Figure 39 Example C++ Code

Notice that each original line of Mumps code and its line number in the original Mumps file appear in a comment prior to the C++ translation of the line. Note that the translation of a line of Mumps code may result in many lines of C++ code.

Thus, to locate the line of Mumps code that caused the C++ error, look for the line of Mumps code preceding the line which the C++ complier flagged as being in error.

Generally speaking, you may receive C++ error messages if you reference non-existent labels or subroutines, or incorrectly specify indented do blocks (see below).

Also, you may see ^M (control-M) characters in the code. These are visible due the differences between the operating systems. Under Windows, each line ends in a *carriage-return* and a *line-feed*. Under Linux, each line ends in a *line-feed* character only. The control-M's you see are the carriage-returns. They are harmless and may be ignored.

10.4 Global Array Storage in Compiled Programs

Global arrays will be stored in Sqlite or the native Btree database depending on which script you used to build the interpreter with. Global arrays created by compiled programs are interchangeable with global arrays created by the interpreter.

10.5 Compiler Implementation Overview

The compiled modules execute faster than the same code executing on the interpreter depending upon the nature of the code and the amount of database activity. Programs will large amounts of database or I/O activity will run at about the same speed.

One advantage of full compilation is interoperability with other languages and with the host operating system. Programs written in C++ have full access to all system features and can be manually edited to improve performance.

11 Multi-Dimensional and Hierarchical Database Class Library (MDH)

The Multi-Dimensional and Hierarchical Toolkit (MDH) is a Linux-based, open sourced, toolkit of libraries that support access to the Mumps database and other services. The package is written in C and C++ and licensed under the GNU GPL/LGPL licenses. Full details are provided in a companion document (*MDH.pdf*)

The toolkit permits manipulation of very large, character string indexed, multi-dimensional, sparse matrices from C++ programs. The toolkit supports access to SQL relational data base servers, the Perl Compatible Regular Expression Library, and the Glade GUI builder.

The toolkit makes Mumps data base and functions available as C++ classes and permits execution of Mumps scripts directly from C++ programs. The toolkit is provided with the Mumps distribution and is available if Mumps is installed. No further installation beyond the basic Mumps installation described above is required.

The class, function and macro libraries primarily operate on global arrays. Global arrays are undimensioned, string indexed, disk resident data structures whose size is limited only by available disk space. They can be viewed either as multi-dimensional sparse matrices or as tree structured hierarchies.

To compile an MDH/C++ program using the script, type:

```
mumpsc myprog.cpp
```

11.1 MDH Class Library Header File

To use the class libraries, add the following to the beginning of your C++ program:

```
#include <mumpsc/libmpscpp.h>
```

This statement inserts in the necessary header files for you C++ program. In addition to the MDH class libraries, the following standard systems headers will be included as well:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <string.h>
#include <math.h>
#include <stdlib.h>
```

11.2 MDH Data Types

The MDH is built upon two data classes. One is for global arrays (**global**)and the other is a string data type (**mstring**) which mimics Mumps strings.

11.2.1 Mstring Data Objects

The **mstring** class provides functionality similar to the basic typeless string data type in Mumps. Objects of **mstring** may contain text, integers and floating point values. Operations on **mstring** objects include addition, multiplication, subtraction, division, modulo, concatenation and so forth. Objects of type **mstring** are declared in the normal manner such as:

```
mstring mvar1,var2,var3;
```

They may be initialized with **int**, **long**, **float**, **double**, **char *** and **string** and **mstring** values such as:

```
mstring var1(10),var2(10.123),var3("test"),var4(stringVar);
```

Objects of type **mstring** may be assigned to most data types and most data types may be assigned to objects of type **mstring**.

Objects of type **mstring**, **string**, and null terminated character strings are the only legal indices for objects of class **global**.

11.2.1.1 Arithmetic Operations on Mstring Objects

When **mstring** objects contain numeric values, you may apply arithmetic operators directly to the **mstring** object or objects.

Both extended precision and basic hardware precision are available.

In hardware precision mode, floating point numbers are processed by the machine's arithmetic processing hardware. Floating point numbers are treated as 64-bit *double* values and integers are treated as signed 64-bit *long* integer values. Thus, integers may range from:

-9,223,372,036,854,775,808 ($-2^{63}+1$) to 9,223,372,036,854,775,807 ($2^{63}-1$)

Hardware floating point numbers utilize a one bit sign, an 11 bit exponent and a 52 bit fraction. This translates into approximately 16 decimal digits of precision in the range of $\pm \sim 10^{-323.3}$ to $\pm \sim 10^{308.3}$.

Extended precision is available through use of the GNU multiple precision arithmetic library[11] and the GNU MPFR library[12]. For integers, this means effectively unlimited precision. For floating point, the exponent is 64 bits and the fraction is user specified (default of value of 72 bits).

Hardware arithmetic will be selected during system build if (1) *configure* does not find the extended precision libraries or (2) the user specifies the configuration option:

--with-hardware-math.

If the extended precision libraries are found and the above option has *not* been specified, extended precision will be in effect.

If extended precision is used, the number of bits in the fraction of a floating point number can be set with:

--with-float-bits=value

where *value* is the number of bits. The default value is 72.

For extended precision floating point numbers, the number of digits of precision that may be printed is controlled by:

--with-float-digits=value

where *value* is the number of digits. The default is 20.

When printing an extended precision floating point number, the number of digits being printed should be consistent with the number of bits in the fraction. If the number of digits is too large, insignificant, random low-order digits may appear in the output.

11.3 Global Data Objects

Objects of class **global** provide access to the global array database. The class includes functions to create, delete (kill), and navigate global arrays.

In your C++ program, you must declare each global array that the program will use. Normally, these declarations will appear at the beginning of the program. A global declaration has the form:

```
global program_ref(database_name);
```

Where *program_ref* is the name by which the global array will be referred to in your program and *database_name* is the name of the actual global array in the file system. Both may be the same. The value for *database_name* may be expressed as a pointer to a character string constant.

11 http://www.mpfr.org/
12 http://gmplib.org/manual/index.html

For example, if your program uses a Mumps global array stored in the file system with the name *patient*, you might have the following C++ declaration in your program:

```
global patient("patient");
```

Once declared, a global array object may be used to access the contents of the global array database. For example, for the global array object *patient* declared above, the following reference might be made:

```
patient(ptid,test,date,time)=result;
```

where *ptid, test, data, result* and *time* are **mstring** or **char *** null terminated variables or constants.

Although objects of class **mstring** may be C++ arrays, objects of class **global** may not.

Objects of class **global** may *not* be initialized in declaration statements**.**

11.4 Operators Defined on Mstring & Global Objects

Objects of class **mstring** may appear as the operands of most C++ builtin operators by means of C++ operator overloading.

In the cases of binary operators, the other operand may be most other builtin data types as well as **global** and **mstring** objects.

Figure 40 contains the full list of C++ operators that have been overloaded for use with objects of types **mstring** and **global**. In these examples, assume the declarations:

```
mstring ms, msa[10];
global gb("test");
```

Unary Operators	Description	Examples
++ --	Suffix/postfix increment and decrement	ms++; gb("123")++;
[]	Array subscripting[13]	mstring msa[10]; msa[1] = "abc";
++ --	Prefix increment and decrement	++ms; ++gb("123");
+ -	Unary plus and minus	cout << +gb("123") << endl; cout << -ms << endl;
(*type*)	C-style explicit cast	ms = "123" int k = (int) ms("123");
*	Indirection (dereference)	global *p1 = &gb; (*p1)("111") = 10; mstring *p2 = msa; (*p2)[3] = "abc";
& (unary)	Address-of	mtstring *p1 = &ms;
new, new[]	Dynamic memory allocation	global *p3 = new global("xxx"); (*p3)("xxx") =2 2; mstring *p4 = new mstring; *p4=123;

13 Only with an **mstring** operand.

delete, delete[]	Dynamic memory deallocation	delete p1;
Binary Operators[14]	Description	Examples
* / %	Multiplication, division, and remainder	ms = ms * 2; ms = gb("123") / ms; ms = gb("123") % 5;
+ -	Addition and subtraction	ms = ms + 2; ms = gb("123") - ms;
<< >>	stream insertion / extraction	cout << ms; cin >> gb("123");
< <=	For relational operators < and ≤ respectively[15]	if (ms <= gb("123")) ... if (ms < gb"abc")) ... if ("abc" < gb("123") ...
> >=	For relational operators > and ≥ respectively[15]	if (ms >= gb("123") ... if (ms > gb("abc")) ... if ("abc" > gb("123")) ...
== !=	For relational operators = and ≠ respectively[15]	if (ms == gb("123") ... if (ms != gb("abc")) ...
&&	Logical AND	if (ms && gb("123")) ...
||	Logical OR	if (ms || gb("123")) ...
Ternary Operator	Description	Examples
?:	Ternary conditional	ms ? ms : y
Assignment[16]	Description	Examples
=	Direct assignment	ms = 123 gb("123") = 1.3456 ms = "test"
+= -=	Compound assignment by sum and difference	ms=0; ms += 123 ms+="123"; gb("123")=0; gb("123") -= 10
*= /= %=	Compound assignment by product, quotient, and remainder	ms=0; ms *= 123 gb("123")=10; gb("123") /= 10 gb("123")=10; gb("123") %= 10
& (binary)	Concatenate. First operand must be of type **global** or **mstring**[17]. The second operand may be **string**, **mstring**, **global**, **char*** **int**, **long**, or **double**.	mstring i="aaa",j="bbb",k="ccc"; i=i&j&k; // i -> aaabbbccc

Figure 40 Operators Defined on **mstring** and **global**

11.5 Example Arithmetic Operations on global and mstring Objects

The operations of add, subtract, multiply, divide, pre/post increment and pre/post decrement are defined (overloaded) for **global** and **mstring** variables either together (in binary or the ternary operator) or in connection with other builtin data types. The contents of the **global** array node or **mstring** variable must be compatible with the dominant data type of the operation. If the contents not compatible with the operation (example, incrementing a string of text), the value of the **global** will be interpreted as zero. Examples:

Code Examples	Results

14 One operand, the first, may be of type **mstring** or **global** and the other may be of type **mstring**, **global**, **float**, **double**, **int**, **long**, **char***, or **string**.

15 If one operand is a numeric type (**long**, **float** etc.), the **mstring** or **global** will be interpreted as a numeric value rather than as a string. If both operands are of type **global** or **mstring**, they will be compared as strings. If one operand is of type **global** or **mstring** and the other is of type **char*** or **string**, they will be compared as strings.

16 The left-hand-side must be of type **mstring** or **global** while the right-hand-side may be of types **mstring**, **global**, **float**, **double**, **int**, **long**, **char***, or **string**. When arithmetic assignment operators are used, right-hand-side **string**, **char***, and **global** operands will be converted to numeric following the default Mumps conversion rules.

17 Note: because the overloaded bitwise *and* operator (&) is of lower precedence than the bit shift operator <<, in output operations (such as when using *cout*), an expression involving the bitwise & operator must to be in parentheses.

`global gbl("gbl");`	
`int i, j=10;` `string a = "10", b = "20", c = "30";` `char aa[] = "10", bb[] = "20", cc[] = "30";` `mstring aaa = "10", bbb = "20", ccc = "30";`	
`gbl.Kill();`	
`gbl(a,b,c) = 10;` `gbl(aa,bb,cc) = 20;` `gbl(aaa,bbb,ccc) = 30;`	
`i = gbl(a,b,c) + 20;` `cout << i << endl;`	50
`i = 20 + gbl(a,b,c);` `cout << i << endl;`	50
`i = gbl(a,b,c) / j;` `cout << i << endl;`	3
`i = gbl(a,b,c) * 2;` `cout << i << endl;`	60
`gbl(a,b,c) ++;` `cout << gbl(a,b,c) << endl;`	31
`gbl(a,b,c) --;` `cout << gbl(a,b,c) << endl;`	30
`i = ++ gbl(a,b,c);` `cout << i << " " << gbl(a,b,c) << endl;`	31 31
`i = gbl(a,b,c) ++;` `cout << i << " " << gbl(a,b,c) << endl;`	31 32
`gbl(a,b,c) += 10;` `cout << gbl(a,b,c) << endl;`	42
`gbl(a,b,c) -= 10;` `cout << gbl(a,b,c) << endl;`	32
`gbl(a,b,c) *= 2;` `cout << gbl(a,b,c) << endl;`	64
`gbl(a,b,c) /= 2;` `cout << gbl(a,b,c) << endl;`	32
`aaa="aaa"; bbb="bbb"; ccc="ccc";` `cout << (aaa&bbb&ccc) << endl;`	aaabbbccc

Figure 41 Code Examples

11.6 Functions for Global and Mstring Objects

As is the case with Mumps functions, characters in strings are counted beginning with one, not zero. Thus, the substring beginning at position 3 through and including position 5 in the string *"abcdef"* is *"cde"*.

If an object of type **mstring** contains a string that is to be used as a global array reference in connection with one of the functions below, the global array reference must be preceded by a circumflex character (^) as is the case in Mumps and, also, the indices must be constants. Example:

```
mstring x="^g(1)";
cout x.Qlength() << endl; // prints 1
```

Function Parameters
INT An expression involving **int**, **long**, **float**, **double**, **mstring** or **global** the result of which can be interpreted as an integer. Data of type **char*** may not be used.
STR An expression involving **int**, **long**, **float**, **double**, **mstring** or **global** the result of which can be interpreted as a string. Data of type **char*** may be used but not as part of an expression.

Function	Description
`int mstring::Ascii( [INT] )` `int global::Ascii( [INT] )`	Returns the decimal value of the first ASCII character in the invoking **global** or **mstring**. If an integer argument is given, it returns the decimal value of the character at the offset designated by the argument. **mstring** and **global** arguments will be interpreted as integers. `mstring s1="abcdef";` `s1.Ascii()  -> 97` `s1.Ascii(2) -> 98`
`void mstring::Assign(global)` `void mstring::Assign(mstring)` `void mstring::Assign(string)` `void mstring::Assign(char*)` `void mstring::Assign(int)` `void mstring::Assign(long)` `void mstring::Assign(double)`	Assign a value to the global array reference containg in the invoking **mstring**. Contents of invoking **mstring** must conform to Mumps global array naming conventions and all indices must be constants, global array references, or variables previously defined in the Mumps Interpreter symbol table (see: *SymPut()*). Items placed in the Mumps Interpreter symbol table are discarded when the program ends. This function throws a *MumpsGlobalException* in the event of error. `mstring x="^g(1,1)";` `global g("g");` `x.Assign("test test");` `cout << g(1,1) << endl; // -> test test` `SymPut("a","1"); // a put in symTab` `x="^g(a,a)";  // reference uses a` `x.Assign("abc");` `cout << g(1,1) << endl; // -> abc` `g(1)=1;` `x="^g(^g(1),^g(1))";` `x.Assign("xyz");` `cout << g(1,1) << endl; // -> xyz`
`double global::Avg()`	Returns the average of the values of data bearing nodes beneath the given global array reference. `global a("a");` `for (i=0; i<1000; i++)` `  for (j=1; j<10; j++)` `     a(i,j) = j;` `a("100").Avg() -> avg below node a("100")` `a().Avg()      -> average of all nodes`
`void global::Centroid(global B)`	A centroid vector B is calculated from the invoking two dimensional **global** array matrix. An element of the centroid vector is the average of the values of each for the corresponding column of the matrix. Any previous

	contents of the **global** array named to receive the centroid vector are lost. The invoking **global** array must contain at least two dimensions. `global A("A");` `global B("B");` `mstring i,j;` `for (i=0; i<10; i++)` ` for (j=1; j<10; j++)` `  A(i,j) = 5;` `A().Centroid(B());` `mstring a="";` `while (1) {` ` a=B(a).Order();` ` if (a=="") break;` ` cout << a << " --> " << B(a) << endl;` ` }` `Yields:` `    1 --> 5` `    2 --> 5` `    3 --> 5` `    4 --> 5` `    5 --> 5` `    6 --> 5` `    7 --> 5` `    8 --> 5` `    9 --> 5`
`mstring mstring::Concat(char *)` `mstring mstring::Concat(global)` `mstring mstring::Concat(mstring)` `mstring mstring::Concat(string)` `mstring mstring::Concat(int)` `mstring mstring::Concat(long)` `mstring mstring::Concat(double)` `mstring global::Concat(string)` `mstring global::Concat(global)` `mstring global::Concat(char *)` `mstring global::Concat(mstring)` `mstring global::Concat(int)` `mstring global::Concat(long)` `mstring mstring::Concat(double)`	Returns **mstring** consisting of the value from the invoking object concatenated with the value of the parameter `mstring a="aaa",b="bbb",c;` `c=a.Concat(b); // c contains aaabbb`
`long global::Count()`	Returns the number of data bearing nodes beneath the given global array reference. `global a("a");` `mstring i,j;` `for (i=1; i<11; i++)` `  for (j=1; j<11; j++)` `    a(i,j) = 5;` `a().Count()    -> 100` `a("5").Count() -> 10`
`void global::DocCorrelate(global B,` `      mstring fcn, double threshold)` `void global::DocCorrelate(global B, char *` `      fcn, double threshold)`	DocCorrelate() builds a square *document-document* correlation matrix from the invoking **global** array *document-term matrix*. The name of the function to be used in calculating the *document-document* similarity is given by *fcn* and may be *Cosine*, *Jaccard*, *Dice*, or *Sim1*. The minimum correlation threshold is given in *threshold*

	which defaults to 0.80 if omitted. `global A("A");` `global B("B");` `long i,j;` `A("1","computer")=5;` `A("1","data")=2;` `A("1","program")=6;` `A("1","disk")=3;` `A("1","laptop")=7;` `A("1","monitor")=1;` `A("2","computer")=5;` `A("2","printer")=2;` `A("2","program")=6;` `A("2","memory")=3;` `A("2","laptop")=7;` `A("2","language")=1;` `A("3","computer")=5;` `A("3","printer")=2;` `A("3","disk")=6;` `A("3","memory")=3;` `A("3","laptop")=7;` `A("3","USB")=1;` `A().DocCorrelate(B(),"Cosine",.5);` `B.TreePrint();` `Yields` `1` ` 2=0.887096774193548` ` 3=0.741935483870968` `2` ` 1=0.887096774193548` ` 3=0.701612903225806` `3` ` 1=0.741935483870968` ` 2=0.701612903225806`
`mstring global::Extract( [INT [,INT] ] )` `mstring mstring::Extract( [INT [,INT] ] )`	Returns the substring of the invoking **global** or **mstring** beginning at the position designated by the 1st argument and ending at the position designated by the second argument, inclusive. If no second argument is given, the single character designated by the first argument is returned. If the second argument specifies a position beyond the end of the string, the remainder of the string including and following the character designated by the first argument is returned. `global g1("g1");` `g1("1")="abcdef";` `g1("1").Extact(2)      -> b` `g1("1").Extact(2,4)    -> bcd` `g1("1").Extract(2,99) -> bcdef`
`mstring mstring::Eval()`	Evaluates the Mumps expression in the invoking **mstring** object and returns the result in an **mstring**. If

	an error occurs, an *InterpreterException* is thrown. The invoking **mstring** object may contain a valid mumps expression. `mstring x="5*2";` `x.Eval()  -> 10` `global g("g");` `g("1","1")=22;` `x="^a(1,1)";` `x.Eval()  -> 22`
`int global::Find(STR [,INT] )` `int mstring::Find(STR [,INT] )`	Searches the invoking string for the firts instance of the **STR** argument and, if **STR** is found, returns the character position of the character immediately following the instance of **STR**. If an **INT** argument is provided, the search begins at that character offset in the invoking string. Returns -1 is **STR** is not found. `mstring p="abcdefabcdef";` `p.Find("def")   -> 7` `p.Find("def",5) -> 13`
`mstring Horolog()`	Returns an **mstring** of the form "x,y" where *x* is the number of days since December 31, 1840 and *y* is the number of seconds since midnight.
`void global::IDF(double DocCount)`	The IDF() function calculates for the invoking **global** array vector the i*nverse document frequency* weight of each term. The vector indices should be words and have as stored values the number of documents in which each word occurs. The document count for each element will be replaced by the calculated IDF value. The IDF is calculated as: l*og2(DocCount/*W_n*)+1* where W_n is the number of documents in which a term appears (the document frequency). The value *DocCount* is the total number of documents present in the collection. `global a("a");` `a("now")=2;` `a("is")=5;` `a("the")=6;` `a("time")=3;` `a().IDF(4);` `a().TreePrint();` `Yields:` `is=0.678072` `now=2.000000` `the=0.415037` `time=1.415037`
`mstring global::Justify(INT [,INT] )` `mstring mstring::Justify(INT [,INT] )`	Right justifies the invoking object in an **mstring** field whose length is given by the first argument. If the second argument is present and a positive integer, the invoking object is right justified in a field whose length is given by the first argument with the number decimal places as specified by the second argument. The two argument form imposes a numeric interpretation upon the first argument. Rounding occurs in the two argument case.

	`mstring p=123.456` `p.Justify(10)   ->    123.456` `p.Justify(10,2) ->     123.46` `p="abcdef";` `p.Justify(p,10) ->     abcdef`
`void global::Kill()`	Kill (delete) the named **global** array node and all descendants. To kill and entire **global** array use: `global gb("gb");` `gb().Kill;`
`int global::Length( [STR ] )` `int mstring::Length( [STR] )`	Returns the length of the invoking string. If an argument **STR** is given, the number returned is the number of invoking string segments divided by the argument. `mstring p="abc & def";` `p.Length()    -> 9` `p.Length("&") -> 2`
`double global::Max()`	Returns the maximum numeric value of the data bearing nodes beneath the given reference. Non-numeric values are treated as zeros. `global a("a");` `mstring i,j;` `for (i=1; i<11; i++)` `  for (j=1; j<11; j++)` `    a(i,j) = rand()%1000;` `a().Max()     -> 996 (results will vary)` `a("10").Max() -> 932`
`double global::Min()`	Returns the minimum numeric value of the data bearing nodes beneath the given reference. Non-numeric values are treated as zeros. `global a("a");` `mstring i,j;` `for (i=1; i<11; i++)` `  for (j=1; j<11; j++)` `    a(i,j) = rand()%1000;` `a().Min()     -> 11 (results will vary)` `a("10").Min() -->12`
`void global::Multiply(global, global)`	The invoking global array matrix is multiplied by the first argument global array matrix and the result is placed in the second argument global array matrix. The number of columns of the invoking global array matrix must equal the number of rows of the first argument global array matrix. The resulting matrix (second argument) will have *n* rows and *m* columns where *n* is the number of rows of invoking global array matrix and *m* is the number of columns of the first argument global array matrix. The contents of the second argument, if any, will be deleted before the operation begins. The data stored at each node in the invoking matrix and the first argument matrix must be numeric. All calculations are performed in double precision arithmetic. Each input matrix must be two dimensional. The output matrix is also two dimensional.

	`global d("d");` `global e("e");` `global f("f");` `d("1","1")=2; d("1","2")=3;` `d("2","1")=1; d("2","2")=-1;` `d("3","2")=0; d("3","2")=4;` `e("1","1")=5; e("1","2")=-2;` `e("1","3")=4; e("1","4")=7;` `e("2","1")=-6; e("2","2")=1;` `e("2","3")=-3; e("2","4")=0;` `d().Multiply(e(),f());` `f().TreePrint();` `Yields:` `1` ` 1=-8` ` 2=-1` ` 3=-1` ` 4=14` `2` ` 1=11` ` 2=-3` ` 3=7` ` 4=7` `3` ` 1=-24` ` 2=4` ` 3=-12` ` 4=0`
`mstring` **`global`**`::Name()`	Returns an **mstring** containing of the **global** reference with all variables and expressions in the indices evaluated. `global a("a");` `mstring b="1",c="2",d="3";` `a(b,c,d,c+d).Name()  -> a("1","2","3","5")`
`int global::Pattern(STR)` `int mstring::Pattern(STR)`	Evaluates the invoking string according to the pattern string **STR** (see Mumps documentation) and returns 0 (does not match) or 1 (does match). `mstring p=12345;` `p.Pattern("5N" -> 1`
`mstring global::Piece(STR, INT [,INT] )` `mstring mstring::Piece(STR, INT [,INT] )`	Returns a substring of the invoking object delimited by the instances of the first **STR** argument. The **STR** delimiter divides the invoking object into pieces. The substring returned in the two argument case is the *i*th substring of the invoking object there *i* is the value of the first **INT** argument. In the three argument form, the string returned begins at the *i*th piece and ends at the *j*th piece where *j* is the value of the second **INT** argument. If only one argument is given, *i* is assumed to be 1. `mstring p="abc.def.ghi";` `p.Piece(".")     -> abc` `p.Piece(".",2)   -> def` `p.Piece(".",2,3) -> def.ghi`

`int global::Qlength(mstring ref)` `int mstring::Qlength(char * ref)`	Returns the number of subscripts in the global array reference. **mstring** global array references must include the circumflex (^) character.[18] `global g("g");` `g(1,2,3,4,5).Qlength() -> 5` `mstring x="^g(1,2,3,4,5,6)";` `x.Qlength() -> 6`
`mstring mstring::Query()` `mstring global::Query()`	Returns an object of type **mstring** containing the next global array reference in the data base following the invoking global array reference or the empty string if there are none. The invoking object is either a global array reference or an **mstring** containing a string corresponding to a global array reference. **mstring** global array references must include the circumflex (^) character.[18] `mstring i,j;` `global g("g");` `for (i=1; i<10; i++)` `  for (j=1; j<10; j++)` `    g(i,j)=i+i;` `g().Query()    -> ^g("1","1")` `g(2).Query()   -> ^g("2","1")` `g(2,2).Query() -> ^g("2","3")` `i="^g()"` `i.Query() -> ^g("1","1")` `i=i.Query();` `i.Query() -> ^g("1","2")`
`mstring mstring::Qsubscript(int)` `mstring global::Qsubscript(int)`	Returns the subscript of a global array reference designated by the argument. **mstring** global array references must include the circumflex (^) character.[18] `global g("g");` `g(9,8,7).Qsubscript(3) -> 7` `mstring x="^g(9,8,7)";` `x.Qsubscript(3) -> 7`
`bool global::ReadLine()` `bool global::ReadLine(FILE *)` `bool global::ReadLine(istream &)` `bool mstring::ReadLine()` `bool mstring::ReadLine(FILE *)` `bool mstring::ReadLine(istream &)`	Reads the next input line into the invoking objsect. If no argument is given *stdin* is used. Otherwise, the inout file is determined by the argument.
`int sw(mstring s, mstring t, [int` `    show_aligns=0, int show_mat=0, int` `    gap=-1, int mismatch=-1, int match=2])` `int sw(string s, string t, [int` `    show_aligns=0, int show_mat=0, int` `    gap=-1, int mismatch=-1, int match=2])` `int sw(char *s, char *t, [int` `    show_aligns=0, int show_mat=0, int`	Calculate the *Smith-Waterman* Alignment between strings *s* and *t*. Result returned is the highest alignment score achieved. Parameters other than the first two are optional. If only some of the optional parameters are supplied, only trailing parameters may be omitted, as per C/C++ rules. If you compare very long strings (>100,000 character), you may exceed stack space. This can be increased under Linux with the command:

18 See example in Figure 44 on page 88.

gap=-1, int mismatch=-1, int match=2])	ulimit -s unlimited Other options are: `ulimit -a` and `ulimit -aH` to show limits. If *show_aligns* is zero, no printout of alternative alignments is produced (default). If *show_aligns* is not zero, a summary of the alternative alignments will be printed. If *show_mat* is zero, intermediate matrices will not be printed (default). The parameters *gap*, *mismatch* and *match* are the gap and mismatch penalties (normally negative integers) and the match reward (a positive integer). If insufficient memory is available, a *segmentation* violation will be raised. **The first character of each sequence string MUST be blank.** In the printed output, a colon represents a match, a hyphen represents a stretch of the associated string and a blank indicates mismatch. char s[]=" now is the time for all good men to come to the aid of the party"; char t[]=" time for good men"; int i=sw(s,t,1,0,-1,-1,3); cout << "Score: " << i << endl; Results in: 12 time- for all good-- men 32 :::: ::::: ::::: :::: 1 time for -- good men 22 score=48
int SQL_Command(mstring) int SQL_Command(string) int SQL_Command(char *)	Passes the string argument to the SQL database server. See Mumps *sql* command for a description of the argument. The results are written to a file named *mumps.tmp* where columns are <tab> separated.
int SQL_Connect(char *) int SQL_Connect(string) int SQL_Connect(mstring)	Establishes connection with the database server.
int SQL_Disconnect();	Disconnects from the database server.
mstring SQL_Message()	Returns most recent SQL database server returned message or the empty string if there is none.
bool SQL_Native()	Returns *true* if the **global** arrays are being stored in a native database.
bool SQL_Open()	Returns *true* if there is a connection to the database server, *false* otherwise.
mstring SQL_Table() mstring SQL_Table(mstring, [int]) mstring SQL_Table(string, [int]) mstring SQL_Table(char *, [int])	Returns an **mstring** containing name of the current **global** array table (default: *mumps*), followed by a comma, followed by the maximum number of columns permitted in the table (default is 10). If arguments are

	provided, they set the name of the table and the maximum number of columns in the table (maximum of 10). If the second argument is omitted, it defaults to 10.
`double global::Sum()`	The global array nodes beneath the invoking referenced global array are summed. Non-numeric quantities are treated as zero. `global a("a");` `mstring i, j;` `for (i = 1; i < 11; i++)` ` for (j = 1; j < 11; j++)` `  a(i, j) = 5;` `cout << a().Sum() << endl;   // -> 500` `cout << a("5").Sum() << endl; // -> 50`
`mstring SymGet(T1 name)`	Retrieves the value of the variable whose name is contained in *name* from the Mumps Interpreter symbol table. Throws *MumpsSymbolTableException* if the variable is not found. The data type **T1** may be **global**, **mstring** or **char***. See also: *SymPut().* `SymPut("k","100");` `cout << SymGet("k") << endl; // -> 100`
`bool SymPut(T1 name, T1 value)`	Insert into the Mumps Interpreter symbol table a variable whose name is contained in *name* with the value contained in *value*. The data type **T1** and **T2** may be any combination of **global**, **char*** or **mstring**. Returns *true* if successful, *false* otherwise. Variables in the Mumps Interpreter symbol table may be accessed by expressions passed to the function *mstring::Eval()* or *mstring::Assign()*. See also: *SymGet()*. `mstring i="3*k";` `SymPut("k","100");` `cout << i.Eval() << endl; // -> 300`
`void global::TermCorrelate(global B)`	*TermCorrelate()* builds a square *term-term* correlation matrix in **global** array B from the invoking **global** array document-term matrix. `global A("A");` `global B("B");` `int main() {` ` long i,j;` ` A("1","computer")=5;` ` A("1","data")=2;` ` A("1","program")=6;` ` A("1","disk")=3;` ` A("1","laptop")=7;` ` A("1","monitor")=1;` ` A("2","computer")=5;` ` A("2","printer")=2;` ` A("2","program")=6;` ` A("2","memory")=3;` ` A("2","laptop")=7;` ` A("2","language")=1;` ` A("3","computer")=5;` ` A("3","printer")=2;` ` A("3","disk")=6;`

```
 A("3","memory")=3;
 A("3","laptop")=7;
 A("3","USB")=1;

 A.TermCorrelate(B);

 mstring a;
 mstring b;

 a="";

 while (1) {
  a=B(a).Order();
  if (a=="") break;
  cout << a << endl;
  b="";
  while (1) {
   b=B(a,b).Order();
   if (b=="") break;
   cout <<"    " << b << "(" << B(a,b)
        << ")" << endl;
   }
  }
 return 0;
 }
```

Yields:

```
    USB
    computer(1)
    disk(1)
    laptop(1)
    memory(1)
    printer(1)
computer
    USB(1)
    data(1)
    disk(2)
    language(1)
    laptop(3)
    memory(2)
    monitor(1)
    printer(2)
    program(2)
data
    computer(1)
    disk(1)
    laptop(1)
    monitor(1)
    program(1)
disk
    USB(1)
    computer(2)
    data(1)
    laptop(2)
    memory(1)
    monitor(1)
    printer(1)
    program(1)
language
    computer(1)
    laptop(1)
```

<table>
<tr><td></td><td><pre>   memory(1)
    printer(1)
    program(1)
laptop
    USB(1)
    computer(3)
    data(1)
    disk(2)
    language(1)
    memory(2)
    monitor(1)
    printer(2)
    program(2)
memory
    USB(1)
    computer(2)
    disk(1)
    language(1)
    laptop(2)
    printer(2)
    program(1)
monitor
    computer(1)
    data(1)
    disk(1)
    laptop(1)
    program(1)
printer
    USB(1)
    computer(2)
    disk(1)
    language(1)
    laptop(2)
    memory(2)
    program(1)
program
    computer(2)
    data(1)
    disk(1)
    language(1)
    laptop(2)
    memory(1)
    monitor(1)
    printer(1)</pre></td></tr>
<tr><td><code>void global::Transpose(global)</code></td><td>The invoking two dimensional matrix global object is transposed and the result is placed in two dimensional global array object given as the argument. Any prior contents of the output array out are deleted before the operation commences.

<pre>global d("d");
global f("f");

d("1","1")=2;
d("1","2")=3;
d("2","1")=4;
d("2","2")=0;
d().Transpose(f());
f.TreePrint();

Results:</pre></td></tr>
</table>

	1 1=2 2=4 2 1=3 2=0
`void global::TreePrint([int, [ char ] ] )`	Prints the invoking global array as a tree. If a the first **int** argument is given, it is the number of spaces to indent each level (default is 1 if not specified). If the second argument is given, it is the character used to indent (default is blank character). See example in *global::Multiply()* above.
`bool ZSeek(FILE *file, mstring offset)` `bool ZSeek(FILE *file, global offset)` `bool ZTell(FILE *file)`	These functions are used in connection with direct access files opened with **FILE** pointers (see: *fopen()*). They are compatible with 64 bit file systems. *ZSeek()* positions the file designated by *file* to the offset specified in *offset*, a positive integer contained in a variable of type **mstring** or **global**. *ZTell()* places the current file offset in the file designated by *file* to the integer value in the **mstring** or **global** variable represented given by *offset*. Both functions return *true* if successful. Ordinarily, file offsets will be obtained by *ZTell()* and these will be stored in a data base. These values will be subsequently used by *ZSeek()* to reposition the file to the point it was at when the *ZTell()* was performed. After re-positioning, the next input or output operation on the file will occur at the point designated by *offset*. All offsets are positive integers relative to the start of the file.

Figure 42 Functions Defined on **mstring** and **global**

Some Function Examples	Results
`char gname[]="doc";` `global doc(gname);` `doc("1")="abcdef";` `mstring ppp = "abcdef";` `mstring aaa;`	
`cout << ppp.Ascii() << endl;` `cout << doc("1").Ascii() << endl;` `cout << ppp.Ascii(1) << endl;` `cout << doc("1").Ascii(1) << endl;`	97 97 97 97
`cout << ppp.Length() << endl;` `cout << doc("1").Length() << endl;`	6 6
`ppp="aaa & bbb";` `aaa="&";`	
`cout << ppp.Length("&") << endl;` `cout << ppp.Length("*") << endl;` `cout << ppp.Length(aaa) << endl;`	2 1 2
`doc("1")="&";`	

`cout << ppp.Length(doc("1")) << endl;`	2
`string strng="&";`	
`cout << ppp.Length(strng) << endl;`	2
`ppp = "123abc456abc";`	
`doc("1")="123abc456abc";`	
`doc("9")="abc";`	
`cout << ppp.Find("abc") << endl;`	7
`cout << doc("1").Find("abc") << endl;`	7
`cout << ppp.Find("abc",5) << endl;`	13
`cout << doc("1").Find("abc",5) << endl;`	13
`cout << doc("1").Find(doc("9"),5) << endl;`	13
`strng="abc";`	
`cout << ppp.Find(strng,5) << endl;`	13
`cout << Horolog() << endl;`	63815,68346
`doc("1").ReadLine();`	abcdef [input]
`cout << "readline global " <<doc("1") << endl;`	readline global abcdef
`ppp.ReadLine();`	abcdef [input]
`cout << "readline mstring " <<ppp << endl;`	readline mstring abcdef
`ppp="123";`	
`doc("1")=ppp;`	
`strng="3N";`	
`cout << ppp.Pattern("3N") << endl;`	1
`doc("9")="3N";`	
`cout << ppp.Pattern(doc("9")) << endl;`	1
`cout << doc("1").Pattern("3N") << endl;`	1
`doc("1")="3N";`	
`cout << ppp.Pattern(doc("1")) << endl;`	1
`cout << doc("1").Justify(10,2) << endl;`	3.00
`cout << doc("1").Justify(10) << endl;`	3N
`cout << ppp.Justify(10,2) << endl;`	123.00
`cout << ppp.Justify(10) << endl;`	123
`cout << doc("1").Data() << endl;`	1
`doc("2","3")=123;`	11
`cout << doc("2").Data() << endl;`	
`ppp="abcdef";`	
`mstring off="2";`	
`cout << ppp.Extract(2,3) << endl;`	bc
`cout << ppp.Extract(off,off+1) << endl;`	bc
`cout << ppp.Extract(2) << endl;`	b
`cout << ppp.Extract() << endl;`	a
`doc("1")=ppp;`	
`cout << doc("1").Extract(2,3) << endl;`	bc
`cout << doc("1").Extract(2) << endl;`	b
`cout << doc("1").Extract() << endl;`	a
`ppp=-123.45678;`	
`cout << ppp.Fnumber("P","2") << endl;`	(123.46)

cout << ppp.Fnumber("P") << endl;	(123.457)
doc("1")=-123.45678;	
cout << doc("1").Fnumber("P","2") << endl;	(123.46)
cout << doc("1").Fnumber("P") << endl;	(123.45678)
ppp="abc.def.ghi";	
cout << ppp.Piece(".",2) << endl;	def
cout << ppp.Piece(".",2,3) << endl;	def.ghi
strng=".";	
cout << ppp.Piece(strng,2,3) << endl;	def.ghi
doc("9")=strng;	
cout << ppp.Piece(doc("9"),2,3) << endl;	def.ghi
doc("1")=".";	
cout << ppp.Piece(doc("1"),2) << endl;	def
cout << ppp.Piece(doc("1"),2,3) << endl;	def.ghi
long d=1;	
float e=1.0;	
int f=1;	
doc("9")="abcdef";	
cout << doc("9").Ascii(e) << endl;	97
cout << doc("9").Ascii(f) << endl;	97
cout << doc("9").Ascii(d+1) << endl;	98
cout << doc("9").Ascii(e+1) << endl;	98
cout << doc("9").Ascii(f+1) << endl;	98
off=1;	
cout << doc("9").Ascii(off+d) << endl;	98
cout << doc("9").Ascii(off+e) << endl;	98
cout << doc("9").Ascii(off+f) << endl;	98
mstring g=1;	
cout << doc("9").Ascii(off+g) << endl;	98
cout << doc("9").Ascii(off+g) << endl;	98
cout << doc("9").Ascii(off+g) << endl;	98

Figure 43 Function Examples

Assume that the following entries have been made into the global array data base:

```
 set ^mesh("A01")="Body Regions"
 set ^mesh("A01","047")="Abdomen"
 set ^mesh("A01","047","025")="Abdominal Cavity"
 set ^mesh("A01","047","025","600")="Peritoneum"
 set ^mesh("A01","047","025","600","225")="Douglas' Pouch"
 set ^mesh("A01","047","025","600","451")="Mesentery"
 set ^mesh("A01","047","025","600","451","535")="Mesocolon"
 set ^mesh("A01","047","025","600","573")="Omentum"
 set ^mesh("A01","047","025","600","678")="Peritoneal Cavity"
 set ^mesh("A01","047","025","750")="Retroperitoneal Space"
 set ^mesh("A01","047","050")="Abdominal Wall"
 set ^mesh("A01","047","365")="Groin"
 set ^mesh("A01","047","412")="Inguinal Canal"
 set ^mesh("A01","047","849")="Umbilicus"
 set ^mesh("A01","176")="Back"
 set ^mesh("A01","176","519")="Lumbosacral Region"
```

```
 set ^mesh("A01","176","780")="Sacrococcygeal Region"
 set ^mesh("A01","236")="Breast"
 set ^mesh("A01","236","500")="Nipples"
 set ^mesh("A01","378")="Extremities"
 set ^mesh("A01","378","100")="Amputation Stumps"
 set ^mesh("A01","378","610")="Lower Extremity"
 set ^mesh("A01","378","610","100")="Buttocks"
 set ^mesh("A01","378","610","250")="Foot"
 set ^mesh("A01","378","610","250","149")="Ankle"
 set ^mesh("A01","378","610","250","300")="Forefoot, Human"
 set ^mesh("A01","378","610","250","300","480")="Metatarsus"
 set ^mesh("A01","378","610","250","300","792")="Toes"
 set ^mesh("A01","378","610","250","300","792","380")="Hallux"
 set ^mesh("A01","378","610","250","510")="Heel"
 set ^mesh("A01","378","610","400")="Hip"
 set ^mesh("A01","378","610","450")="Knee"
 set ^mesh("A01","378","610","500")="Leg"
 set ^mesh("A01","378","610","750")="Thigh"
 set ^mesh("A01","378","800")="Upper Extremity"
 set ^mesh("A01","378","800","075")="Arm"
 set ^mesh("A01","378","800","090")="Axilla"
 set ^mesh("A01","378","800","420")="Elbow"
 set ^mesh("A01","378","800","585")="Forearm"
 set ^mesh("A01","378","800","667")="Hand"
 set ^mesh("A01","378","800","667","430")="Fingers"
 set ^mesh("A01","378","800","667","430","705")="Thumb"
 set ^mesh("A01","378","800","667","715")="Wrist"
 set ^mesh("A01","378","800","750")="Shoulder"
```

```
global mesh("mesh");
mstring x;
int i,j;

x = "^mesh()"; // initial global array reference - beginning of array
x = x.Query(); // find first real reference

while (1) {
 if (x == "") break; // nothing to print
 i = x.Qlength(); // how many subscripts
 for (j=0; j<i; j++) cout << " "; // indent by number of subscripts
 cout << x.Qsubscript(i) << " " << x.Eval() << endl; // show index & value
 x = x.Query(); // get next
 }
```

The above code yields:

```
  047 Abdomen
   025 Abdominal Cavity
    600 Peritoneum
     225 Douglas' Pouch
     451 Mesentery
      535 Mesocolon
     573 Omentum
     678 Peritoneal Cavity
    750 Retroperitoneal Space
   050 Abdominal Wall
   365 Groin
   412 Inguinal Canal
   849 Umbilicus
  176 Back
   519 Lumbosacral Region
   780 Sacrococcygeal Region
  236 Breast
```

```
   500 Nipples
  378 Extremities
   100 Amputation Stumps
   610 Lower Extremity
    100 Buttocks
    250 Foot
     149 Ankle
     300 Forefoot, Human
      480 Metatarsus
      792 Toes
       380 Hallux
     510 Heel
    400 Hip
    450 Knee
    500 Leg
    750 Thigh
   800 Upper Extremity
    075 Arm
    090 Axilla
    420 Elbow
    585 Forearm
    667 Hand
     430 Fingers
      705 Thumb
     715 Wrist
    750 Shoulder
```

Figure 44 Query(), Qsubscript() and Qlength() Example

11.7 Examples

```
#include <fstream>
#include <mumpsc/libmpscpp.h>

global doc("doc");
global idf("idf");
global indx("index");

int main() {

 FILE *u1;

 ofstream u2 ("document-term-matrix-
             weighted.txt", ios::out);
 assert (u2 != 0);

 mstring d,tt,w,null;
 double x,idfmin=6.0;

 null="";

 indx().Kill();

 for (d=doc(null).Order(); d != null; d =
      doc(d).Order() ) {
  u2 << "doc=" << d << "   ";

  for (w = doc(d,null).Order(); w != null;
       w = doc(d,w).Order()) {
   if (idf(w) < idfmin) {
    doc(d,w).Kill();
    }
```

```
#!/usr/bin/mumps
# weight.mps December 26, 2011

 open 2:"document-term-matrix-
        weighted.txt,new"

 idfmin=6.0;

 kill ^index

 for d=$order(^doc(d)) do

 . use 2 write !,"doc=",d,?15

 . for w=$order(^doc(d,w)) do

 .. if ^idf<w<idfmin kill ^doc(d,w)
```

```
   else {
    x = idf(w)*doc(d,w);
    doc(d,w)=x;
    indx(w,d)=x;
    u2 << w << "(" << x << ") ";
    }
   }
  u2 << endl << endl;
  }

 u2.close();

 ofstream u3
  ("term-document-matrix-weighted.txt",
    ios::out);
 assert (u3 != 0);

 for (w=indx(null).Order(); w != null;
      w=indx(w).Order()) {
  u3 << w << "   ";
  for (d=indx(w,null).Order(); d != null;
       d=indx(w,d).Order()) {
   u3 << d << "(" << indx(w,d) << ") ";
   }
  u3 << endl << endl;
  }

 u3.close();

 return 0;

 }
```

```
.. else do
 ... set x=^idf(w)*^doc(d,w)
 ... set ^doc(d,w)=x
 ... set ^index(w,d)=x
 ... write w,"(",x,") "
 . write !




close 2

 open 2:"term-document-matrix-
        weighted.txt,new"
 use 2


for w=$order(^index(w)) do

 . write w,?26
 . for d=$order(^index(w,d)) do

 .. write d,"(",^index(w,d),") "

 . write !


 close 2
```

Figure 45 Document Weighting

12 Licenses

12.1 GNU Licenses

12.1.1 GNU General Public License

```
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

      Preamble

 The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  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
this service 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 make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

 For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  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.

 We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

 Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

 Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.
```

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

<>

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

<>

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
<>
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
<>

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.

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 convey 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) 19yy <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 2 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, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

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

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

Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision 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, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This 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 Library General
Public License instead of this License.

12.1.2 GNU Free Documentation License

GNU Free Documentation License
Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other
written document "free" in the sense of freedom: to assure everyone
the effective freedom to copy and redistribute it, with or without
modifying it, either commercially or noncommercially. Secondarily,
this License preserves for the author and publisher a way to get
credit for their work, while not being considered responsible for
modifications made by others.

This License is a kind of "copyleft", which means that derivative
works of the document must themselves be free in the same sense. It
complements the GNU General Public License, which is a copyleft
license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
To use this License in a document you have written, include a copy of
the License in the document and put the following copyright and
license notices just after the title page:

      Copyright (c)  YEAR  YOUR NAME.
      Permission is granted to copy, distribute and/or modify this document
      under the terms of the GNU Free Documentation License, Version 1.1
      or any later version published by the Free Software Foundation;
      with the Invariant Sections being LIST THEIR TITLES, with the
      Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
      A copy of the license is included in the section entitled "GNU
      Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections"
instead of saying which ones are invariant.  If you have no
Front-Cover Texts, write "no Front-Cover Texts" instead of
"Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we
recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.

--------------------------------------------------------------------------------
```

12.1.3 GNU LESSER GENERAL PUBLIC LICENSE

```
GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

 Copyright (C) 1991, 1999 Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]

Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

  This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it.  You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

  When we speak of free software, we are referring to freedom of use,
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 this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

  To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
```

rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining

```
    where to find the accompanying uncombined form of the same work.

  8. You may not copy, modify, sublicense, link with, or distribute
the Library except as expressly provided under this License.  Any
attempt otherwise to copy, modify, sublicense, link with, or
distribute the Library is void, and will automatically terminate your
rights under this License.  However, parties who have received copies,
or rights, from you under this License will not have their licenses
terminated so long as such parties remain in full compliance.

  9. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Library or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Library (or any work based on the
Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

  10. Each time you redistribute the Library (or any work based on the
Library), the recipient automatically receives a license from the
original licensor to copy, distribute, link with or modify the Library
subject to these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties with
this License.

  11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
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
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all.  For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply,
and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Library under this License may add
an explicit geographical distribution limitation excluding those countries,
```

so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser 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 Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "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 LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY 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 LIBRARY (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 LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

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

```
    Copyright (C)

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library 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
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

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

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library `Frob' (a library for tweaking knobs) written by James Random Hacker.

  , 1 April 1990
  Ty Coon, President of Vice
```

That's all there is to it!

12.2 Perl Compatible Regular Expression Library License

Programs written with the MDH may call upon the Perl Compatible Regular Expression Library. In some cases, this library is distributed with the Mumps Compiler. The PCRE Library is not covered by the GNU GPL/LGPL Licenses but, rather, by the license shownn below. The following is the PCRE license:

```
PCRE LICENCE
------------
PCRE is a library of functions to support regular expressions whose syntax
and semantics are as close as possible to those of the Perl 5 language.
Written by: Philip Hazel
University of Cambridge Computing Service,
Cambridge, England. Phone: +44 1223 334714.
Copyright (c) 1997-2001 University of Cambridge
Permission is granted to anyone to use this software for any purpose on any
computer system, and to redistribute it freely, subject to the following
restrictions:
1. This software 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.
2. The origin of this software must not be misrepresented, either by
   explicit claim or by omission. In practice, this means that if you use
   PCRE in software which you distribute to others, commercially or
   otherwise, you must put a sentence like this
     Regular expression support is provided by the PCRE library package,
     which is open source software, written by Philip Hazel, and copyright
     by the University of Cambridge, England.
   somewhere reasonably visible in your documentation and in any relevant
   files or online help data or similar. A reference to the ftp site for
```

```
   the source, that is, to
     ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/
   should also be given in the documentation.
3. Altered versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
4. If PCRE is embedded in any software that is released under the GNU
   General Purpose Licence (GPL), or Lesser General Purpose Licence (LGPL),
   then the terms of that licence shall supersede any condition above with
   which it is incompatible.
The documentation for PCRE, supplied in the "doc" directory, is distributed
under the same terms as the software itself.
End
```

Alphabetical Index