

Mumps Programming Language Interpreter, Compiler, and C++ Class Library User's Guide

Kevin C. O'Kane

kc.okane@gmail.com
okane@uni.edu

<http://threadsafefbooks.com/>
<http://www.cs.uni.edu/~okane/>

December 20, 2025

Table of Contents

1 Installation	6
1.2 INTERPRETER VS COMPILER.....	7
1.3 MUMPS GLOBAL ARRAY DATABASE.....	7
1.4 REQUIRED SYSTEM SOFTWARE.....	9
1.5 BASIC SOFTWARE INSTALLATION.....	10
1.6 INSTALLING SQLITE3 DATABASE OPTION.....	10
1.7 COMPILATION OPTIONS.....	12
1.8 FULL LIST OF CONFIGURE OPTIONS.....	14
2 Running a Mumps Program	17
2.1 GLOBAL ARRAY DATABASER FILE(S).....	17
2.2 RUNNING THE MUMPS CLI INTERPRETER.....	17
2.3 INTERPRETING A MUMPS PROGRAM.....	17
2.4 COMPILING A PROGRAM.....	18
3 Relational Database Functions	19
3.1 \$ZSQLITE.....	19
3.2 \$ZSQLITE("BEGIN TRANSACTION").....	19
3.3 \$ZSQLITE("COMMIT TRANSACTION").....	19
3.4 \$ZSQLITE("SAVEPOINT"[,SAVEPOINT_NAME]).....	19
3.5 \$ZSQLITE("ROLLBACK"[,SAVEPOINT]).....	19
3.6 \$ZSQLITE("SQL",SQL_COMMAND).....	19
3.7 \$ZSQLITE("PRAGMA",OPTION).....	19
3.8 \$ZSQLOPEN.....	19
3.9 \$ZNATIVE.....	19
4 Implementation Notes	20
4.1 SOURCE CODE FORMAT.....	20
4.2 MODULO OPERATOR.....	20
4.3 GOTO COMMAND.....	20
4.4 NOTES ON ARITHMETIC PRECISION.....	20
4.5 NEW COMMAND.....	21
4.6 KILL COMMAND.....	24
4.7 FOR COMMAND EXTENSIONS.....	24
4.8 BREAK AND QUIT.....	25
4.9 LOCK COMMAND WITH SQL.....	27
4.10 LOCK COMMAND IN NATIVE DATABASE MODE.....	28

4.11 NAKED INDICATOR.....	28
4.12 JOB COMMAND.....	28
4.13 FILE NAMES CONTAINING DIRECTORY INFORMATION.....	28
4.14 FILE NAMES.....	28
4.15 ARRAY INDEX COLLATING SEQUENCE.....	28
4.16 SUBROUTINE & FUNCTION CALLS.....	29
4.17 \$FNUMBER() FUNCTION.....	30
4.18 \$SELECT() FUNCTION.....	30
4.19 COMPILING LARGE PROGRAMS.....	30
4.20 EMBEDDED EXPRESSIONS.....	31
4.21 INLINE C++ CODE (COMPILER ONLY).....	31
4.22 FUNCTIONS.....	31
5 Shell Commands	34
5.1 SHELL/P.....	34
5.2 SHELL/G.....	34
5.3 SHELL.....	34
6 Added Commands	35
6.1 DATABASE EXPR.....	35
6.2 ZHALT RETURN_CODE.....	35
6.3 DECLARE.....	35
7 Z Functions and System Variables	36
7.1 SYSTEM VARIABLES.....	36
7.2 BASH FUNCTIONS.....	36
7.3 MATH FUNCTIONS.....	37
7.4 DATE FUNCTIONS.....	38
7.5 SPECIAL PURPOSE FUNCTIONS.....	39
7.6 VECTOR AND MATRIX FUNCTIONS.....	43
7.7 TEXT PROCESSING FUNCTIONS.....	45
7.8 SQL FUNCTIONS.....	54
8 Pattern Matching	56
8.1 MUMPS 95 PATTERN MATCHING.....	56
8.2 USING PERL REGULAR EXPRESSIONS.....	56
9 Mumps Compiler	58
9.1 HOW TO COMPILE AND RUN A MUMPS.....	58
9.2 COMPILER ERROR MESSAGES.....	58
9.3 GLOBAL ARRAY STORAGE IN COMPILED PROGRAMS.....	59
9.4 COMPILER PERFORMANCE AND INTEROPERABILITY.....	59
10 The Multi-Dimensional and Hierarchical Database Toolkit	60
10.1 INTRODUCTION.....	60
10.2 INSTALLATION.....	60
10.3 COMPILING PROGRAMS.....	60
10.4 WRITING C++ MDH PROGRAMS.....	61
10.5 MDH IMPLEMENTATION OF GLOBALS.....	61
10.6 GLOBAL ARRAYS AS TREES AND MATRICES.....	62
10.7 ACCESSING GLOBAL ARRAYS.....	65
10.8 GLOBAL ARRAY INDICES.....	65
10.9 NAVIGATING GLOBALS.....	66
10.10 LOCKING THE DATA BASE.....	69
10.11 CLASS MSTRING.....	70
10.12 MSTRING OPERATIONS.....	70

10.13 MSTRING FUNCTIONS AND METHODS.....	73
10.14 BASIC MSTRING EXAMPLE.....	80
10.15 DETAILED MSTRING EXAMPLES.....	80
10.16 CLASS GLOBAL.....	85
10.17 DIRECT BTREE ACCESS.....	113
10.18 INVOKING THE MUMPS INTERPRETER.....	115
10.19 MISCELLANEOUS FUNCTIONS.....	115
10.20 BOYER-MOORE-GOSPER FUNCTIONS.....	116
10.21 CVT().....	118
10.22 XECUTE() AND COMMAND().....	118
10.23 ERRORMESSAGE().....	118
10.24 ERROR EXCEPTIONS.....	118
10.25 HITRATIO().....	119
10.26 HASHING FUNCTIONS.....	120
10.27 DUMP GLOBAL ARRAY DATABASE.....	120
10.28 STREAM OUTPUT.....	120
10.29 SMITH-WATERMAN ALIGNMENT FUNCTION.....	120
10.30 STOP LIST FUNCTIONS: STOPINIT(), STOPLOOKUP().....	122
10.31 SYNONYM FUNCTIONS: SYMINIT(), SYN().....	122
10.32 INT \$TEST.....	122
10.33 XECUTE().....	122
10.34 ZSEEK() ZTELL().....	123
10.35 MDH FUNCTIONS.....	123
11 GTK Desktop GUI Apps	125
11.1 GLADE GUI DESIGN TOOL.....	125
11.2 GTK EXAMPLE.....	126
12 Licenses	131
12.1 GNU LICENSES.....	131
12.2 PERL COMPATIBLE REGULAR EXPRESSION LIBRARY LICENSE.....	150

Index of Figures

Figure 1 Directory Map of Mumps Distro.....	7
Figure 2 Tree Structured Medical Record.....	8
Figure 3 new Command without Arguments.....	23
Figure 4 new Command with Comma List.....	24
Figure 5 new Command with Parenthesized List.....	24
Figure 6 Subroutine/Function Calls.....	30
Figure 7 Local Functions.....	32
Figure 8 Call by Value Functions.....	32
Figure 9 Call by Reference Functions.....	33
Figure 10 Function Return Values.....	33
Figure 11 Shell Command Example.....	34
Figure 12 \$Zb() Examples.....	39
Figure 13 \$Zseek() Examples.....	41
Figure 14 \$Zwi() Examples.....	42
Figure 15 Scan Functions Examples.....	43
Figure 16 \$zzAvg() Example.....	44
Figure 17 \$zzCentroid() Example.....	44
Figure 18 \$zzCount() Example.....	44
Figure 19 \$zzMax() Example.....	45
Figure 20 \$zzMin() Example.....	45

Figure 21 Similarity Formulae.....	46
Figure 22 Similarity Functions.....	47
Figure 23 \$zzBMGSearch() Example.....	47
Figure 24 \$zzMultiply() Example.....	48
Figure 25 \$zzSum() Example.....	48
Figure 26 \$zzTranspose() Example.....	48
Figure 27 \$zPerlMatch() Example.....	49
Figure 28 \$zReplace() Example.....	49
Figure 29 \$zShred() Example.....	50
Figure 30 \$ShredQuery() Example.....	50
Figure 31 \$zSmithWaterman() Example.....	51
Figure 32 \$zzIDF() Example.....	52
Figure 33 \$zTermCorrelate() Example.....	53
Figure 34 \$zDocCorrelate()Example.....	54
Figure 35 Stop List Functions.....	55
Figure 36 Example C++ Code.....	59
Figure 37 Global Array as a Matrix.....	62
Figure 38 Global Array as Matrix with Numeric Subscripts.....	63
Figure 39 Global Array as Matrix with Additional Nodes.....	63
Figure 40 Global Array as Sparse Matrix.....	64
Figure 41 Tabular View of Tree.....	64
Figure 42 Global Array Tree.....	65
Figure 43 Navigating Global Arrays - Data().....	66
Figure 44 Navigating Global Arrays - Order().....	67
Figure 45 Global Array Navigation Example.....	67
Figure 46 Hierarchical Global Array Example.....	69
Figure 47 mstring Operator Overloads.....	73
Figure 48 Mumps Pattern Codes.....	76
Figure 49 Shred Function.....	78
Figure 50 ShredQuery.....	79
Figure 51 mstring Examples.....	80
Figure 52 mstring Assignment Examples.....	81
Figure 53 mstring Arithmetic Operations.....	83
Figure 54 Exceptions.....	85
Figure 55 Example Global Array Arithmetic.....	87
Figure 56 Global Array Operator Overloads.....	90
Figure 57 Accessing Global Array Data Example.....	91
Figure 58 TreePrint.....	92
Figure 59 TreePrint Output.....	93
Figure 60 Count Example.....	94
Figure 61 Max Example.....	95
Figure 62 Multiply Example.....	97
Figure 63 Name Example.....	97
Figure 64 Order Example.....	98
Figure 65 Avg Example.....	99
Figure 66 MeSH Headings.....	101
Figure 67 Query Functions Example.....	103
Figure 68 Sim1 Example.....	104
Figure 69 Jaccard Example.....	104
Figure 70 Dice Example.....	105
Figure 71 Cosine Example.....	106
Figure 72 Transpose Example.....	106
Figure 73 Centroid Example.....	107

Figure 74 TermCorrelate Example.....	109
Figure 75 DocCorrelate Example.....	111
Figure 76 IDF Example.....	111
Figure 77 Sum Example.....	112
Figure 78 BTREE Example.....	114
Figure 79 Boyer-Moore Example.....	116
Figure 80 Exceptions Examples.....	119
Figure 81 Smith-Waterman Example.....	121
Figure 82 Glade Canvas.....	125
Figure 83 Toggle Button Screen 1.....	126
Figure 84 Toggle Button Screen 2.....	127
Figure 85 Toggle Button Screen 3.....	127
Figure 86 Toggle Button Screen 4.....	128

1 Installation

1.1.1 Distro File Structure

The Figure 1 gives the directory tree structure of the distribution. The main directory is named *mumps* and contains two sub-directories:

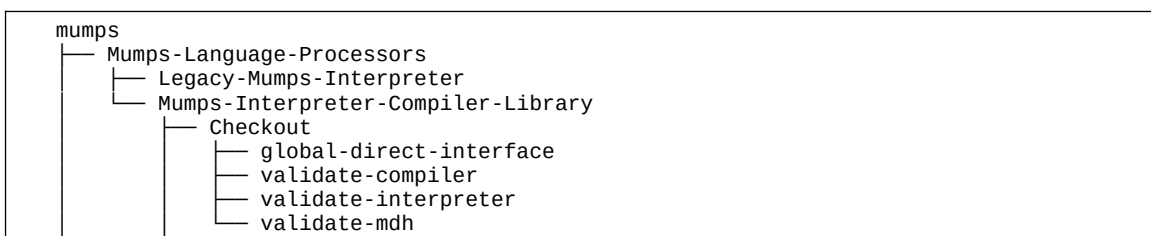
1. *Mumps-Language-Processors*
Contains the Mumps interpreter source code and installers.
2. *Mumps-Projects*
Contains programming examples including:
 - A genetic sequence search tool;
 - A compiler to generate GTK3 based GUIs with Mumps and Glade;
 - A collectio of information storage & retrieval modules and databases
 - An optimum binary tree example.

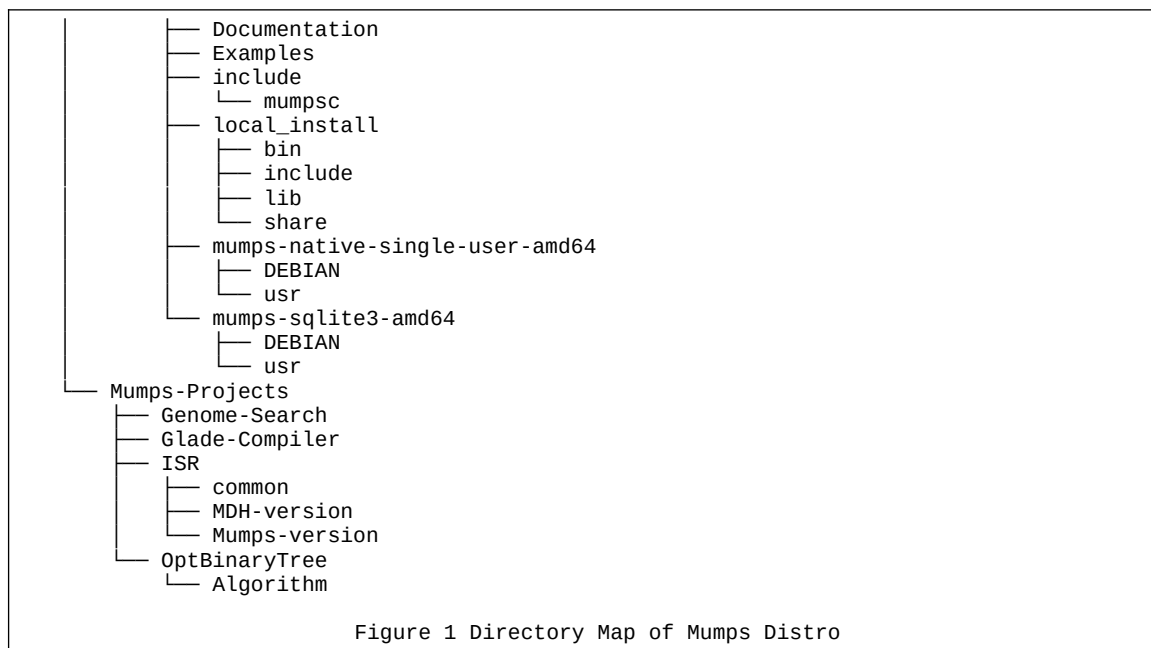
The directory *Mumps-Language-Processors* containss two parts:

1. *Legacy-Mumps-Interpreter*
Contains the original, unmaintained, Mumps iterpreter.
2. *Mumps-Interpreter-Compiler-Library*
Contains the current Mumps interpreter, compiler, MDH Toolkit and installers.

The *Mumps-Interpreter-Compiler-Library* contains directories:

1. *Checkout*
Contains code to check the interpreter, compiler, and toolkit.
2. *Documentation*
Contains several docymentaion files including this one.
3. *Examples*
Some short Mumps examples.
4. *include*
The include files for the source code.
5. *local_install*
Contains a local version of the installed code used for building the installers.
6. *mumps-native-single-user-amd64*
Contains files needed to build a Debian isntaller for the native single user file system.
7. *mumps-sqlite3-amd64*
Contains files needed to build a Debian installer for the Sqlite3 version of the database.
8. The directory also contains source code, object files, installers. Makefiles, configure and Bash scripts.





1.2 Interpreter vs Compiler

The interpreter executes code directly from source text parsing each line as it is executed. The Mumps indirection arguments permit Mumps source code to be dynamically created and executed.

On the other hand, the compiler generates a C++ file which is subsequently compiled and linked with Mumps run-time libraries to produce binary executables. The compiler supports most of the features supported by the interpreter but not all. Compiler code is usually faster but, since both use the same database engines, database intensive programs run at about the same speed whether compiled or interpreted.

When the compiler translates Mumps to C++, the resulting C++ programs can be edited and other additional code introduced which might not be available in original Mumps.

1.3 Mumps Global Array Database

Globals arrays are unique to Mumps. They are a disk resident tree structured database whose elements and structure appear in programs as indexed arrays. The indices may be numeric or text and the arrays are not declared. In an array, each successive index describes part of the path from root to a leaf node. Data may be stored at any node.

Mumps was developed in the late 1960s for use on small computers. It was originally an interpreted language similar to BASIC which was also developed in the 1960s.

Unlike BASIC, Mumps was designed to be a medical database language. Its only basic data type was string although strings containing numbers could be used with arithmetic operators.

The main challenge Mumps was designed to satisfy was the storage of hierarchical medical records.

The medical record structure, as envisioned by Mumps was a tree where contents were organized hierarchically as shown in Figure 2.

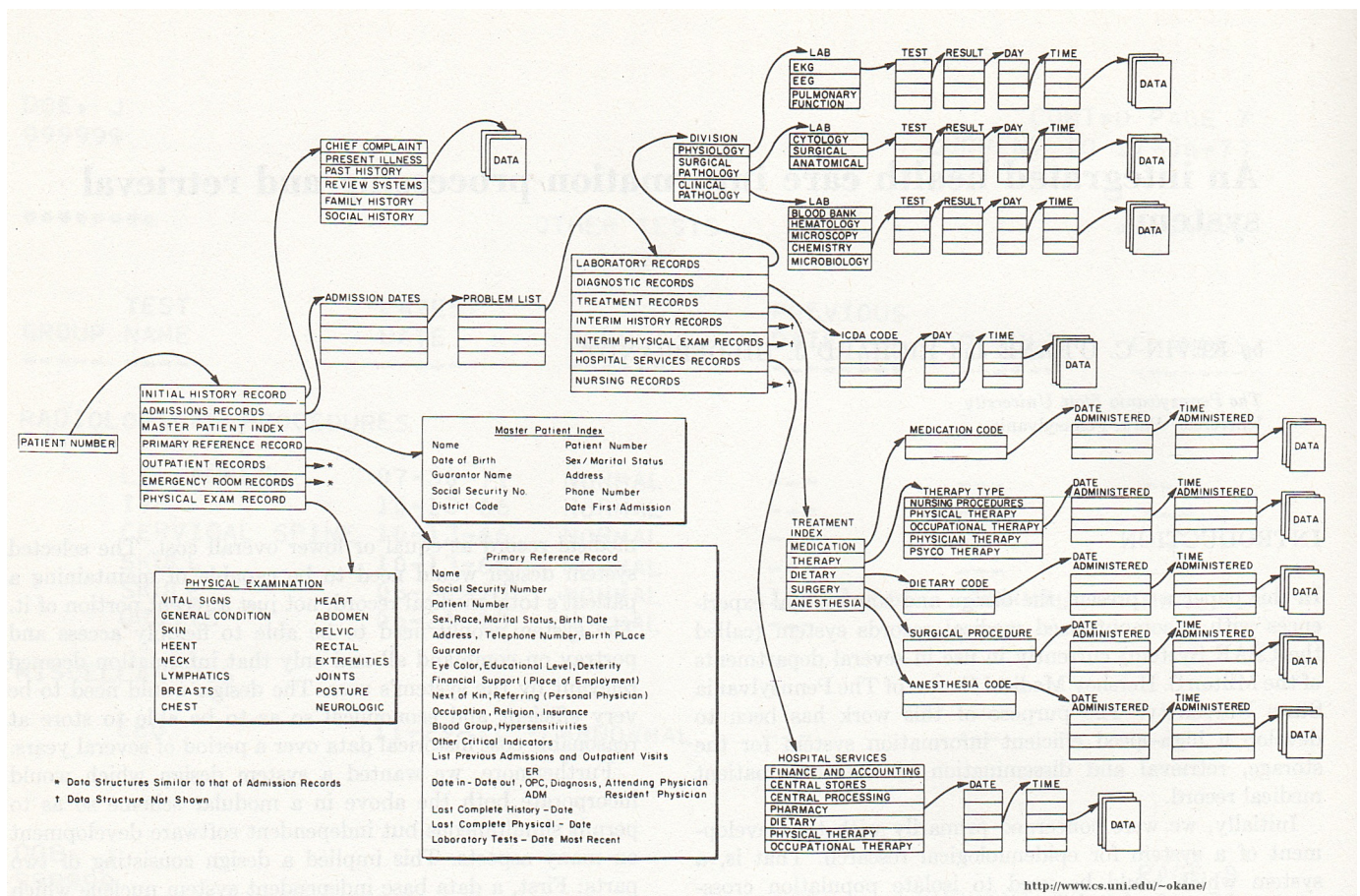


Figure 2 Tree Structured Medical Record

Trees were implemented in Mumps in structures known as **global arrays**. Global arrays are sparse disk resident trees represented in the language as string indexed arrays. Array notation follows the convention of FORTRAN and consists of an array name followed by a parenthesized list of indices. Indices trace a path through the global array tree for the named global array from the root to the final leaf. The height of the tree is variable depending on the path.

Global arrays are distinguished from ordinary volatile memory arrays by being preceded by a circumflex (^) character. While memory resident arrays disappear when a program ends, global arrays persist and are accessible to other programs in the system. A typical global array reference might appear as:

```
^patient(ptid, "hx", "PRR")
```

Data may be stored at any global array node either intermediate or terminal.

In this implementation of Mumps, global arrays may be stored either in an Sqlite3 relational database or in a native, single user B-tree based file system.

1.3.1 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 is a *single user* B-tree based 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* 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. Global arrays are stored in two files: *key.dat* and *data.dat*.

1.3.2 Sqlite3 Database Option

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

While this option is slower than the native database option, due to relational database 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 script will build various versions of the system. These are detailed next.

The scripts assume a Debian (*apt*) 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 according to the procedures for your system.

Some of the scripts provided with the distribution will 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*.

1.4 Required System Software

Building mumps requires that your system have certain software installed. These packages are available through the Synaptic Package Manager or *apt-get*. The build script, and the installers provided, automatically install these if they are not present on your system.

The required software included the following:

1. Linux Debian based version such as Debian, Ubuntu or Mint. The Windows WSL (Windows Subsystem for Linux) implementation with Ubuntu may also 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>

1.5 Basic Software Installation

The instructions to install Mumps are in the README.txt file in the directory:

mumps/Mumps-Language-Processors/Mumps-Interpreter-Compiler-Library

This file describes how to install and compile the system from source code or installer.

1.6 Installing Sqlite3 Database Option

Either the native B-tree based or Sqlite3 database option may be selected. The native database requires no special preparation while the Sqlite3 database needs to be initialized. Files created by one version of the database are not interchangeable with the other directly.

1.6.1 Sqlite3 Database Configuration

In order for Mumps to store and retrieve global arrays in Sqlite3, there must be a pre-existing database file named **mumps.sqlite** which is accessible to the instance of Mumps being executed. Normally, that means the file is in the directory where you started Mumps from. Links, however, may be used if the database file is located in another directory.

You may create or re-initialize **mumps.sqlite** with the script file:

mumps-sql-db-create

which is installed by the installers. The file *mumps.sqlite* contains the Sqlite3 database and must be present prior to starting Mumps if you are using the Sqlite3 database option. If not, Mumps will halt.

Note: the installer should remove the older version of this code (named mumps-sql-db-create.script) but if not, type:

```
sudo rm /usr/bin/mumps-sql-db-create.script
```

Options contained in install scripts can be used to set the maximum number of indices, the maximum number of characters per Mumps global array index, and the maximum number of characters that can be stored at a node.

1.6.2 Sqlite3 Database Global Array Implementation

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

On the other hand, the Sqlite3 relational database provides flexible multi-user, robust, is fully ACID (*Atomicity, Consistency, Isolation, Durability*) compliant and provides 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). 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**), followed by the value stored (if any) (**a10**).

For example, the Mumps code:

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

would map to the Sqlite3 database table named *mumps* as follows:

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*.

In most modern database systems, the *null* columns do not seriously impact speed. Alternative methods to implement trees in relational databases introduce very high levels of overhead. Further, the **mumps** table is indexed by Sqlite3 by global array name and the first index (first two columns) for faster access.

If your program instantiates array elements as shown in the following Mumps code:

```
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 like¹:

¹ Table row order may differ but this is not important.

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							flightless
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 index duplication seen in the above is also present in the native B-tree implementation.

An advantage, as mentioned above, is that data stored in such a table may be queried using the *sqlite3* CLI app *with* an ordinary SQL command such as:

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

which yields *flying*.

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

1.7 Compilation Options

The *build* script contains an invocation of the *configure* script with the basic options pre-set. Other additional options are described below. Do not run configure directly. Use the provided script.

1.7.1 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.

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.

1.7.1.1 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=yes

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.

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.

1.7.1.2 Extended Precision Math

Extended precision is available through use of the GNU multiple precision arithmetic library² and the GNU MPFR library³. 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 specified as a configuration option:

--with-hardware-math=no

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

² <http://www.mpfr.org/>

³ <http://gmplib.org/manual/index.html>

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.

1.8 Full List of Configure Options

The configure step as is typical contains many options. Specifying these causes modification to the source code and changes the final product. These must be set correctly. Not all combinations work.

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:

1.8.1 **configure prefix=**

The directory where the runtime modules will be stored. If this is not specified, the default location is in a directory named ***local_install*** in the mumps distro directory.

1.8.2 General Relational Database Options

1.8.2.1 **--with-dbname=name**

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

1.8.2.2 **--with-index_size=number**

Maximum number of characters in each Sqlite3 global array index.

1.8.2.3 **--with-data_size=nbr**

Maximum number of data characters stored for an Sqlite3 global array reference (final column).

1.8.2.4 **--with-index-max=nbr**

Maximum number of database columns. Default is 7. Maximum is 32. A value of 7 permits a global array name, 5 levels of indexing, and a datavalue.

1.8.2.5 **--with-dbfile=name**

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

1.8.2.6 --with-cache=VAL

Native global database in-memory cache size. The number is the number of blocks (see: *--with-block*) to maintain in memory. Not used by Sqlite3.

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

1.8.2.7 --with-block=blksize

Native global Btree block size. Not used by Sqlite3.

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.

1.8.3 --with-ibuf=

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

1.8.4 --with-strmax=

Maximum internal string size [default: 4096].

1.8.5 --with-locale=locale

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

1.8.6 --with-terminate-on-error

Halt interpreter on error [default: off]

1.8.7 --with-float-bits=val

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

1.8.8 --with-float-digits=val

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

1.8.9 --with-hardware-math={yes|no}

Use hardware arithmetic facilities.

1.8.10 --with-no-inline

Do not use inline functions.

1.8.11 --with-profile

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

1.8.12 --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 Global Array Databaser File(s)

If you are using Sqlite3, be sure you have created **mumps.sqlite** database file using the **mumps-sql-db-create**. If the Sqlite3 database is not found, the program will terminate. No action is needed if you are using the native database as it will create the files it requires (**key.dat** and **data.dat**) as needed.

2.2 Running the 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. You may line edit previously entered commands. To execute, hit *enter*.

Command line input to Mumps follows GNU **readline** conventions.

To exit the Mumps CLI, use the **Halt** (**h** or **H** or **halt**) command or **^d**.

2.3 Interpreting a Mumps Program

Mumps source code programs are ASCII files that can be created by any ASCII text editor.

However, avoid using word processing editors as they may embed hidden formatting characters into the text. These will cause errors.

One way to interpret a Mumps program is to pass the name of the program as an argument to the Mumps interpreter:

mumps progname.mps

This will invoke the Mumps interpreter and pass the name of the program to it for execution. The program file passed need only be readable – execute permission is not required.

Alternatively, you may run a Mumps program directly source file by identifying the Mumps interpreter on the first source code line and making the file executable. In this case, the Mumps program will have the following as its first line:

#!/usr/bin/mumps

The pound sign (#) makes the line a comment to Mumps but **#!** is recognized by the Linux shell as introducing the name of the program (**mumps** in this case) to receive the source file text. The default location of the Mumps interpreter is: **/usr/bin/mumps**.

The Mumps source file must be made executable:

```
chmod u+x progname.mps
```

where **progname.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 **progname.mps** to your terminal prompt. The program above will write **Hello World**, followed by a number ten times.

Lines beginning (column 1) with pound sign (#) are ignored by the Mumps Interpreter and compiler.

2.4 Compiling a Program

The Mumps Compiler is invoked with the script file **mumpsc**. This executable script will translate a Mumps program to an intermediate C++ file and compile the result using the Mumps runtime libraries. The result will be an executable binary. The intermediate C++ file is not deleted and can be inspected or edited. If available, the intermediate file will be processed by **astyle** by default to improve readability.

Most, but not all, Mumps language features are available in the compiler.

The **mumpsc** script may be used to compile a C++ intermediate file from a previous translation from Mumps to C++. You may edit the intermediate file.

*Compiled programs, in nearly all cases, **must** begin with the **zmain** command as the first executable command. Omitting this line will result in many errors.*

You may edit the C++ file created by the Mumps Compiler and include calls to other routines. You may compile the result (the C++ file) to a binary executable also using **mumpsc**.

You should not pass a Mumps compiler generated C++ file directly to the C++ compiler because the C++ file needs access to libraries which the **mumpsc** script command automatically provides.

If you use the Mumps Compiler, you should avoid using the **xecute** command and the indirection operator (@) as these invoke the Mumps interpreter and greatly increase overhead.

3 Relational Database Functions

If Sqlite3 relational database storage for 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. Except as noted, a return value of zero (0) indicates success while a non-zero result indicates failure.

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 Sqlite3.

3.3 \$zSqlite("commit transaction")

Sends a **COMMIT TRANSACTION**; command to Sqlite3.

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

If the second argument is omitted, send a **SAVEPOINT default**; command to Sqlite3. If the second argument is present, send a **SAVEPOINT savepoint**; command to Sqlite3 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 Sqlite3. If the second argument is present, send a **ROLLBACK TRANSACTION savepoint**; command to Sqlite3 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 Source Code Format

C++ and C code were formatted using:

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

C++ generated by the Mumps compiler is formatted in the same manner if *astyle* is available.

4.2 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++ modulo.

4.3 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.

4.4 Notes on Arithmetic Precision

See section 1.7.1 on page 12 for additional details.

4.4.1 \$fnumber()

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

4.4.2 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
```

4.4.3 Extended Arithmetic Precision

If enabled, Mumps will use the GNU *bignum* integer and MPFR floating point packages (this can be enabled/disabled by a **configure** in the **build...** script).

4.4.3.1 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.

4.4.3.2 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*.

4.4.3.3 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.

4.4.4 Rounding

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

4.5 New Command

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

4.5.1 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.

4.5.2 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)
```

4.5.2.1 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 deleted⁴.

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.

⁴ A block is any sequence of code entered as a result of a **do** 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 a 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 variable appears at several layers, only the most recent version will be copied.

An example is given in Figure 3. 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**. Another **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.

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 3 **new** Command without Arguments

4.5.2.2 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.

4.5.2.2.1 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 exited⁵.

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 4.

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
```

⁵ A block is any sequence of code entered as a result of a **do** command.

```

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 4 **new** Command with Comma List

4.5.2.2.2 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 not⁶.

See Figure 5.

```

#!/usr/bin/mumps
  set i=10
  set j=20
  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 5 **new** Command with Parenthesized List

4.6 Kill Command

For non-globals, the **kill** command operates only on the current symbol table level.

4.7 For Command Extensions

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

```
for i=$order(^a(i)) ...
```

⁶ 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.


```
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*.

4.8 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 it 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 of "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 l=1 do
. write "test 1: ",$test,!
. if l=2 write "wow",!
. else write "not wow",!
. write "test 2: ",$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.

4.9 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 19).

4.10 Lock Command in Native Database Mode

The *lock* command has no effect on the single user database as there are no other users to lock data access to.

4.11 Naked indicator

This version of Mumps does not support the naked indicator.

It was originally included in early versions of Mumps because of the 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.

4.12 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 Sqlite3 version has no such restriction.

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

4.13 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 @x
goto @""^/home/user/xxx.mps""
```

Note the extra quotes. These are required in order to embed the file name in quotes.

4.14 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. The underscore character may not be used.

4.15 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
```

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.

4.16 Subroutine & Function Calls

Subroutines and functions may be performed in several ways as shown in Figure 6. 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 6 Subroutine/Function Calls

4.17 \$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
```

4.18 \$Select() Function

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

4.19 Compiling Large Programs

When compiling⁷ large programs, especially if SQL is enabled, there may be a warning about **variable tracking** from the gcc/g++ compiler. You may ignore this.

⁷ Using the compiler is not presently recommended.

4.20 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~
```

4.21 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. Note the omission of the “.” indent.

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,!
+   }
```

4.22 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 in the example in Figure 7, 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 7 Local Functions

4.22.1 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 they are **killed** 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 8 Call by Value Functions

4.22.2 Call by Reference.

Same as the above but 'call by 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 9 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 are 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 10.

```
#!/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 10 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.

5.3.1 Expression 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*.

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

Figure 11 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 database 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, it 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 scalar variables. They may not, at present, be subscripted.

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

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.

7.1.2 \$zDatabase

Returns a number indicating the database server:

- 1 Native database server with disk resident files
- 2 Sql database with disk resident files
- 3 Sql database with memory resident files (transient)

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 12 gives examples.

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

output:
abc xyz 123 ***
```

Figure 12 **\$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) or 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, alphabets 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 alphabets 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 13 gives examples.

```
1  #!/usr/bin/mumps
2  open 1:"tdb,new"
3  for j=1:1:1000 do
4    . use 1
5    . set i=$ztell
6    . set ^a(j)=i
7    . write "**** ",j,!
8
9  close 1
10 open 1:"tdb,old"
11 for j="":$order(^a(j)):"" do
12   . use 1
13   . set i=$zseek(^a(j))
14   . read a
15   . use 5
```



```
16 . write a,!
```

output:

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

```

Figure 13 \$Zseek() Examples

7.5.16 \$zsrnd(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 14 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**

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

output:

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

Figure 14 \$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 15 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 15 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 16.

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

```
4 write "average=",i,!
```

Figure 16 \$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 17 for an example. The matrix must be a top level global array.

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

output:

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

Figure 17 \$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 11.

```
1 #!/usr/bin/mumps
2 kill ^a
3 for i=1:1:10 set ^a(99,i)=i
4 set i=$zzCount(^a(99))
5 write "count=",i,!
```

writes: count=10

Figure 18 \$zzCount() Example

7.6.4 \$zzMax(gbl)

Computes and returns the maximum numeric value in the vector and any descendants. See Figure 19 for an example.

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

output:

10

Figure 19 \$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 20 for an example.

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

```

output:

2

Figure 20 \$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 24 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 25.

7.6.8 \$zzTranspose(gblMatrix1,gblMatrix2)

Transposes the first global array matrix leaving the result in the second. For example, see Figure 26. 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 21 and an example in code is given in Figure 22. 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.

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

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

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

$$Similarity_{i1}(i, j) = \sum_{k=1}^{k=t} Term_{ik} \cdot Term_{jk}$$

Figure 21 Similarity Formulae

```

1  #!/usr/bin/mumps
2  kill ^A
3  kill ^B
4
5  set ^A("1")=3
6  set ^A("2")=2
7  set ^A("3")=1
8  set ^A("4")=0
9  set ^A("5")=0
10 set ^A("6")=0
11 set ^A("7")=1
12 set ^A("8")=1
13
14 set ^B("1")=1
15 set ^B("2")=1
16 set ^B("3")=1
17 set ^B("4")=0
18 set ^B("5")=0
19 set ^B("6")=1
20 set ^B("7")=0
21 set ^B("8")=0
22
23 write "Cosine=", $zzCosine(^A, ^B), !
24 write "Siml=", $zzSiml(^A, ^B), !
25 write "Dice=", $zzDice(^A, ^B), !
26 write "Jaccard=", $zzJaccard(^A, ^B), !

```

output:

```

Cosine=0.75
Siml=6
Dice=1
Jaccard=1

```

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 23 gives an example.

```
1 #!/usr/bin/mumps
2 set key="now"
3 set str="now is the now of the now in the know"
4 write $zBMGSearch(key,str),!
```

output:

4

Figure 23 \$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 27 contains examples (long lines wrapped).

```
1 #!/usr/bin/mumps
2 set ^d("1","1")=2
3 set ^d("1","2")=3
4 set ^d("2","1")=1
5 set ^d("2","2")=-1
6 set ^d("3","1")=0
7 set ^d("3","2")=4
8
9 set ^e("1","1")=5
10 set ^e("1","2")=-2
11 set ^e("1","3")=4
12 set ^e("1","4")=7
13 set ^e("2","1")=-6
14 set ^e("2","2")=1
15 set ^e("2","3")=-3
16 set ^e("2","4")=0
17
18 set %=$zzMultiply(^d,^e,^f)
19
20 for i="" :$order(^f(i)):"" do
21 . for j="" :$order(^f(i,j)):"" do
22 .. 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 24 \$zzMultiply() Example

```

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

```

output:

```
55
```

Figure 25 \$zzSum() Example

```

1 #!/usr/bin/mumps
2 kill ^f
3
4 set ^d("1","1")=2
5 set ^d("1","2")=3
6 set ^d("2","1")=4
7 set ^d("2","2")=0
8
9 set %=$zzTranspose(^d,^f)
10
11 for i="":$order(^f(i)):"" do
12 . for j="":$order(^f(i,j)):"" do
13 .. write i," ",j," ",^f(i,j),!

```

output:

```

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

```

Figure 26 \$zzTranspose() Example

```

1 #!/usr/bin/mumps
2 write "Please enter a telephone number:",!
3 read phonenum
4
5 set p="^(1-)?(\{3\})?(-| )?\{3\}-?\{4\}$"
6 if $zperlmatch(phonenum,p) do
7 . write "+++ This looks like a phone number.",!
8 . write "The area code is: ",$2,!
9 else do
10 . 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 27 `$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 28 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 29 contains an example.

```
1 #!/usr/bin/mumps
2 set a="now is the time for all"
3 set a=$zReplace(a,"is","IS")
4 write a,!
5
6 set a="[[now is the time]]"
7 if $zPerlMatch(a,"(\[\[)(.*)\]\]") do
8 . set a=$zReplace(a,$2,"ABC")
9 . write a,!
```

output:

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

Figure 28 `$zReplace()` Example

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

output:

```
nowis
theti
```

```

mefor
allgo
odmen
tocon
etoth
eaido
fthep

```

Figure 29 \$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 30.

```

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

```

output:

nowis	tothe	goodm
theti	aidof	entoc
mefor	thepa	ometo
allgo	wisth	theai
odmen	etime	dofth
tocon	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 30 \$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 31 gives an example.

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

output:

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

score=23

Figure 31 \$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 32 gives an example. The vector argument *global* must be a top level array.

```
1 #!/usr/bin/mumps
2 set ^a("now")=2
3 set ^a("is")=5
4 set ^a("the")=6
5 set ^a("time")=3
6 set j=4
7 set %=$zzIDF(^a,j)
8 for i="":$order(^a(i)):"" write i," ",^a(i),!
```

```
output:
```

```
is 0.7  
now 2.0  
the 0.4  
time 1.4
```

Figure 32 \$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 33 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.

```
1  #!/usr/bin/mumps  
2  kill ^A,^B  
3  
4  set ^A("1","computer")=5  
5  set ^A("1","data")=2  
6  set ^A("1","program")=6  
7  set ^A("1","disk")=3  
8  set ^A("1","laptop")=7  
9  set ^A("1","monitor")=1  
10  
11 set ^A("2","computer")=5  
12 set ^A("2","printer")=2  
13 set ^A("2","program")=6  
14 set ^A("2","memory")=3  
15 set ^A("2","laptop")=7  
16 set ^A("2","language")=1  
17  
18 set ^A("3","computer")=5  
19 set ^A("3","printer")=2  
20 set ^A("3","disk")=6  
21 set ^A("3","memory")=3  
22 set ^A("3","laptop")=7  
23 set ^A("3","USB")=1  
24  
25 set %=$zzTermCorrelate(^A,^B)  
26  
27 for i="":$order(^B(i)):"" do  
28 . write i,!  
29 . for j="":$order(^B(i,j)):"" do  
30 .. write ?10,j," ",^B(i,j),!
```

output:

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

Figure 33 \$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 corresponding 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 34 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.

```

1 #!/usr/bin/mumps
2 kill ^A,^B
3
4 set ^A("1","computer")=5
5 set ^A("1","data")=2
6 set ^A("1","program")=6
7 set ^A("1","disk")=3

```

```

8  set ^A("1","laptop")=7
9  set ^A("1","monitor")=1
10
11 set ^A("2","computer")=5
12 set ^A("2","printer")=2
13 set ^A("2","program")=6
14 set ^A("2","memory")=3
15 set ^A("2","laptop")=7
16 set ^A("2","language")=1
17
18 set ^A("3","computer")=5
19 set ^A("3","printer")=2
20 set ^A("3","disk")=6
21 set ^A("3","memory")=3
22 set ^A("3","laptop")=7
23 set ^A("3","USB")=1
24
25 set %=$zDocCorrelate(^A,^B,"Cosine",.5)
26
27 for i="":$order(^B(i)):"" do
28   . write i,!
29   . for j="":$order(^B(i,j)):"" do
30     .. 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 34 \$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 35 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 35 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 function. Otherwise, the function will return 1 (success) or 0 (failure).

8 Pattern Matching

8.1 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`.

8.2 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)?(-| )?\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).

9 Mumps Compiler

Included in the distribution package is a compiler for the Mumps language and the Multi-Dimensional and Hierarchical library (MDH) (see page 60).

The compiler translates Mumps source code to C++ then compiles the C++ file to an executable binary.

The MDH toolkit consists of a collection of class libraries and other software that facilitate Mumps style coding in C++. It supports global arrays, Mumps-style string operations, and many functions.

9.1 How to Compile and Run a Mumps

In order to be compiled, programs written in Mumps must have the extension **.mps**.

When you compile a Mumps program, a C++ translation of your program is created and written to 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.

You may compile a Mumps program by using the executable script **mumpsc**. To compile a Mumps program using the script, type:

```
mumpsc myprog.mps
```

The actual 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.

Both ordinary C++ programs that use the MDH class libraries, and C++ programs created by the Mumps compiler, make use of the same Mumps compile and runtime libraries. The **mumpsc** script passes libraries these to the **gpp** compiler. Compiling Mumps related code with the **gpp** compiler without these libraries will result in errors.

9.2 Compiler Error Messages

Generally speaking, in most cases you will receive syntax error messages from the Mumps compiler. These 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 ++ compiler error messages, the line number on the error message will refer to the line number in the C++ translation of your Mumps program. To connect 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 like⁸:

```
/*=====*
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)
```

⁸ The code in the example does not contain an error. It is for example purposes only.

```
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 36 Example C++ Code

Note 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 normally results in multiple lines of C++ code.

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

9.3 Global Array Storage in Compiled Programs

Global arrays will be stored in Sqlite3 or the native Btree database depending on which installer you used to build the interpreter. Global arrays created by compiled programs are interchangeable with global arrays created by the interpreter.

9.4 Compiler Performance and Interoperability

Compiled modules, exclusive of database access, execute faster than the same code executing on the interpreter. However, programs with large amounts of database or I/O activity will run at about the same speed as interpreted versions of the same program.

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.

10 The Multi-Dimensional and Hierarchical Database Toolkit

10.1 Introduction

The MDH (Multi-Dimensional and Hierarchical) Database Toolkit is a C++ Linux-based, open sourced, toolkit of software that supports fast, flexible, multi-dimensional and hierarchical storage, retrieval and manipulation of information in data bases in a manner similar to the Mumps language.

The package is written in C and C++ and is available under the GNU GPL/LGPL licenses in source code form.

The distribution kit contains demonstration implementations of text and sequence retrieval tools that function with very large genomic data bases and illustrate the toolkit's capability to manipulate massive data sets of genomic information.

The toolkit is distributed as part of the Mumps Compiler for Linux.

The toolkit is a solution to the problem of manipulating very large, character string indexed, multi-dimensional, sparse matrices. It is based on Mumps (also referred to as *M*), a general purpose programming language that originated in the mid 60's at the Massachusetts General Hospital. The toolkit supports access to the SQLite relational data base server, the Perl Compatible Regular Expression Library, and the Glade GUI builder.

The principal database feature in this project is the ***global array*** which permits direct, efficient manipulation of multi-dimensional arrays of effectively unlimited size.

A global array is a persistent, sparse, undeclared, multi-dimensional, string indexed data disk based structure. A global array may appear anywhere an ordinary array reference is permitted and data may be stored at leaf nodes as well as intermediate nodes in the data base array. The number of subscripts in an array reference is limited only by the system's maximum length array reference restriction with all subscripts expanded to their string values. The toolkit includes several functions to traverse the data base and manipulate the arrays.

The toolkit makes the data base and function set available as C++ classes and also permits execution of legacy Mumps scripts. To use the toolkit, you install the MDH and Mumps distribution. kit and related code.

10.2 Installation

The class libraries and related functions along with the Mumps Compiler and Interpreter must be installed before attempting to use the MDH package. The Mumps Compiler/Interpreter distribution code has instructions in **README.txt** on how to install the software and a description of options available (see page 6).

The distribution contains a number of example programs written both in Mumps and C++/MDH.

10.3 Compiling Programs

To compile a C++ program that uses the MDH (Multi-Dimensional and Hierarchical) library, use the command:

```
mumpsc myprog.cpp
```

This will invoke the ***g++*** compiler and make available the necessary libraries. The result will be a program named *myprog* which is executable. The script ***mumpsc*** is part of the Mumps Compiler which must be installed prior to using the toolkit.

Note: the *mumpsc* command is also used to compile Mumps source code to C++ and then to binary executables so Mumps programs may be executed directly rather than by the Mumps interpreter.

10.4 Writing C++ MDH Programs

In order to use global arrays or other MDH features, each C++ program must include the MDH header file at the beginning of the program:

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

This header and related library code is installed on your system when you install the Mumps Compiler/Interpreter software.

10.5 MDH Implementation of Globals

The main feature of the MDH is its implementation of Mumps global arrays as a C++ class. This implementation also includes a number of builtin functions to manipulate and traverse global arrays as well as a class of string, **mstring**, which imitates the behavior of Mumps strings.

Global arrays are undimensioned, string indexed, disk resident data structures whose size is limited only by available disk space.

For example:

```
patient("1234", "Labs", "hct", "31 May 2022 03:05:56 PM EDT") = 44;9
```

where a node or cell in the global array *patient*, indexed by the four strings shown, is assigned the value 44.

The resulting global array *patient* node can be viewed either as a leaf node in a four level tree (where the array indices select the tree path) or as a cell in a four dimensional matrix.

Global arrays are derived from a C++ **class**. Instances must be declared in your C++ program as instances of class **global**.

For example, to create the global array named **gbl**, use the following:

```
global gbl("gbl");
```

The instance consists of two parts: the name of the global array object and the name of the global array on disk associated with this object. Normally, these are the same.

In the above example, they are both **gbl**. Note that the disk name of the global is enclosed in a parenthesized character string expression following the object name.

The value in the expression need not (but usually does) match the name of the object. The name given in the parenthesized character string is the disk name of the global array. The global array object is associated with the disk name when the object is created. When the program object is destroyed (for example, at program termination), the disk based global array persists.

Note: programs that use global arrays MUST close the array file system with the **GlobalClose;** command before exiting. Failure to do so may corrupt the file system.

Global objects may be created through declarations as shown above or dynamically:

```
global *gptr;  
gptr = new global ("gbl_name");  
(*gptr)("1","2","3") = "test";
```

⁹ Blanks in the Mumps code have been inserted for clarity. In actual Mumps, they would be errors. Where appropriate, blanks are permitted in C++ code.

which is equivalent to:

```
global g("gbl_name");  
g("1","2","3") = "test";
```

Each **global** declaration creates a global array as an object or instance of the **global** class. Each global array you use must be first declared as an object of the **global** class. Global names can be any valid C/C++ variable name.

A global array will typically have one or more subscripts as discussed below. These will be of type **mstring**, or a null terminated array of **char**. Subscripts of global arrays must evaluate to printable characters in the range of decimal 32 (space) to, but not including, decimal 126 (tilde ~).

Note:

- No data types other than **mstring**, or a null terminated array of **char** (*i.e.*, **char ***) may be used as subscripts. Numeric data types (**int**, **short**, **long**, **float**, **double**, etc.) may not be used as global array subscripts.
- In any given global array reference, all the indices must all be of the same data type (**mstring** or **char ***)

mstring is a data type (class) whose behavior is similar to the basic typeless string data type used in Mumps.

Objects of **mstring** are stored internally as strings and may contain text, integers and floating point values.

Addition, multiplication, subtraction, division, modulo, and concatenation may be performed directly on **mstring** objects (see details below). Many of the following examples use **mstring** objects.

10.6 Global Arrays as Trees and Matrices

10.6.1 Traditional Matrix

Global arrays may be viewed either as sparse multi-dimensional matrices or as tree structured hierarchies.

As matrices, data may be stored at fully subscripted matrix elements but also at other levels in the manner used by many programming languages. For example, given a three dimensional matrix *mat1*, you could initialize it as shown in Figure 37.

```
#include <mumpsc /libmpscpp.h>  
  
global mat1("mat1");  
  
int main() {  
    mstring i, j, k;  
    for (i = 0; i < 100; i++)  
        for (j = 0; j < 100; j++)  
            for (k = 0; k < 100; k++) {  
                mat1(i, j, k) = 0;  
            }  
  
    GlobalClose;  
    return 0;  
}
```

Figure 37 Global Array as a Matrix

Alternatively, the above can be performed with **int** but the numeric indices must be converted to **mstring** before use. See Figure 38

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

global mat1("mat1");

int main() {
    int i, j, k;
    for (i = 0; i < 100; i++)
        for (j = 0; j < 100; j++)
            for (k = 0; k < 100; k++) {
                mat1(mcv1(i), mcv1(j), mcv1(k)) = 0;
            }
    GlobalClose;
    return 0;
}
```

Figure 38 Global Array as Matrix with Numeric Subscripts

In this example, all the elements of a three dimensional matrix of 100 rows, 100 columns and 100 planes are initialized to zero. The function *mcvt()* converts from **int** to **mstring**.

In the view expressed by the code above, the matrix is a traditional three dimensional structure with data stored at each fully indexed position or node.

10.6.2 Alternative Matrix View

Unlike other programming languages, however, there are additional nodes of the matrix which could have been initialized as indicated by Figure 39.

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

global mat1("mat1");

int main() {
    mstring i, j, k;
    for (i = 0; i < 100; i++) {
        mat1(i) = i;
        for (j = 0; j < 100; j++) {
            mat1(i, j)=j;
            for (k = 0; k < 100; k++) {
                mat1(i, j, k) = 0;
            }
        }
    }
    return 0;
}
```

Figure 39 Global Array as Matrix with Additional Nodes

In effect, this means that **mat1** can also be a single dimensional vector, a two dimensional matrix and a three dimensional matrix simultaneously.

Furthermore, not all elements of a matrix need exist. That is, the matrix can be sparse as shown in Figure 40.

```

#include <mumpsc/libmumpscpp.h>

global mat1("mat1");

int main() {
  mstring i, j, k;
  for (i = 0; i < 100; i = i + 10)
    for (j = 0; j < 100; j = j + 10) {
      for (k = 0; k < 100; k = k + 10) {
        mat2(i, j, k) = 0;
      }
    }
  return 0;
}

```

Figure 40 Global Array as Sparse Matrix

In the above, only index values 0, 10, 20, 30, 40, 50, 60, 70, 80, and 90 are used to create each of the dimensions of the array and only those elements of the matrix are created. The omitted elements do not exist.

For example, if you are running a drug protocol on a number of patients and are dosing with medications M1, M2, M3, ... on patients P1, P2, P3, ... and collecting observations on days D1, D2, D3, ... you could create a three dimensional matrix named *protocol* in which each plane consisted of the observations for each patient on each medication for a given day as shown in Figure 41.

D1						D2						D3						D4					
	M1	M2	M3	M4	M5		M1	M2	M3	M4	M5		M1	M2	M3	M4	M5		M1	M2	M3	M4	M5
P1						P1						P1						P1		X			
P2						P2						P2						P2					
P3						P3						P3						P3					

Figure 41 Tabular View of Tree

You could refer to patient P1, medication M2 on day D4 with the reference:

```
protocol("P1", "M2", "D4")="X";
```

10.6.3 Tree View

Alternatively, you can view the same data base as a tree structure with patient id at the root, followed by medication, followed by day of study as shown in Figure 42.

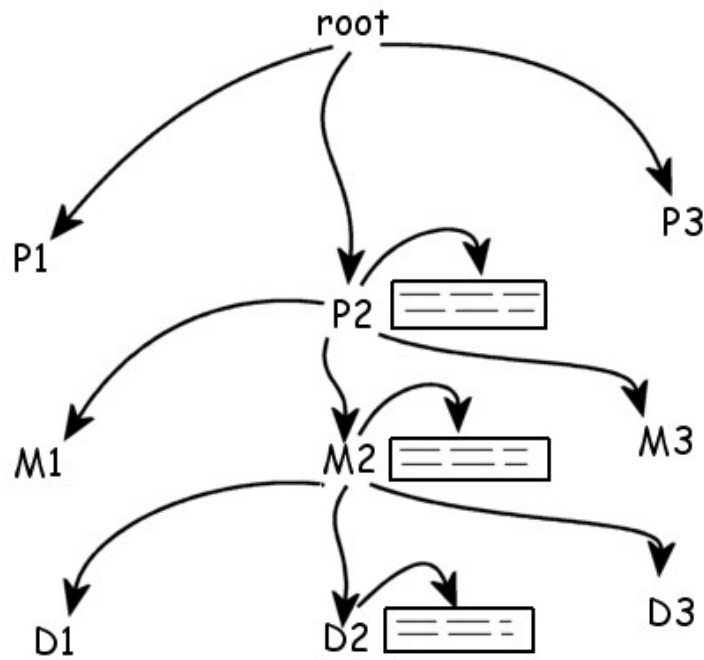


Figure 42 Global Array Tree

Note that at each node in the tree, a data box may appear containing information about the node. Addressing a node is accomplished by giving its path description such as:

```
protocol("P2", "M2", D2)
```

10.7 Accessing Global Arrays

Note: prior to exiting a program that accessed globals arrays, you should execute the *GlobalClose* macro to shut down the global array facility. This flushes the system buffers to disk and insures that the file system is properly closed. Failure to do this may result in data base errors. This appears in your program as:

```
GlobalClose;
```

You may assign global array elements to variables of type **mstring** using the assignment operator (=).

You may assign values of type **int**, **float**, **double**, **mstring**, **string** and **char *** to global array elements using the assignment operator (=).

When global array references are passed to function, no more than one instance of the same **global** object should be used in the argument list. Each **global** object maintains a private static string which contains the most recent value fetched from the data base. When a **global** object is passed to a function, its this string value is effectively passed. This means that, in a function reference where two references to the same **global** object are passed, even though they have differing indices, the value passed will be the value for the second instance of the **global**. This restriction only applies where there are two or more instances of the same **global**.

If you use a reference to a **global** without a parenthesized list following the name of the **global**, the reference will be to the most recent referenced **global**. Effectively, this is similar to the "naked indicator" from Mumps.

10.8 Global Array Indices

Internally, the indices of global arrays are always stored as character strings. If you initialize a global array with a loop, you must insure that the indices are represented as either values of type **mstring** or null

terminated arrays of type **char**. Indices to globals may be either **char*** or **mstring** but MUST all be of the same type (i.e. all **char*** or all **mstring**). For example:

```
mstring A, B, C;

for (A = 0; A < 1000; A++)
    for (B = 0; B < 1000; B++)
        for (C = 0; C < 1000; C++) {
            array1(A, B, C) = "0";
        }
```

The above initializes an array of 1 billion elements to zero.

10.9 Navigating Globals

There are several builtin functions used to navigate the globals. The two most important are the *Data()* function and the *Order()* function. The *Data()* function tells you if a node exists and if it has descendants and the *Order()* function gives you the next higher (or lower) index at a given level in the global array tree.

The *Data()* function returns an integer which indicates whether the global array node is defined:

1. 0 if the global array node is undefined;
2. 1 if it is defined and has no descendants;
3. 10 if it is defined but has no value stored at the node (but does have descendants);
4. 11 if it is defined and has descendants.

A global is defined if data has been stored at it. A "10" is returned for a node at which nothing has been stored but the node has descendants. For example, assuming the global array has only the contents created in the example in Figure 43.

```
global array1("array1");

int result;

array1("1","11") = "foo"
array1("1","11","21") = "bar"

result = array1("1").Data() ;           // yields 10
result = array1("1","11").Data();       // yields 11
result = array1("1","11","21").Data();  // yields 1
```

Figure 43 Navigating Global Arrays - Data()

The other major navigation function is the *Order()* function. This gives you, for a given global array index, the next ascending or descending value for the last index. If the parameter to *Order()* is 1 or missing, the next ascending index is returned. If the parameter is -1, the next descending index is returned. To get the first (or last if the parameter is -1) value of an index, start with a null (empty) string. See Figure 44.

```
mstring x, null;
global array1("array1");

array1("100") = "a";           // initialize the array with three entries
array1("200") = "b";
array1("300") = "c";

null = "";
```

```

x = array1(null).Order();    // get the first value of the first index: 100
x = array1(x).Order();      // get the second value of the first index: 200
x = array1(x).Order();      // get the third value of the first index: 300
x = array1(x).Order();      // no more indices - returns empty string
x = array1(null).Order(-1);  // get the last value of the first index: 300
x = array1(x).Order(-1);    // get the second value of the first index: 200
x = array1(x).Order(-1);    // get the first value of the first index: 100
x = array1(x).Order(-1);    // no more indices - returns empty string

for ( x = array1(null).Order(); x != null; x = array1(x).Order())
    cout << endl;          // writes 100 200 300 on separate lines

for ( x = array1(null).Order(-1); x != null; x = array1(x).Order(-1))
    cout << endl;          // writes 300 200 100 on separate lines

for ( x = 10; x < 100; x = x + 10) array1("200" , x) = x;

for ( x = array1("200", null).Order(); x != null; x = array1("200", x).Order())
    cout << endl;          // writes 10 20 30 ... 90 on separate lines

```

Figure 44 Navigating Global Arrays - Order()

Each call to *Order()* gives the next value of the last index. The numeric parameter indicates if the direction is ascending (1) or descending (-1). If omitted, 1 is assumed. To get the first index, the empty string is supplied and the function returns the first index of the global array. For subsequent calls, it returns the next ascendant index value until there are no more indices. Then it returns the empty string.

In the following example, we build a global array vector from an input file consisting of keywords with one keyword per line, keep a count of each time the keyword is used, and, at the end, print an alphabetized list of the keywords followed by the number of times each occurs, do as shown in Figure 45.

```

#include <mumpsc/libmpscpp.h>
global key("key");

int main() {

mstring word, null;
long i;

null = "";

while (1) {
    if ( ! word.ReadLine(cin)) break;
    if (key(word).Data())          // is word in vector?
        key(word)++;              // yes, increment count
    else key(word) = 1;            // not in vector - add
}

word = null;

while ((word = key(word).Order(1)) != null) // next word

cout << word << " " << key(word) << endl; // print word and count

return EXIT_SUCCESS;
}

```

Figure 45 Global Array Navigation Example

In the above, each line is read into the variable *word* until the end of file is reached. Each word is tested with the *Data()* function of the global array to determine if *word* exists in the *key* vector. The *Data()* returns zero if the element does not exist, non-zero if it does. In the case where the word is in the *key* global array vector, the value stored in the vector for the word is extracted into the variable *i*, incremented and stored back into the vector. If the *word* does not exist in the vector, it is added and its initial count is set to one.

When all the words have been read and stored into the vector, the program sequences through the word entries and prints the words and the total number of times each one was present in the input file. Since global arrays are stored in ascending key order, the display of words will be alphabetic.

Similarly, given a global array of patient lab data organized hierarchically first by patient id, then by lab test, then by date, we can print a table of patient id's, labs, dates and results as shown in Figure 46.

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

global Labs("labs");

int main() {

mstring null, ptid, lab_test, date, rslt;

null = "";

// create dummy example data base

Labs("1000", "hct", "July 12, 2003") = "45";
Labs("1000", "hct", "July 13, 2003") = "46";
Labs("1000", "hct", "July 14, 2003") = "47";
Labs("1000", "hct", "July 15, 2003") = "48";
Labs("1000", "hgb", "July 12, 2003") = "15";
Labs("1000", "hgb", "July 15, 2003") = "14";
Labs("1001", "hct", "July 12, 2003") = "35";
Labs("1001", "hct", "July 13, 2003") = "36";
Labs("1001", "hct", "July 14, 2003") = "37";
Labs("1001", "hct", "July 15, 2003") = "38";
Labs("1001", "hgb", "July 13, 2003") = "15";
Labs("1001", "hgb", "July 14, 2003") = "15";
Labs("1002", "hct", "Sept 12, 2003") = "35";
Labs("1002", "hct", "Sept 13, 2003") = "36";
Labs("1002", "hct", "Sept 14, 2003") = "37";
Labs("1002", "hct", "Sept 15, 2003") = "38";
Labs("1002", "hgb", "Sept 13, 2003") = "15";
Labs("1002", "hgb", "Sept 14, 2003") = "15";

ptid = null;

while ( (ptid = Labs(ptid).Order(1)) != null) {
    lab_test = null;
    while ( (lab_test = Labs(ptid,lab_test).Order(1)) != null) {
        date = null;
        while ( (date = Labs(ptid,lab_test,date).Order(1)) != null) {
            cout << ptid << " " << lab_test << " " << date;
            cout << " " << Labs(ptid,lab_test,date) << endl;
        }
    }
}

GlobalClose;
```

```

    return 1;
}

```

Output

```

1000 hct July 12, 2003 45
1000 hct July 13, 2003 46
1000 hct July 14, 2003 47
1000 hct July 15, 2003 48
1000 hgb July 12, 2003 15
1000 hgb July 15, 2003 14
1001 hct July 12, 2003 35
1001 hct July 13, 2003 36
1001 hct July 14, 2003 37
1001 hct July 15, 2003 38
1001 hgb July 13, 2003 15
1001 hgb July 14, 2003 15

```

Figure 46 Hierarchical Global Array Example

The example in Figure 46 begins with an empty string for patient id *ptid*. This is used at the outer loop level to cycle through all the patient ids. At the first nexted loop, the program cycles through all the lab test names (*lab_test*) then at the innermost level, it cycles through all the dates (*date*). The resulting table is of the form:

10.10 Locking the Data Base

There are several functions for locking portions of the data base. Following legacy convention, a lock does not prevent access to an element but merely flags the element as locked. Locking views a global array as a tree structure. If an element is locked, its descendants are locked. An attempt to lock a locked element of an element that has a locked parent or a locked descendant will fail. The primary locking functions are *\$lock()*, *Lock()* and *UnLock()*:

```

if ($lock(gbl(a, b, c)) cout << "locked" << endl;
if (gbl(a, b, c).Lock()) cout << "locked" << endl;
gbl(a, b, c).UnLock();

```

The *\$lock()* and *Lock()* functions test to see if the node can be locked and locks it if possible. It returns *true* (1) if successful and false (0) otherwise (*\$test* is set accordingly). A node can be locked if it itself is not locked, if it has no descendants that are locked and if it is not the descendant of a locked node. The *UnLock()* function releases a lock on a node.

Additionally, there are functions to release all locks for the current process and all locks for all processes:

```

CleanLocks();      // release all locks for this process only
CleanAllLocks();   // release all locks for all processes

```

10.11 Class mstring

The **mstring** class provides Mumps-like strings that can be used to in C++ programs. They treat variable values in a manner similar to that of native Mumps strings.

mstring objects are based on C++ **string** strings on which arithmetic and other operations may be performed. The **mstring** includes overloads for many operators as well provides many Mumps-like functions.

10.12 mstring Operations

The operator overload for **mstring** are shown in Figure 73. Additional overloads may be added in time.

Addition

<code>mstring operator+(int);</code>	<code>friend mstring operator+(int,mstring);</code>
<code>mstring operator+(long);</code>	<code>friend mstring operator+(long,mstring);</code>
<code>mstring operator+(double);</code>	<code>friend mstring operator+(double,mstring);</code>
<code>mstring operator+(float);</code>	<code>friend mstring operator+(float,mstring);</code>
<code>mstring operator+(mstring);</code>	<code>friend mstring operator+(string,mstring);</code>
<code>mstring operator+(string);</code>	<code>friend mstring operator+(global,mstring);</code>
<code>mstring operator+(const char *);</code>	<code>friend mstring operator+(const char *,mstring);</code>
<code>mstring operator+(char *);</code>	<code>friend mstring operator+(char *,mstring);</code>
<code>mstring operator+(global);</code>	
<code>mstring operator+=(mstring);</code>	
<code>mstring operator+=(int);</code>	
<code>mstring operator+=(long);</code>	
<code>mstring operator+=(double);</code>	
<code>mstring operator+=(float);</code>	
<code>mstring operator+=(string);</code>	
<code>mstring operator+=(const char *);</code>	
<code>mstring operator+=(global);</code>	

Subtraction

<code>mstring operator-(int);</code>	<code>mstring operator-(int);</code>
<code>mstring operator-(long);</code>	<code>mstring operator-(long);</code>
<code>mstring operator-(double);</code>	<code>mstring operator-(double);</code>
<code>mstring operator-(float);</code>	<code>mstring operator-(float);</code>
<code>mstring operator-(mstring);</code>	<code>mstring operator-(mstring);</code>
<code>mstring operator-(string);</code>	<code>mstring operator-(string);</code>
<code>mstring operator-(const char *);</code>	<code>mstring operator-(const char *);</code>
<code>mstring operator-(char *);</code>	<code>mstring operator-(char *);</code>
<code>mstring operator-(global);</code>	<code>mstring operator-(global);</code>
<code>mstring operator-=(mstring);</code>	
<code>mstring operator-=(int);</code>	
<code>mstring operator-=(long);</code>	
<code>mstring operator-=(double);</code>	
<code>mstring operator-=(float);</code>	
<code>mstring operator-=(string);</code>	
<code>mstring operator-=(const char *);</code>	
<code>mstring operator-=(global);</code>	

Multiplication

<code>mstring operator*(int);</code>	<code>friend mstring operator*(int,mstring);</code>
<code>mstring operator*(long);</code>	<code>friend mstring operator*(long,mstring);</code>

<pre> mstring operator*(double); mstring operator*(float); mstring operator*(mstring); mstring operator*(string); mstring operator*(global); mstring operator*(const char *); mstring operator*=(mstring); mstring operator*=(int); mstring operator*=(long); mstring operator*=(double); mstring operator*=(float); mstring operator*=(string); mstring operator*=(const char *); mstring operator*=(global); </pre>	<pre> friend mstring operator*(double,mstring); friend mstring operator*(float,mstring); friend mstring operator*(string,mstring); friend mstring operator*(global,mstring); friend mstring operator*(const char *,mstring); </pre>
Division	
<pre> mstring operator/(int); mstring operator/(long); mstring operator/(double); mstring operator/(float); mstring operator/(mstring); mstring operator/(string); mstring operator/(global); mstring operator/(const char *); mstring operator/=(mstring); mstring operator/=(int); mstring operator/=(long); mstring operator/=(double); mstring operator/=(float); mstring operator/=(string); mstring operator/=(const char *); mstring operator/=(global); </pre>	<pre> friend mstring operator/(int,mstring); friend mstring operator/(long,mstring); friend mstring operator/(double,mstring); friend mstring operator/(float,mstring); friend mstring operator/(string,mstring); friend mstring operator/(global,mstring); friend mstring operator/(const char *,mstring); </pre>
Increment/Decrement	
<pre> mstring operator++(); mstring operator--(); mstring operator++(int); mstring operator--(int); </pre>	
Unary Operations	
<pre> mstring operator!(); // unary mstring operator+(); // unary mstring operator-(); // unary </pre>	
Modulo	
<pre> mstring operator%(long); mstring operator%(int); mstring operator%(mstring); mstring operator%(string); mstring operator%(const char *); mstring operator%(global); mstring operator%=(mstring); mstring operator%=(int); mstring operator%=(long); </pre>	<pre> friend mstring operator%(int,mstring); friend mstring operator%(long,mstring); friend mstring operator%(string,mstring); friend mstring operator%(global,mstring); friend mstring operator%(const char *,mstring); </pre>

```
mstring operator%=(string);
mstring operator%=(const char *);
mstring operator%=(global);
```

Concatenation

```
mstring operator||(mstring);
mstring operator||(string);
mstring operator||(global);
mstring operator||(const char *);
mstring operator||(int);
mstring operator||(long);
mstring operator||(float);
mstring operator||(double);

friend mstring operator||(string, mstring);
friend mstring operator||(global, mstring);
friend mstring operator||(const char *, mstring);
friend mstring operator||(int, mstring);
friend mstring operator||(long, mstring);
friend mstring operator||(float, mstring);
friend mstring operator||(double, mstring);

mstring operator&(mstring);
mstring operator&(char *);
mstring operator&(global);
mstring operator&(string);
mstring operator&(int);
mstring operator&(long);
mstring operator&(double);
```

Relational

```
bool operator==(int);
bool operator==(long);
bool operator==(double);
bool operator==(float);
bool operator==(const char *);
bool operator==(char *);
bool operator==(string);
bool operator==(global);
bool operator==(mstring);

bool operator!=(int);
bool operator!=(long);
bool operator!=(double);
bool operator!=(float);
bool operator!=(char *);
bool operator!=(const char *);
bool operator!=(string);
bool operator!=(global);
bool operator!=(mstring);

bool operator<(int);
bool operator<(long);
bool operator<(double);
bool operator<(float);
bool operator<(char *);
bool operator<(const char *);
bool operator<(string);
bool operator<(global);
bool operator<(mstring);

bool operator>(int);
bool operator>(long);
bool operator>(double);
bool operator>(float);
bool operator>(char *);
bool operator>(const char *);
```



```

bool operator>(string);
bool operator>(global);
bool operator>(mstring);

bool operator>=(int);
bool operator>=(long);
bool operator>=(double);
bool operator>=(float);
bool operator>=(char *);
bool operator>=(const char *);
bool operator>=(string);
bool operator>=(global);
bool operator>=(mstring);

bool operator<=(int);
bool operator<=(long);
bool operator<=(double);
bool operator<=(float);
bool operator<=(char *);
bool operator<=(const char *);
bool operator<=(string);
bool operator<=(global);
bool operator<=(mstring);

```

Figure 47 **mstring** Operator Overloads

10.13 mstring Functions and Methods

10.13.1 Ascii Function

```

int mstring::Ascii()
int mstring::Ascii(int start)
int mstring::Ascii(int start)

```

Returns the numeric value of an ASCII character. If no "start" is specified, the numeric values of the first character of invoking mstring is used. If "start" is specified, the numeric value of "start"th character of nvoking is chosen. If the empty string is given, -1 is returned.

```

mstring a;
a="ABC";
a.Ascii() yields 65
a.Ascii(1) yields 65
a.Ascii(2) yields 66

```

10.13.2 begins Function

```
int mstring::begins(mstring pattern)
```

Returns an integer which is the starting point in the string of pattern or -1 if the pattern is not found. Throws: PatternException if the pattern is in error.

10.13.3 c_str Function

```
char * mstring::c_str()
```

Returns a pointer to a null terminated char array containing the contents of the invoking mstring object.

10.13.4 decorate Function

```
int mstring::decorate(mstring pattern, mstring prefix, mstring suffix)
```

Attempts to locate *pattern* in the invoking mstring and inserts *prefix* immediately to the left of the string that matched the pattern and inserts *suffix* immediately to the right of the found pattern. Returns 1 if the pattern was found and the insertions were made, -1 if the pattern was not found, and less than -1 for other errors (see PCRE documentation concerning `pcre_exec()` return codes). Throws: `PatternException()`.

10.13.5 EncodeHTML Function

```
char * mstring EncodeHTML(char * arg)  
mstring EncodeHTML(mstring arg)
```

Encodes the argument string according to HTML rules and returns the result. Alphabets and numbers are unchanged. Blanks become plus signs and all other characters replaced by "%xx" where "xx" is the hexadecimal value of the character in the ASCII collating sequence. The function is used mainly in connection with parameters passed with URL's which may not contain blanks or special characters. the code in *cgi.h* is used to decode these strings. Example:

```
#include <mumpsc /libmpscpp.h>  
int main() {  
    char x[]="now is =()$.& the time";  
    cout << EncodeHTML(x) << endl;  
    return EXIT_SUCCESS;  
}
```

Yields

```
now+is+%3D%28%29%24%2E%26+the+time
```

10.13.6 ends Function

```
int mstring::ends(mstring pattern)
```

Returns an integer giving the character position (relative to zero) immediately following the string that matched pattern. Returns -1 if the string did not match. Throws: `PatternException`.

10.13.7 Eval Function

```
mstring mstring::Eval()
```

Evaluates the mumps expression of 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 involving calling program mstring variables.

10.13.8 Extract Function

```
mstring mstring::Extract(int=1, int=-1)
```

Returns an mstring containing a substring of the first argument. The substring begins at the position noted by the second operand. If the third operand is omitted, the substring consists only of the "start" character of invoking source string. If the third argument is present, the substring begins at position "start" and ends at position "end". If no argument is given, the function returns the first character of the string. If "end" specifies a position beyond the end of source string, the substring ends at the end of source string. String position counting begins at one (not zero).

10.13.9 Find Function

```
int mstring::Find(const char *, int=1)  
int mstring::Find(mstring, int=1)
```

Find() searches the first argument for an occurrence of the second argument. If one is found, the value returned is one greater than the end position of the second argument in the first argument. If "start" is specified, the search begins at position "start" in argument 1. If the second argument is not found, the value returned is 0. String position counting begins at position one.

```
mstring x;  
x="ABCDEF";  
x.Extract(2) yields "B"  
x.Extract(3,5) yields "CDE"
```

10.13.10 Horolog Function

mstring Horolog()

Returns an **mstring** of the form "x,y" where x is the number of days since December 31, 1984 and y is the number of seconds since midnight.

10.13.11 Justify Function

mstring mstring::Justify(int,int=-1)

Justify() right justifies the invoking mstring in an mstring field whose length is given by the first argument. If the second argument is present and a positive integer, the invoking mstring is right justified in a field whose length is given by the first argument with "precision" decimal places. The two argument form imposes a numeric interpretation upon the first argument.

```
x="39";  
x.Justify(3) yields " 39"  
  
x="TEST";  
x.Justify(7) yields " TEST"  
  
x="39";  
x.Justify(4,1) yields "39.0"
```

10.13.12 Length Function

```
int mstring::Length()  
int mstring::Length(mstring pattern_string)  
int mstring::Length(char * pattern_string)
```

The function returns the string length of the invoking mstring.

10.13.13 mcvf Function

mstring mcvf(arg)

Converts the arg to **mstring**. Arg may be int, char *, float long or double.

10.13.14 Pattern Function

```
int mstring::Pattern(mstring &)  
int mstring::Pattern(const char *)
```

Evaluates the invoking source string according to the pattern_string and returns 0 (does not match) or 1 (does match). *Pattern_string* rules are as shown below but you must remember to place a backslash before quotes in the pattern string (as per usual C++ rules). The pattern match function is used to determine if a string conforms to a certain pattern. Pattern match operations are converted to Perl Compatible Regular Expressions and are executed by functions in the PCRE library which must be present. You may access the

PCRE directly, using Perl expression format with the "*perl_pm(string, pattern, 1, svPtr)*" function discussed in Appendix D. The basic Mumps pattern codes are shown in Figure 48.

The Mumps pattern codes are:

- A for the entire upper and lower case alphabet.
- C for the 33 control characters.
- E for any of the 128 ASCII characters.
- L for the 26 lower case letters.
- N for the numerics
- P for the 33 punctuation characters.
- U for the 26 upper case characters.
- A literal string.

Figure 48 Mumps Pattern Codes

A pattern code is made up of one or more of the those shown in Figure 48, each preceded by a count specifier. The count specifier indicates how many of the named item must be present. Alternatively, an indefinite specifier - a decimal point - may be used to indicate any count (including zero). For example:

```
mstring A;  
A="123-45-6789";  
if (A.Pattern(command("3N1"- "2N1"- "4N"))) cout << "OK" << endl;  
A="JONES, J. L.";  
if (A.Pattern(command(".A1", ".A") )) cout << "OK" << endl;
```

Full pattern matching syntax, including support for alternation, are supported as described in Appendix D of the Compiler manual. The macro "command()" will handle the required backslash escape characters required before quote marks.

10.13.15 Perl Function

```
int Perl(mstring string, mstring regex)  
int Perl(mstring string, char * regex)
```

The regular expression in the null terminated character array or **mstring** given by *regex* is applied to the **mstring** *string*. If the pattern match succeeds, true (1) is returned, false (0) otherwise and *\$test* is set accordingly. This macro also sets variables in the run-time symbol table. See *SymGet()* and *SymPut()* for details on accessing the symbol table. See Appendix D for examples of using this function.

10.13.16 Piece Function

```
mstring mstring::Piece(const char *, int, int=-1)  
mstring mstring::Piece(mstring &, int, int=-1)
```

The *Piece()* function returns a substring of the invoking mstring delimited by the instances of the first argument. The substring returned in the two argument case is that substring of the invoking mstring that lies between the "start" minus one and "start" occurrence of the first argument. In the three argument form, the string returned is that substring of the invoking mstring delimited by the "start" minus one instance of the first argument and the end'th instance of the first argument. If only two arguments are given, end is assumed to be start. For example:

```
x="aaa.bbb.ccc.eee.fff";  
cout << x.Piece(".",1) << endl; // writes aaa  
cout << x.Piece(".",2) << endl; // writes bbb  
cout << x.Piece(".",5) << endl; // writes fff  
cout << x.Piece(".",4,5) << endl; // writes eee.fff
```

Global arrays may be used in any argument position but only one instance of the same global may appear (see note in [Accessing global arrays](#)) section.

10.13.17 ReadLine Function

```
bool mstring::ReadLine(FILE *)
bool mstring::ReadLine(istream &)
```

The next line from the file designated by "unit" is read into the invoking object of mstring. Carriage-returns and line-feeds are removed. The maximum length line that can be read is STR_MAX-1. Returns 'true' if the operation succeeded, 'false' otherwise or if end of file.

10.13.18 replace Function

```
int mstring::replace(mstring pattern, mstring replacement)
```

Replaces the string matching pattern with replacement. Returns 1 if successful, 0 if there was no match and less than -1 on error (See PCRE documentation for pcre_exec()). Throws: PatternException.

10.13.19 ScanAlnum Function

```
mstring ScanAlnum(FILE *, int min=3, int max=25)
mstring ScanAlnum(istream, int min=3, int max=25)
```

Returns the next token from the input file with all punctuation removed. Returns empty string on end of file. If min and/or max are provided, only words whose length are less than min and greater than max are discarded. The default values for these parameters are 3 and 25, respectively. Use stdin for file to scan standard input.

10.13.20 shred Function

```
mstring Shred(mstring str, int size)
```

The Shred() function shreds the input string *str* into fragments of length *size* upon successive calls. The function returns a string of length zero when there are no more fragments of length *size* remaining (thus, short fragments at the end of a string are not returned). Shred() 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 49 contains an example.

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

int main() {

    char x[] = "abcdefghijklmnopqrstuvwxy";
    char *p;

    while(1) {
        p = Shred(x, 5);
        if (strlen(p) == 0) break;
        cout << p << endl;
    }
    return 0;
}

yields:

abcde
```

```
fg hij
kl mno
pq rst
uv wxy
```

Figure 49 Shred Function

10.13.21 ShredQuery Function

mstring ShredQuery(mstring str, int size)

The *ShredQuery()* function shreds *size* shifted copies of the input string *str* into fragments of length *size* upon successive calls. That is, the function first returns all the *size* fragments of the string in the same manner as *Shred()*. However, it then shifts the starting point of the input string to the right by one and returns all the *size* length fragments relative to the shifted starting point. It repeats this process a total of *size* times.

The function returns a string of length zero when there are no more fragments of length *size* remaining (thus, short fragments at the end of a string are not returned). *ShredQuery()* initially 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. See Figure 50.

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

int main() {

    char x[] = "abcdefghijklmnopqrstuvwxy";
    char *p;

    while(1) {
        p = ShredQuery(x, 5);
        if (strlen(p) == 0) break;
        cout << p << endl;
    }
    return 0;
}
```

Yields:

```
abcde
fg hij
kl mno
pq rst
uv wxy
```

```
bcdef
ghijk
lmnop
qrstu
```

```
cdefg
hijkl
mnopq
rstuv
```

```
defgh
ijklm
nopqr
stuvw
```

```
efghi
```

jklmn opqrs tuvwX

Figure 50 ShredQuery

10.13.22 Stem Function

mstring stem(**mstring** & word)

Returns the original word or the English linguistic root stem of the word, if one can be found.

10.13.23 SymGet SymPut Functions

mstring SymGet(**mstring** name)
mstring SymGet(**char** * name)
mstring SymGet(**global** name)
mstring SymPut(name, value)

These functions retrieve and store values from/to the run-time symbol table. In all, *name* is a string containing the name of the variable and *value* is the value to be stored. The *SymPut()* functions return true if successful. A MumpsSymbolTableException exception is raised if *SymGet()* fails. If *SymPut()* fails, the program terminates (out of memory). For SymPut(), 'name' and 'value' may be any combination of **mstring**, **global** or null terminated **char**.

10.13.24 s_str Function

string **mstring::s_str()**

Returns a **string** copy of the contents of the invoking **mstring** object.

10.13.25 Token Function

mstring Token()
mstring TokenInit(**mstring**)

Token() returns the next word token from the input string. Initially a line of text is passed to *TokenInit()*. For each subsequent call to *Token()*, the next lexical token from the original string is returned. Upper case letters are converted to lower case letters. When there are no more words, the empty string is returned. After the the empty string is returned (or when initially called), the function will accept and store a new line of text.

10.13.26 Translate Function

mstring **mstring::Translate(mstring)**
mstring **mstring::Translate(mstring, mstring)**

If only one **mstring** argument is given, characters appearing in the argument **mstring** are removed from the invoking **mstring**.

If two argument **mstrings** appear and the first and second argument **mstring** are of the same length, characters from the invoking **mstring** that appear in the first argument **mstring** are replaced by their counterparts from the second argument **mstring**.

If the first argument **mstring** is longer than the second argument **mstring**, the characters from the first argument **mstring** which have no counterpart in the second argument **mstring** are removed.

A "counterpart" is a character equally offset in the second argument **mstring** to the character in the first argument **mstring**.

10.14 Basic mstring Example

Figure 51 gives some examples of the data type **mstring**. In the Mumps language there is one basic data type: string. All operations, including arithmetic calculations result in string values.

The **mstring** data type imitates the string data type in Mumps. It can be used as a traditional string or in mathematics.

In Figure 51 we see the **mstring** variables *a*, *b*, and *c* being used as traditional strings with the MDH concatenation operator (`||`) to form the string *hello world*. The **mstring** variable *a* is then used as a numeric counter to print zero through nine. It is then used to as a numeric index to a global array (*x(a)*) and finally in several expressions accessing the global array.

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

global x("x");

int main() {
    mstring a, b, c;

    a = "hello ";
    b = "world";

    cout << (a || b) << endl;    // concatenation
                                // prints "hello world"
    for (a = 0; a < 10; a++)
        cout << a << endl;      // prints 0 thru 9

    for (a = 0; a < 10; a++)
        x(a) = a;                // sets global array elements

    a = "";

    while (1) {
        a = x(a).Order(1);
        if (a == "") break;
        cout << a << endl;      // prints 0 thru 9
    }

    cout << x(a).Data() << endl; // prints 1

    c = "123 elm street";
    c = c + 1;
    cout << c << endl;          // prints 124

    return EXIT_SUCCESS;
}
```

Figure 51 mstring Examples

Note: the code "**(a || b)**" in the cout expression is parenthesized. If not parenthesized, the C++ compiler precedence will result in an error since the precedence of `<<` is greater than `||`.

10.15 Detailed mstring Examples

10.15.1 Assignment from Other Data Types

Variables of type **mstring** may be assigned values from variables or constants of types **char ***, **string**, **global**, **mstring**, **float**, **int**, **long**, or **double** as shown in Figure 52.


```

#include <mumpsc/libmpscpp.h>

int main() {
    // examples of assignment to mstring

    mstring x;

    x = 10;          cout << x << endl;
    x = 10.99;       cout << x << endl;
    x = "test";      cout << x << endl;

    string a1="abcdef";
    float  a2=99.9;
    double a3=99.8;
    int    a4=99;
    short  a5=98;
    char   a6[]="abcdef";
    global a7("a7"); a7("1")=99;

    x = a1;          cout << x << endl;
    x = a2;          cout << x << endl;
    x = a3;          cout << x << endl;
    x = a4;          cout << x << endl;
    x = a5;          cout << x << endl;
    x = a6;          cout << x << endl;
    x = a7("1");     cout << x << endl;

    GlobalClose;

    return EXIT_SUCCESS;
}

```

which writes:

```

10
10.99
test
abcdef
99.9
99.8
99
98
abcdef
99

```

Figure 52 mstring Assignment Examples

10.15.2 Arithmetic Operations on mstring

Figure 53 gives examples of arithmetic operations of **mstring**.

```

#include <mumpsc /libmpscpp.h>

int main() {

    // examples mstring operators.

```

```

mstring x;
mstring y;
mstring z;

y = 1;

x = 10;

cout << "expect 11 " << x + 1 << endl;
cout << "expect 9 " << x - 1 << endl;
cout << "expect 20 " << x * 2 << endl;
cout << "expect 5 " << x / 2 << endl;
cout << "expect 1 " << x % 3 << endl;
cout << "expect 11 " << x + y << endl;

cout << "-----\n";

x = 10; x = x + 1;      cout << "expect 11 " << x << endl;
x = 10; x = x - 1;      cout << "expect 9 " << x << endl;
x = 10; x = x * 2;      cout << "expect 20 " << x << endl;
x = 10; x = x / 2;      cout << "expect 5 " << x << endl;
x = 10; x = x % 3;      cout << "expect 1 " << x << endl;
x = 10; x = x + y;      cout << "expect 11 " << x << endl;
x = 10; x = y + x;      cout << "expect 11 " << x << endl;

cout << "-----\n";

x = 10; x += 1;      cout << "expect 11 " << x << endl;
x = 10; x -= 1;      cout << "expect 9 " << x << endl;
x = 10; x *= 2;      cout << "expect 20 " << x << endl;
x = 10; x /= 2;      cout << "expect 5 " << x << endl;
x = 10; x %= 3;      cout << "expect 1 " << x << endl;

cout << "-----\n";

x=10; x += y;      cout << "expect 11 " << x << endl;
x=10; x -= y;      cout << "expect 9 " << x << endl;
x=10; x *= y;      cout << "expect 10 " << x << endl;
x=10; x /= y;      cout << "expect 10 " << x << endl;
x=10; x %= y;      cout << "expect 0 " << x << endl;

cout << "-----\n";

x = 10; x = 1 + x + y;  cout << "expect 12 " << x << endl;
x = 10; x = 1 - x + y;  cout << "expect - 8 " << x << endl;
x = 10; x = 1 * x + y;  cout << "expect 11 " << x << endl;
x = 10; x = 1 / x + y;  cout << "expect 1.1 " << x << endl;

cout << "-----\n";

x = 10; x = 1 + ( x + y ); cout << "expect 12 " << x << endl;
x = 10; x = (x + y) + ( x + y ); cout << "expect 22 " << x << endl;

x = 10; cout << "expect 11 " << ++x ;
      cout << " expect 11 " << x << endl;
x = 10; cout << "expect 10 " << x++ ;
      cout << " expect 11 " << x << endl;
x = 10; cout << "expect 9 " << --x ;
      cout << " expect 9 " << x << endl;
x = 10; cout << "expect 10 " << x-- ;
      cout << " expect 9 " << x << endl;

```

```

cout << "-----\n";

x = 10; cout << "expect yes "; if ( x == 10 ) cout << "yes\n";
x = 10; cout << "expect yes "; if ( x >= 10 ) cout << "yes\n";
x = 10; cout << "expect yes "; if ( x <= 10 ) cout << "yes\n";
x = 10; cout << "expect yes "; if ( x >= 9 ) cout << "yes\n";
x = 10; cout << "expect yes "; if ( x > 9 ) cout << "yes\n";

x = 10; cout << "expect no ";
if ( x != 10 ) cout << "yes\n"; else cout << "no\n";

x = 10; cout << "expect no ";
if ( x > 10 ) cout << "yes\n"; else cout << "no\n";

x = 10; cout << "expect no ";
if ( x < 10 ) cout << "yes\n"; else cout << "no\n";

x = 10; cout << "expect no ";
if ( x <= 9 ) cout << "yes\n"; else cout << "no\n";

x = 10; cout << "expect no ";
if ( x < 9 ) cout << "yes\n"; else cout << "no\n";

cout << "-----\n";

x = "test message";
cout << "expect \"test message\" " << x << endl;

cin >> x;
cout << "expect what you typed " << x << endl;

x = "test ";
y = "message";

z = x || y;
cout << "expect \"test message\" " << z << endl;

x = x || x || x;
cout << "expect \"test test test\" " << x << endl;

x = "test";
z = y = (x || " test");
cout << "expect \"test test test test\" " << y << " " << z << endl;

x = "test";
z = y = x = x || " test";
cout << "expect \"test test test test\" " << y << " " << z << endl;
cout << "expect \"test test\" " << x << endl;

GlobalClose;

return EXIT_SUCCESS;

}

```

Figure 53 mstring Arithmetic Operations

10.15.3 Miscellaneous mstring Rules

1. Objects of **mstring** may be not initialized in declaration statements.

2. Objects of type **mstring** may participate in add(+, +=), subtract(-, -=), multiply(*, *=), divide(/, /=), modulo (% , %=) (integers values only) pre/post increment/decrement (++/--), and concatenation (| |) operations. The mode of the operation will depend on the mode of the other operand.
3. Objects of type **mstring** may participate in relational expressions >, >=, <, <=. The mode of comparison will depend on the mode of the other operand.
4. Objects of type **mstring** may participate in equality expressions == and !=. The mode of the comparison will depend on the mode of the other operand.
5. Objects of type **mstring** may participate in input and output stream operations >> and <<.
6. Objects of type **mstring** may not be assigned directly to ASCII null terminated string (**char ***) or **string**.
7. Objects of type **mstring** may be declared as arrays or allocated/freed by the **new/delete** operators. Only numeric subscripts permitted at this time.
8. If an object of type **mstring** is to be used in connection with the interpreter, it must be declared with a string giving its name in the run time symbol table. For example:

```
mstring x("x");
```

If this is done, variables in the C++ program are linked to variables of the same name in the Mumps interpreter. That is, values from variables in the C++ program are known by the same name to interpreted programs invoked by the C++ program. Changes made to these variables in the interpreter are changes to the variables in the C++ program. Variable names selected must be compatible with the interpreter's naming conventions.

10.16 Class global

10.16.1 Assignment Operations on global Arrays

Assignments to global arrays may be accomplished the assignment operator (=).

When you access a global array, the access may result in the thrown error exceptions *GlobalNotFoundException* and/or *ConversionException*. The first can occur in any context that attempts to retrieve data from a global array where none exists. The second occurs if you attempt to convert the contents of a global to a numeric type where the contents of the global are not valid data for the conversion.

If uncaught, both exceptions will result in program termination. Both exceptions may be caught, however, with code such as shown in Figure 54.

```
#include <mumpsc /libmpscpp.h>
global a("a");

int main() {

    long i;
    a.Kill();

    a("1") = "now is the time";

    cout << "expect error message" << endl;

    try {
        i = a("1");
    }

    catch ( ConversionException ce) {
        cout << ce.what() << endl;
    }

    cout << "expect error message" << endl;

    try {
        i = a("22");
    }

    catch (GlobalNotFoundException nf) {
        cout << nf.what() << endl;
    }

    GlobalClose;

    return 0;
}
```

Figure 54 Exceptions

You may assign data of the following types directly to global arrays: **char ***, **int**, **string**, **mstring**, **double**, **global**, **unsigned int**, **float**, **short**, **unsigned short**, **long**, and **unsigned long**.

You may assign global arrays directly to variables of the following types: **int**, **mstring**, **double**, **global**, **unsigned int**, **float**, **short**, **unsigned short**, **long**, and **unsigned long**.

10.16.2 Arithmetic Operations on Global Arrays

The operations of add, subtract, multiply, divide, pre/post increment and pre/post decrement are defined (overloaded) for global variables. The operations are defined for **mstring**, **short**, **unsigned short**, **int**, **unsigned int**, **long**, **unsigned long**, **float** and **double**. Note: the contents of the global array node must be compatible with the dominant data type of the operation. If the contents of a global are not compatible with the operation (example, incrementing a string of text), the value of the global will be interpreted as zero. See Figure 55 for examples.

```
#include <mumpsc /libmusc.cpp.h>

global gbl("gbl");

int main () {

    int i, j = 10;
    string a = "10", b = "20", c = "30";

    gbl.Kill();

    gbl(a, b, c) = 10;

    i = gbl(a, b, c) + 20;
    cout << "expect 20 " << i << endl; // prints 30

    i = 20 + gbl(a, b, c);
    cout << "expect 30 " << i << endl; // prints 30

    i = gbl(a, b, c) / j;
    cout << "expect 1 " << i << endl; //prints 1

    i = gbl(a, b, c) * 2;
    cout << "expect 20 " << i << endl; // prints 20

    gbl(a, b, c) ++;
    cout << "expect 11 " << gbl(a, b, c) << endl; // prints 11

    gbl(a, b, c) --;
    cout << "expect 10 " << gbl(a, b, c) << endl; // prints 10

    i = ++ gbl(a, b, c);
    cout << "expect 11 11 " << i << " " << gbl(a, b, c) << endl; // prints
    11

    i = gbl(a, b, c) ++;
    cout << "expect 11 12 " << i << " " << gbl(a, b, c) << endl; // prints
    11 12

    gbl(a, b, c) += 10;
    cout << "expect 22 " << gbl(a, b, c) << endl; // prints 22

    gbl(a, b, c) -= 10;
    cout << "expect 12 " << gbl(a, b, c) << endl; // prints 12

    gbl(a, b, c) *= 2;
    cout << "expect 24 " << gbl(a, b, c) << endl; //prints 24

    gbl(a, b, c) /= 2;
    cout << "expect 12 " << gbl(a, b, c) << endl; // prints 12
```

```
GlobalClose;
return 0;
}
```

Figure 55 Example Global Array Arithmetic

10.16.3 Operations on global

Figure 56 shows the current list of operator overloads for class **global**. Additional overloads will be added in time.

Assignment

```
global & operator=(const char *);
global & operator=(int);
global & operator=(double);
global & operator=(string);
global & operator=(global);
global & operator=(unsigned int);
global & operator=(float);
global & operator=(short);
global & operator=(unsigned short);
global & operator=(long);
global & operator=(unsigned long);
global & operator=(mstring);
```

Addition

int operator+(int);	friend int operator+(int,global);
unsigned int operator+(unsigned int);	friend unsigned int operator+(unsigned int,global);
long operator+(long);	friend unsigned long operator+(unsigned long,global);
unsigned long operator+(unsigned long);	friend long operator+(long,global);
short operator+(short);	friend short operator+(short,global);
float operator+(float);	friend unsigned short operator+(unsigned short,global);
unsigned short operator+(unsigned short);	friend float operator+(float,global);
double operator+(double);	friend double operator+(double,global);
double operator+(global);	
int operator+=(int);	friend int operator+=(int &,global);
unsigned int operator+=(unsigned int);	friend unsigned int operator+=(unsigned int,global);
short operator+=(short);	friend short operator+=(short,global);
unsigned short operator+=(unsigned short);	friend unsigned short operator+=(unsigned short,global);
long operator+=(long);	friend long operator+=(long,global);
unsigned long operator+=(unsigned long);	friend unsigned long operator+=(unsigned long,global);
float operator+=(float);	friend float operator+=(float,global);
double operator+=(double);	friend double operator+=(double,global);

Subtraction

int operator-(int);	friend int operator-(int,global);
unsigned int operator-(unsigned int);	friend unsigned int operator-(unsigned int,global);
long operator-(long);	friend unsigned long operator-(unsigned long,global);
unsigned long operator-(unsigned long);	friend short operator-(short,global);
short operator-(short);	friend long operator-(long,global);
float operator-(float);	friend float operator-(float,global);
double operator-(double);	friend double operator-(double,global);
double operator-(global);	friend unsigned short operator-(unsigned short,global);
unsigned short operator-(unsigned short);	

	<pre> short,global); int operator-=(int); unsigned int operator-=(unsigned int); short operator-=(short); unsigned short operator-=(unsigned short); long operator-=(long); unsigned long operator-=(unsigned long); float operator-=(float); double operator-=(double); </pre>	<pre> friend int operator-=(int &,global); friend unsigned int operator-=(unsigned int,global); friend short operator-=(short,global); friend unsigned short operator-=(unsigned short,global); friend long operator-=(long,global); friend unsigned long operator-=(unsigned long,global); friend float operator-=(float,global); friend double operator-=(double,global); </pre>
Multiplication	<pre> int operator*(int); unsigned int operator*(unsigned int); long operator*(long); unsigned long operator*(unsigned long); short operator*(short); float operator*(float); double operator*(double); double operator*(global); unsigned short operator*(unsigned short); </pre>	<pre> friend int operator*(int,global); friend unsigned int operator*(unsigned int,global); friend long operator*(long,global); friend unsigned long operator*(unsigned long,global); friend short operator*(short,global); friend float operator*(float,global); friend double operator*(double,global); friend unsigned short operator*(unsigned short,global); </pre>
	<pre> int operator*=(int); unsigned int operator*=(unsigned int); short operator*=(short); unsigned short operator*=(unsigned short); long operator*=(long); unsigned long operator*=(unsigned long); float operator*=(float); double operator*=(double); </pre>	<pre> friend int operator*=(int &,global); friend unsigned int operator*=(unsigned int,global); friend short operator*=(short,global); friend unsigned short operator*=(unsigned short,global); friend long operator*=(long,global); friend unsigned long operator*=(unsigned long,global); friend float operator*=(float,global); friend double operator*=(double,global); </pre>
Division	<pre> int operator/(int); unsigned int operator/(unsigned int); long operator/(long); unsigned long operator/(unsigned long); short operator/(short); unsigned short operator/(unsigned short); float operator/(float); double operator/(double); double operator/(global); </pre>	<pre> friend int operator/(int,global); friend unsigned int operator/(unsigned int,global); friend long operator/(long,global); friend unsigned long operator/(unsigned long,global); friend short operator/(short,global); friend unsigned short operator/(unsigned short,global); friend float operator/(float,global); friend double operator/(double,global); </pre>
	<pre> int operator/=(int); unsigned int operator/=(unsigned int); short operator/=(short); unsigned short operator/=(unsigned short); long operator/=(long); unsigned long operator/=(unsigned long); float operator/=(float); double operator/=(double); </pre>	<pre> friend int operator/=(int &,global); friend unsigned int operator/=(unsigned int,global); friend short operator/=(short,global); friend unsigned short operator/=(unsigned short,global); friend long operator/=(long,global); friend unsigned long operator/=(unsigned </pre>

	<pre> long,global); friend float operator/=(float,global); friend double operator/=(double,global); </pre>
Increment/Decrement	
<pre> double operator++(); double operator--(); double operator++(int); double operator--(int); </pre>	
Unary	
<pre> mstring operator+() ; // unary plus mstring operator-() ; // unary minus </pre>	
Relational	
<pre> int operator>(global); int operator>(int); int operator>(unsigned int); int operator>(long); int operator>(unsigned long); int operator>(short); int operator>(unsigned short); int operator>(float); int operator>(double); int operator>(char *); int operator>(string); </pre>	<pre> friend int operator>(int,global); friend int operator>(unsigned int,global); friend int operator>(long,global); friend int operator>(unsigned long,global); friend int operator>(short,global); friend int operator>(unsigned short,global); friend int operator>(float,global); friend int operator>(double,global); friend int operator>(char *,global); friend int operator>(string,global); </pre>
<pre> int operator<(global); int operator<(int); int operator<(unsigned int); int operator<(long); int operator<(unsigned long); int operator<(short); int operator<(unsigned short); int operator<(float); int operator<(double); int operator<(char *); int operator<(string); int operator<(mstring); </pre>	<pre> friend int operator<(int,global); friend int operator<(unsigned int,global); friend int operator<(long,global); friend int operator<(unsigned long,global); friend int operator<(short,global); friend int operator<(unsigned short,global); friend int operator<(float,global); friend int operator<(double,global); friend int operator<(char *,global); friend int operator<(string,global); friend int operator<(mstring,global); </pre>
<pre> int operator<=(global); int operator<=(int); int operator<=(unsigned int); int operator<=(long); int operator<=(unsigned long); int operator<=(short); int operator<=(unsigned short); int operator<=(float); int operator<=(double); int operator<=(char *); int operator<=(string); </pre>	<pre> friend int operator<=(int,global); friend int operator<=(unsigned int,global); friend int operator<=(long,global); friend int operator<=(unsigned long,global); friend int operator<=(short,global); friend int operator<=(unsigned short,global); friend int operator<=(float,global); friend int operator<=(double,global); friend int operator<=(char *,global); friend int operator<=(string,global); </pre>
<pre> int operator>=(global); int operator>=(int); int operator>=(unsigned int); int operator>=(long); int operator>=(unsigned long); </pre>	<pre> friend int operator>=(int,global); friend int operator>=(unsigned int,global); friend int operator>=(long,global); friend int operator>=(unsigned </pre>

<code>int operator>=(short);</code>	<code>long,global);</code>
<code>int operator>=(unsigned short);</code>	<code>friend int operator>=(short,global);</code>
<code>int operator>=(float);</code>	<code>friend int operator>=(unsigned</code>
<code>int operator>=(double);</code>	<code>short,global);</code>
<code>int operator>=(char *);</code>	<code>friend int operator>=(float,global);</code>
<code>int operator>=(string);</code>	<code>friend int operator>=(double,global);</code>
	<code>friend int operator>=(char *,global);</code>
	<code>friend int operator>=(string,global);</code>
<code>int operator==(global);</code>	<code>friend int operator==(int,global);</code>
<code>int operator==(int);</code>	<code>friend int operator==(unsigned</code>
<code>int operator==(unsigned int);</code>	<code>int,global);</code>
<code>int operator==(long);</code>	<code>friend int operator==(long,global);</code>
<code>int operator==(unsigned long);</code>	<code>friend int operator==(unsigned</code>
<code>int operator==(short);</code>	<code>long,global);</code>
<code>int operator==(unsigned short);</code>	<code>friend int operator==(short,global);</code>
<code>int operator==(float);</code>	<code>friend int operator==(unsigned</code>
<code>int operator==(double);</code>	<code>short,global);</code>
<code>int operator==(char *);</code>	<code>friend int operator==(float,global);</code>
<code>int operator==(string);</code>	<code>friend int operator==(double,global);</code>
	<code>friend int operator==(char *,global);</code>
	<code>friend int operator==(string,global);</code>
<code>int operator!=(global);</code>	<code>friend int operator!=(int,global);</code>
<code>int operator!=(int);</code>	<code>friend int operator!=(unsigned</code>
<code>int operator!=(unsigned int);</code>	<code>int,global);</code>
<code>int operator!=(long);</code>	<code>friend int operator!=(long,global);</code>
<code>int operator!=(unsigned long);</code>	<code>friend int operator!=(unsigned</code>
<code>int operator!=(short);</code>	<code>long,global);</code>
<code>int operator!=(unsigned short);</code>	<code>friend int operator!=(short,global);</code>
<code>int operator!=(float);</code>	<code>friend int operator!=(unsigned</code>
<code>int operator!=(double);</code>	<code>short,global);</code>
<code>int operator!=(char *);</code>	<code>friend int operator!=(float,global);</code>
<code>int operator!=(string);</code>	<code>friend int operator!=(double,global);</code>
	<code>friend int operator!=(char *,global);</code>
	<code>friend int operator!=(string,global);</code>

Casts

```
operator char*() ;
operator int();
operator unsigned int();
operator short();
operator unsigned short();
operator long();
operator unsigned long();
operator float();
operator double();
operator mstring();
```

Figure 56 Global Array Operator Overloads

10.16.4 Accessing the Value Stored in a Global Array Element

```
int global::Int();
double global::Double();
mstring global::Mstring();
char * global::Char(char * buf, int max);
```

The functions return the content of the invoking global array object converted to the named data type.

The **Char()** function is passed the address of a character array. The null-terminated character string contents of the global array element will be placed in the character array and the address of the array returned.

The *max* argument for **Char()** limits the length of the string returned.

If the global array element does not exist, the *GlobalNotFoundException* exception is thrown. If there is an error in converting the contents of the global to the named data type, a *ConversionException* is thrown. See Figure 57 for examples.

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

global t("t");

int main() {

    int a;
    float b;
    mstring c;
    mstring x;
    char d[100];

    t.Kill();

    x = 50; t(x) = 99;

    a = t(x).Int();
    cout << "expect 99 " << a << endl;

    b = t(x).Double();
    cout << "expect 99 " << b << endl;

    c = t(x).Mstring();
    cout << "expect 99 " << c << endl;

    t(x).Char(d,100);
    cout << "expect 99 " << d << endl;

    GlobalClose;
}
```

Figure 57 Accessing Global Array Data Example

10.16.5 Global Functions and Methods

10.16.5.1 Data()

int global::Data()

The function *Data()* returns an integer which indicates whether the global array node is defined. The value returned is 0 if the global array node is undefined, 1 if it is defined and has no descendants; 10 if it is defined but has no value stored at the node (but does have descendants); and 11 if it is defined and has descendants.

If a global array with no indices is passed to these functions, a value of "10" will be returned if the array exists and "0" if the array does not exist. For example:

Given:

```
global gbl("gbl");
global non("non");
```

```
gbl("1", "11") = "foo";
gbl("1", "11", "21") = "bar";
```

Then:

```
gbl("1").Data() // 10 - node exists, has no data, has children
gbl("1", "11").Data() // 11 - node exists, has data and has children
gbl("1", "11", "21").Data() // 1 - nodes exists, has data, no children
```

10.16.5.2 TreePrint()

```
void global::TreePrint([int indt [, const char indtchr]]);
```

The invoking object is printed as an indented tree. If one argument is present (*indt*), it is the amount of indentation. If the second argument is present (*indtchr*) it is the character used in the indentation. The default indentation character is blank and the default amount of indentation is one. See Figures 58 and 59 for examples.

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

global d("d");

int main() {
    mstring a,b,c;

    for (int i = 1; i < 6; i++)
        for (int j = 1; j < 6; j++)
            for (int k = 1; k < 6; k++) {
                a = mcvT(i);
                b = mcvT(j);
                c = mcvT(k);
                d(a) = rand() % 100;
                d(a,b) = rand() % 100;
                d(a,b,c) = rand() % 100;
            }

    d().TreePrint(1, '.');

    GlobalClose;

    return 0;
}
```

Figure 58 TreePrint

Yields

1=82	2=68	3=72	4=66	5=79
..1=59	..1=54	..1=28	..1=48	..1=72
..1=77	..1=64	..1=96	..1=39	..1=76
..2=35	..2=87	..2=45	..2=69	..2=7
..3=49	..3=78	..3=21	..3=64	..3=79
..4=27	..4=3	..4=88	..4=55	..4=12
..5=63	..5=99	..5=41	..5=11	..5=59
.2=67	.2=78	.2=59	.2=30	.2=21
..1=26	..1=76	..1=0	..1=99	..1=10
..2=11	..2=12	..2=24	..2=68	..2=6
..3=29	..3=94	..3=56	..3=11	..3=72
..4=62	..4=70	..4=27	..4=1	..4=19
..5=35	..5=67	..5=36	..5=78	..5=4
.3=19	.3=44	.3=93	.3=62	.3=69
..1=22	..1=2	..1=37	..1=36	..1=40
..2=67	..2=52	..2=7	..2=22	..2=28
..3=11	..3=80	..3=58	..3=16	..3=84
..4=73	..4=65	..4=37	..4=24	..4=24
..5=84	..5=19	..5=18	..5=24	..5=96
.4=96	.4=53	.4=4	.4=94	.4=98
..1=24	..1=31	..1=11	..1=52	..1=84
..2=13	..2=71	..2=76	..2=50	..2=72
..3=80	..3=9	..3=63	..3=73	..3=85
..4=62	..4=56	..4=6	..4=30	..4=40
..5=81	..5=86	..5=18	..5=60	..5=13
.5=45	.5=8	.5=25	.5=84	.5=69
..1=84	..1=83	..1=69	..1=81	..1=24
..2=5	..2=28	..2=96	..2=59	..2=81
..3=13	..3=29	..3=70	..3=68	..3=32
..4=95	..4=70	..4=99	..4=26	..4=4
..5=14	..5=15	..5=44	..5=40	..5=73

Figure 59 TreePrint Output

10.16.5.3 UnLock

```
int global::UnLock()
```

UnLock() removes a lock from the designated node.

10.16.5.4 Count

```
long global::Count()
```

Returns the number of data bearing nodes beneath the given global array reference. See Figure 60 for example.

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

global A("A");

int main() {
    mstring i, j;
    for (i = 1; i < 11; i++)
        for (j = 1; j < 11; j++) {
            A(i,j) = 5;
        }
    cout << "Full count: " << A().Count() << endl;
    cout << "A row count: " << A("5").Count() << endl;
    return EXIT_SUCCESS;
}
```

```
}
```

Yields

```
Full count: 100
A row count: 10
```

Figure 60 Count Example

10.16.5.5 GlobalGet(), GlobalData(), GlobalSet()

```
mstring GlobalGet (mstring global_ref)
char * GlobalGet (char * global_ref)

mstring GlobalOrder (mstring global_ref, int direction)
char * GlobalOrder (char * global_ref, int direction)

int GlobalData (mstring global_ref)
int GlobalData (char * global_ref)

int GlobalSet (mstring global_ref, mstring source)
int GlobalSet (mstring global_ref, char * source)
int GlobalSet (char * global_ref, mstring source)
```

These functions use the interpreter. These functions are used to permit run time construction and access to global arrays. In both cases *global_ref* is a string containing a global array reference. This string can be dynamically constructed at run time or may be read from a file or another global. Note: as this facility uses the interpreter, global array references must be preceded by the circumflex character (^).

In the case of the *GlobalGet()* functions, the string global array reference is interpreted and the value stored at the reference returned. If the reference is invalid or no data is stored, the value returned is the empty string and *\$test* is set to false (zero). If a value is found, *\$test* is set to true and the value is returned.

GlobalOrder() gives the next or prior value of the last index of the global array reference depending upon if *direction* is 1 (next) or -1 (prior). *\$test* is set to 0 in the event of an error and 1 if there is no error. See *Order()*.

GlobalData() returns a number indicating if the node exists and has descendants (see *Data()*). *\$test* is set to 0 if there is an error, 1 otherwise.

In the case of the *GlobalSet()* functions, the second argument is a string of data to be stored at the global array reference. The runtime routines will interpret the *global_ref* and assign the *source* to it. The value returned is one if successful (*\$test* is set to 1), zero if not successful (*\$test* set to 0). Examples:

```
mstring a,b;
a = "^x(\"l\")";
b = "test string";
if (GlobalSet(a,b) != 0) cout << "error\n";
```

These functions can be used to allow a program to create a text string global array reference and then use the string to address the global. Note that the *target* must contain either quoted literals or variables previously instantiated to the interpreter environment (see *\$SymSet()* and *SymGet()*).

Generally speaking, these functions will be only used for dynamically constructed global array references. Most access to globals will be by overloaded shift or assignment operators.

10.16.5.6 double HitRatio(void)

Calculates the native global array processor cache hit ratio since the beginning of the program or the last call to *HitRatio()*. The native global array file processor, as opposed to the Berkeley Data Base, keeps track of how many file I/O requests are satisfied from data already in the file system's cache. This function gives the percentage of cache hits. It only works with the native global array processor.

10.16.5.7 Kill

```
void global::Kill()
```

This function deletes a node and all its descendants. Examples:

```
gbl().Kill();           // kill entire global array "gbl"  
gbl(a,b,c).Kill();     // kill stated node and all descendants
```

10.16.5.8 Length

```
int mstring::Length()  
int mstring::Length(char * pattern_string)  
int mstring::Length(mstring pattern_string)
```

The function returns the string length of the invoking **mstring**. For example:

```
x="ABC";  
cout << x.Length() << endl;  // writes 3  
  
x = "abcabcabcabc";  
cout << x.Length("abc") << endl;  // writes 5
```

If an argument is given, the function returns the number of non-overlapping occurrences of "pattern_string" in the source string plus 1.

10.16.5.9 Max

```
double global::Max()
```

Returns the maximum numeric value of the data bearing nodes beneath the given reference. Non-numeric values are treated as zeros. See Figure 61 for example.

```
#include <mumpsc/libmpscpp.h>  
global A("A");  
  
int main() {  
    mstring i, j;  
    for (i = 1; i < 11; i++)  
        for (j = 1; j < 11; j++) {  
            A(i, j) = rand()%1000;  
        }  
    cout << "Max value of all:      " << A().Max() << endl;  
    cout << "Max value of row 10:  " << A("10").Max() << endl;  
    return EXIT_SUCCESS;  
}
```

Yields:

```
Max value of all:      996  
Max value of row 10:  932
```

Figure 61 Max Example

10.16.5.10 Merge

```
int global::Merge(global)
```

Copies the first **global** and its descendants to the second **global**. The Merge() function copies from one array to another. Examples:

```
Xecute("for i=1:1:9 for j=1:1:9 set ^a(i,j)=i+j");  
c().Merge(a());           // copies all of ^a to ^c  
  
Xecute("for i=100:1:109 s ^b(i)=i");
```

```

b("103").Merge(a("3")); // copies ^a(3) to ^b(103) and children of
                        // ^a(3) to be children of ^b(103)

d("").Merge(a("3"));    // creates ^d=^a(3); ^d(1)=^a(3,1),...

```

10.16.5.11 Min

double global::Min()

Returns the minimum numeric value of the data bearing nodes beneath the given reference. Non-numeric values are treated as zeros. Example:

```

#include <mumpsc /libmpscpp.h>
global A("A");

int main() {
    mstring i, j;
    for (i = 1; i < 11; i++)
        for (j = 1; j < 11; j++) {
            A(i,j) = rand() % 1000;
        }
    cout << "Min value of all:      " << A().Min() << endl;
    cout << "Min value of row 10:   " << A("10").Min() << endl;
    return EXIT_SUCCESS;
}

```

Yields:

```

Min value of all:      11
Min value of row 10:   12

```

10.16.5.12 Multiply

void global::Multiply(global B,global C)

The invoking global is multiplied by *B* and the result is place in *C*. The number of columns of *A* must equal the number of rows of *B*. The resulting matrix *C* will have "n" rows and "m" columns where "n" is the number of rows of "A" and "m" is the number of columns of "B".

In all cases *C* will be deleted before the operation commences. The data stored at each node must be numeric. All calculations are performed in **double** precision arithmetic. Each matrix must be two dimensional. See Figure 62.

```

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

global d("d");
global e("e");
global f("f");

int main() {

    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());
PRINT("f","1");

return EXIT_SUCCESS;
}

```

Yields:

```

^f(1,1)=-8
^f(1,2)=-1
^f(1,3)=-1
^f(1,4)=14
^f(2,1)=11
^f(2,2)=-3
^f(2,3)=7
^f(2,4)=7
^f(3,1)=-24
^f(3,2)=4
^f(3,3)=-12
^f(3,4)=0

```

Figure 62 Multiply Example

10.16.5.13 Name

mstring global::Name()

Returns a null terminated pointer to array of characters containing of the **global** reference with all variables and expressions in the indices evaluated. See Figure 63.

```

#include <mumpsc/libmpscpp.h>
global a("a");

int main() {
    mstring b = "1", c = "2", d = "3";
    cout << a(b, c, d, c + d).Name() << endl;
    return EXIT_SUCCESS;
}

```

Yields:

```

a("1", "2", "3", "23")

```

Figure 63 Name Example

10.16.5.14 Order

mstring global::Order([int direction])

The *Order()* function gives the next ascending or descending value of the last index in a global array reference. The direction, ascending or descending, is given by either the name of the function or an integer "direction" which is either 1 - next ascending index, or -1 - next descending index. If 'direction' is omitted, ascending is assumed. See Figure 64.

given:

```
global test("test");
test("1") = "";
test("1", "10") = "";
test("1", "20") = "";
test("5", "1") = "";
test("5", "5") = "";
```

Then Order() will return the following values:

```
test().Order(1)           yields "1"
test("1", "").Order(1)    yields "10"
test("1", "10").Order(1)  yields 20
test("1", "20").Order(1)  yields "" (empty string)
test("1").Order(1)        yields "5"
test("5", "").Order(1)    yields "1"
test("5", "1").Order(1)   yields "2"
test("5", "2").Order(1)   yields "" (empty string)
test("5").Order(1)        yields "" (empty string)
```

Similarly, a direction code of -1 will reverse the process:

```
test().Order(-1)          yields 5
test("5").Order(-1)       yields "1"
test("1").Order(-1)       yields "" (empty string)
```

Figure 64 Order Example

Use the empty string ("") to get the initial value of an index. When there are no further values, the empty string is returned.

Note: all keys are stored in ASCII character collating order. This means that numeric keys are sorted alphabetically rather than numerically.

10.16.5.15 Avg

double global::Avg()

Returns the average of the values of data bearing nodes beneath the given global array reference.

Example:

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

global A("A");

int main() {
    mstring i,j;

    A.Kill();

    for (i = 0; i < 1000; i++)
        for (j = 1; j < 10; j++) {
            A(i, j) = j;
        }

    cout << A("100").Avg() << endl; // average of nodes below A("100")
    cout << A().Avg() << endl;      // average of all nodes

    GlobalClose;
```

```

    return 0;
}

```

Figure 65 Avg Example

The above prints 5.5 - the average value of numeric data bearing nodes beneath A("100"). If there are non-numeric data elements, they are treated as a zero values and contribute to the result.

The global array object must be specified with indices (*i.e.*, a parenthesized list must follow the name of the **global** array object. An empty list means the entire array.

10.16.5.16 Locks

```

void CleanLocks(void)
void CleanAllLocks(void)

```

"CleanLocks()" removes all locks for the current process. "CleanAllLocks()" removes all locks for all processes for which the current directory is the default directory. Locks are implemented by entries in a file named "Mumps.Locks" created and maintained in the current directory. This file must be read/write enabled for the current process. You may also delete all locks by removing this file. Locks are discussed elsewhere but, in brief, they are used to signal ownership of a portion of a global array. When a lock has been applied to a node, no other process may lock this node, any descendant node or any parent node. Locking does not actually prevent access, it merely marks a resource as locked.

```

int global::Lock()

```

Creates a lock on the named node. If successful, "\$test" will be true (1), false (0) otherwise. Returns a 1 if the lock succeeds and a 0 otherwise.

The "Lock()" function marks a portion of the data base for exclusive access for an individual user. The "UnLock()" frees prior locks (see below). The locks are stored in a file named "Mumps.Locks" which is opened for exclusive access by the locking/unlocking job. The contents of the file may be deleted to remove all locks. A lock does not actually prevent access to a global but merely marks it as locked. If another task attempts to place a lock on a locked node, the descendant of a locked node or a direct parent of a locked node, the lock attempt will fail. Examples:

```

if (gbl(a, b, c).Lock()) { ..... } // locks gbl(a, b, c) and all children;
if ($lock(gbl(a, b, c))) { ..... }

```

10.16.5.17 GlobalClose

This macro closes the global array files. The global arrays must be closed on exit or they will be corrupt. The macro causes the file system to flush all its buffers and cache and close the file system. Normally, a "GlobalClose" is executed automatically when your program ends except if your program is terminated by SIGKILL or SIGSTOP (which cannot be trapped). If your program is using a large memory based cache (cache's can be 1 GB or more, on some systems), there may be a noticeable delay in file system shutdown due to the time required to write the cache to disk.

10.16.5.18 Btree

```

int BTREE(int code, unsigned char * key, unsigned char * data)

```

BTREE() is a macro permitting direct access to the underlying btree system. The first argument, "code" is an integer indicating the operation to be performed (see below). The second argument is the key to be stored consisting of a null-terminated array printable ASCII characters. The length of the key should be no greater than one quarter of the btree block size whose default value is 8192 (*i.e.*, max key length is about 2048 bytes in the default case). The third argument is the data to be stored with the key. It is a null-terminated string of printable ASCII characters not greater than the system defined limit STR_MAX (defaults to 4096). An empty string is interpreted as no data to be stored. Note that the second and third arguments must be **unsigned**

char *. The macro returns an integer indicating success. It may also alter "key" or "data" to return values or for other purposes. The contents of "key" and "data" are not preserved across in invocation of **BTREE()**. Examples of using **BTREE()** are given in *mumpsc/doc/examples/btree*.

Permitted btree operations:

1. STORE - store a key and data value in the btree; returns zero if successful, non-zero otherwise:

```
unsigned char key[] = "test key";
unsigned char data[] = "test data";
if ( BTREE(STORE, key, data) == 0 ) cout << "stored" << endl;
else cout << "not stored" << endl;
```

2. RETRIEVE - retrieve data stored with a key; returns zero if successful, non-zero otherwise:

```
unsigned char key[] = "test key";
unsigned char data[STR_MAX];

if (BTREE(RETRIEVE, key, data) == 0)
    cout << "retrieved: " << data << endl;
else cout << "not retrieved." << endl;
```

3. CLOSE - close the btree data base; returns zero:

```
unsigned char key[] = "";
unsigned char data[] = "";
BTREE(CLOSE, key, data);
```

4. XNEXT/PREVIOUS - retrieve next ascending/descending key; returns one. Value of second and third arguments become the value of the next ascending/descending key. An initial value of the empty string for the second argument will retrieve the first/last key and the value of the second argument becomes the empty string when there are no more ascending/descending values. An initial value of the empty string for the second argument will retrieve the first/last key.

```
unsigned char key[] = "";
unsigned char data[STR_MAX];

printf("\nbegin retrieve...\n");
while(1) { // retrieve keys in ascending order
    i=BTREE(XNEXT, key, data);
    if (strlen( (char *) data) == 0) break;
    cout << key << endl;
}
```

10.16.5.19 Query functions

```
mstring Query(mstring ref)
mstring Query(char * ref)
```

```
int Qlength(mstring ref)
int Qlength(char * ref)
```

```
mstring Qsubscript(mstring ref, mstring index)
mstring Qsubscript(mstring ref, int index)
mstring Qsubscript(char * ref, int index)
```

Query() returns an **mstring** containing the next global array reference in the data base or the empty string.

Qlength() returns the number of subscripts in the global array reference.

Qsubscript() returns the index'th subscript of a global array reference.

Each of these functions operates on a text representation of a global array reference. See also the *Name()* function. The following example makes use of the MeSH subject headings (National Library of Medicine). The MeSH global array was constructed with statements such as shown in Figure 66.

```
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")="Sacroccocygeal 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"
```

Figure 66 MeSH Headings¹⁰

The MeSH headings can be printed as shown in Figure 67.

¹⁰ The MeSH (Medical Subject Headings) is a controlled vocabulary hierarchical indexing and classification system developed by the National Library of Medicine (NLM). The MeSH codes are used to code medical records and literature as part of an ongoing research project at the NLM. The examples make use of the 2003 MeSH Tree Hierarchy. Newer versions, essentially similar to these, are available from NLM. Note: *for clinical purposes, the copy of the MeSH hierarchy used here is out of date and should not be employed for clinical decision making. It is used here purely as an example to illustrate a hierarchical index.* The 2003 MeSH file contains approximately 40,000 entries. Each line consists of text along with hierarchical codes describing the subject heading.

```

#include <mumpsc/libmpscpp.h>

//      CompiledMtree1.cpp Feb 28, 2007

int main() {

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

    x=Query("^mesh(0)");
    while (1) {
        x=Query(x);
        if (x=="") break;
        if (x.Piece("(",1)!="^mesh") break;
        i=Qlength(x);
        for (j=0; j<i; j++) cout << "  ";
        cout << Qsubscript(x,i) << " " << x.Eval() << endl;
    }
    return 0;
}

```

which 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 67 Query Functions Example

10.16.5.20 Similarity functions

```

double Sim1(global A, global B)
double Cosine(global A, global B)
double Jaccard(global A, global B)
double Dice(global A, global B)

```

The global arrays referenced by the invoking object and the passed object are compared and a similarity value is computed. The functions compute the similarities of the data bearing nodes beneath the global array references.

These are some commonly used similarity metrics. (see Salton, G; and McGill, M, *Introduction to Modern Information Retrieval*, McGraw Hill, 1983). See Figures 68 through 71.

```

#include <mumpsc/libmpscpp.h>

global A("A");
global B("B");

int main() {

    A("1","1","1") = 1;
    A("1","1","2") = 1;
    A("1","1","3") = 1;
    A("1","1","5") = 1;

    B("1","1","1") = 1;
    B("1","1","2") = 1;
    B("1","1","4") = 1;
    B("1","1","6") = 1;

    cout << Sim1(A("1","1"), B("1","1")) << endl;

    GlobalClose;

    return 0;
}

```

The above prints 2 since there are two nodes in common below the "1,1" levels.

Alternatively:

```

#include <mumpsc/libmpscpp.h>

global A("A");
global B("B");

int main() {

    A("1","1","1") = 2;
    A("1","1","2") = 1;

```

```

A("1","1","3") = 1;
A("1","1","5") = 1;

B("1","1","1") = 2;
B("1","1","2") = 1;
B("1","1","4") = 1;
B("1","1","6") = 1;

cout << Sim1(A("1","1"), B("1","1")) << endl;

GlobalClose;

return 0;
}

```

The above prints 5 since there are two nodes in common below the "1,1" levels but one of the set of nodes in common have a stored value of 2. ($2*2+1*1$)

Figure 68 Sim1 Example

```

#include <mumpsc /libmpscpp.h>

global A("A");
global B("B");

int main() {

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

    B("1") = 1;
    B("2") = 1;
    B("3") = 1;
    B("4") = 0;
    B("5") = 0;
    B("6") = 1;
    B("7") = 0;
    B("8") = 0;

    cout << Jaccard(A(), B()) << endl;

    GlobalClose;

    return 0;
}

```

prints 1

Figure 69 Jaccard Example

```

#include <mumpsc/libmpscpp.h>

global A("A");

```



```

global B("B");

int main() {

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

    B("1") = 1;
    B("2") = 1;
    B("3") = 1;
    B("4") = 0;
    B("5") = 0;
    B("6") = 1;
    B("7") = 0;
    B("8") = 0;

    cout << Dice(A(), B()) << endl;

    GlobalClose;

    return 0;
}

```

prints 1

Figure 70 Dice Example

```

#include <mumpsc/libmpscpp.h>

global A("A");
global B("B");

int main() {

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

    B("1") = 1;
    B("2") = 1;
    B("3") = 1;
    B("4") = 0;
    B("5") = 0;
    B("6") = 1;
    B("7") = 0;
    B("8") = 0;

    cout << Cosine(A(), B()) << endl;

    GlobalClose;

    return 0;
}

```

```

    }

    prints 0.75

```

Figure 71 Cosine Example

10.16.5.21 Transpose

void global::Transpose(global out)

The invoking object is transposed and the result is placed in *out*. Any prior contents of the array *out* are deleted before the operation commences. See Figure 72.

```

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

global d("d");
global f("f");

int main() {

    d("1","1")=2;
    d("1","2")=3;
    d("2","1")=4;
    d("2","2")=0;

    d().Transpose(f()); // transpose d() placing result in f()

    cout << f("1","1") << " " f("1","2") << endl;
    cout << f("2","1") << " " f("2","2") << endl;

    GlobalClose;

    return EXIT_SUCCESS;
}

```

Yields:

```

2 4
3 0

```

Figure 72 Transpose Example

10.16.5.22 Centroid

void global::Centroid(global B)

A centroid vector *B* is calculated for the invoking two dimensional global array. 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 invoking global array (*A*) must contain at least two dimensions. See Figure 73.

```

#include <mumpsc/libmpscpp.h>

global A("A");
global B("B");

int main() {

    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(1);
    if (a=="") break;
    cout << a << " --> " << B(a) << endl;
}

return 0;
}

```

Yields:

```

1 --> 5
2 --> 5
3 --> 5
4 --> 5
5 --> 5
6 --> 5
7 --> 5
8 --> 5
9 --> 5

```

Figure 73 Centroid Example

The above yields a vector giving the average value of each named column of the matrix "A" (5 in this case since each column is initialized with 5).

10.16.5.23 Correlation Functions

```

void global::TermCorrelate(global B)
void global::DocCorrelate(global B, mstring fcname, double threshold)

```

These functions build document indexing correlation matrices. The invoking **global** is assumed to be a two dimensional document-term matrix whose rows are documents and whose columns represent the occurrence of terms in the documents (either weights or frequencies).

TermCorrelate() builds a square term-term correlation matrix in *B* from the invoking document-term matrix.

DocCorrelate() builds a square document-document correlation matrix from the invoking document-term matrix. The name of the function to be used in calculating the document-document similarity is given in *fcn* and may be *Cosine*, *Jaccard*, *Dice*, or *Sim1*. The minimum correlation threshold is given in *threshold* which defaults to 0.80 if omitted.

```

#include <mumpsc/libmpscpp.h>

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(1);
        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)
    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 74 TermCorrelate Example

The example in Figure 74 gives the number of co-occurrences of each word with each other word. For example, the words "computer" and "memory" co-occur in two vectors (2 nd 3) while the words "laptop" and "computer" co-occur in all three vectors. If each vector is thought of as a document, the strength of the co-occurrences between words is a measure of similarity for indexing purposes.

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

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().DocCorrelate(B(),"Cosine",.5);

    mstring a;
    mstring b;

    a=""

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

Yields

```
1
    2 0.887096774193548
    3 0.741935483870968

2
    1 0.887096774193548
    3 0.701612903225806
```

3

1 0.741935483870968
2 0.701612903225806

Figure 75 DocCorrelate Example

The example in program in Figure 75 calculates the similarities between the document vectors according to the Cosine method.

10.16.5.24 IDF

void global::IDF(double DocCount)

The *IDF()* function calculates for the global array vector provided the *inverse document frequency* weight of each term. The vector should be indexed by words and have stored the number of documents in which each word occurs. The document count will be replaced by the calculated IDF value. The IDF is $\log_2(\text{DocCount}/W_n)+1$ 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. See Figure 76.

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

global a("a");

int main() {

    kill(a());
    a("now") = 2;
    a("is") = 5;
    a("the") = 6;
    a("time") = 3;
    a().IDF(4);
    a().TreePrint();
    return 0;
}

yields:

is=0.678072
now=2.000000
the=0.415037
time=1.415037
```

Figure 76 IDF Example

10.16.5.25 Sum

double global::Sum()

The global array nodes beneath the referenced global array are summed. Non numeric quantities are treated as zero. See Figure 77.

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

global A("A");

int main() {

mstring i, j;
```

```

for (i = 1; i < 11; i++)
    for (j = 1; j < 11; j++) {
        A(i, j) = 5;
    }
cout << "Full sum:  " << A().Sum() << endl;
cout << "A row sum: " << A("5").Sum() << endl;

GlobalClose;

return EXIT_SUCCESS;
}

Yields

Full sum:  500
A row sum: 50

```

Figure 77 Sum Example

10.16.5.26 Translate

```

mstring global::Translate(mstring)
mstring global::Translate(mstring, mstring)

```

If only one **mstring** argument is given, characters appearing in the argument **mstring** are removed from the invoking **global**.

If two argument **mstrings** appear and the first and second argument **mstring** are of the same length, characters from the invoking **global** that appear in the first argument **mstring** are replaced by their counterparts from the second argument **mstring**.

If the first argument **mstring** is longer than the second argument **mstring**, the characters from the first argument **mstring** which have no counterpart in the second argument **mstring** are removed.

A "counterpart" is a character equally offset in the second argument **mstring** to the character in the first argument **mstring**.

10.17 Direct Btree Access

Programmers may access the btree directly through the builtin **BTREE** macro. A number of examples can be found in *mumpsc/doc/examples/btree* in the distribution.

To access the btree directly from a C++ program:

You must first install the Mumps compiler and MDH. Include at the beginning of your program. You can now access the btree directly with the **BTREE** macro (see description below). Note: any keys you store in the btree co-exist with Mumps/MDH keys. In rare cases, these can interfere with one another if a key you store lies in the range of a global array key set.

For example, the following program stores **NBR_ITERATIONS** (defined in *btree.h* which is included by *libmpscpp.h* usually with the value 100,000) of keys and data into the btree and then retrieves them (this "btest1.cpp" from *mumpsc/doc/examples/btree.cpp*). See the other examples and the documentation below for further details. See Figure 78.

```
/*#+++++
*#+ Mumps Compiler Run-Time Support Functions
*#+ Copyright (c) 2001, 2002, 2003, 2004 by Kevin C. O'Kane
*#+ okane@cs.uni.edu
*#+
*#+ 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
*#+
*#+ http://www.cs.uni.edu/~okane
*#+
*#+++++
*#+
*#+ Some of this code was originally written in Fortran
*#+ which will explain the odd array and label usage,
*#+ especially arrays beginning at index 1.
*#+
*#+++++

#include <mumpsc /libmpscpp.h>

int main() {

    long i,j;

    unsigned char key[1024],data[1024];

    printf("Store sequentially ascending keys");

    for (i = 0; i < NBR_ITERATIONS; i++) {

        sprintf( (char *) key, "key %ld", i);
```

```

    sprintf( (char *) data,"%ld%c", i, 0);

    if (!BTREE(STORE, key, data)) {
        printf("error\n");
        return 1;
    }

    if (i%60000L == 0) { printf("\n %ld ",i); fflush(stdout); }

    if (i%1000 == 0) { putchar('.'); fflush(stdout); }
}

printf("\nretrieve");

for (i = 0; i < NBR_ITERATIONS; i++) {

    sprintf( (char *) key,"key %ld",i);

    if (!BTREE(RETRIEVE, key, data)) {
        printf("error 1\n");
        return 1;
    }

    sscanf( (char *) data, "%ld", &j);

    if (j!=i) {
        printf("error 2\n");
        printf("%d != %d\n", i, j);
        return 1;
    }

    if (i%60000L == 0) { printf("\n %ld ",i); fflush(stdout); }
    if (i%1000 == 0) { putchar('.'); fflush(stdout); }
}

printf("\nlooks good!\n");

strcpy( (char *) key, "");
strcpy( (char *) data, "");

BTREE(CLOSE,key,data);
return 1;
}

```

Figure 78 BTREE Example

10.18 Invoking the Mumps Interpreter

The full facilities of the Mumps interpreter can be invoked from C++ programs. The interpreter reads, parses and executes commands presented to it at run time. It may also read and execute text files containing Mumps programs. The interpreter is invoked by means of the *Xecute()* macro and *xecute()* functions:

```
int Xecute("command")
int xecute(mstring command)
int xecute(string command)
int xecute(char * command)
```

These functions and macro invoke the Mumps interpreter and execute the text replacing "command". They return 1 if successful, 0 otherwise. With *Xecute()*, if the mumps command contains quotes or other special symbols, they will be automatically prefixed with backslashes (e.g., quote becomes `\`).

```
Xecute("set i=\"test\")");
Xecute("fors i=$order(^a(i)) quit:i=\"\" set sum=sum+^a(i)");
```

Details on the Mumps Language are contained in the file *compiler.html* in the *mumpsc/doc* subdirectory of the Mumps Compiler distribution. See also: `mtring::Eval()` for expression interpretation.

10.19 Miscellaneous Functions

10.19.1 GTK / Glade functions

The following functions may be used with the GTK / Glade programming facility:

10.19.1.1 void mdh_tree_level_add(GtkTreeStore *tree, int depth, char * col1 [, char *col2 ...]);

Add the value in *col1* to the tree at level *depth* and populate the remaining columns of this row with *col2*, *col3*, ... to a limit of five columns.

10.19.1.2 int mdh_dialog_new_with_buttons(GtkWindow *win, char * text)

Open a modal popup dialog box with the options "Yes" and "No". The contents of *text* will be displayed. Returns 0 if *no* is clicked and 1 if *yes* is clicked. The value -1 is returned if the box is dismissed without selection.

10.19.1.3 int mdh_toggle_button_get_active(GtkToggleButton *b)

Returns 1 if the button is active; 0 otherwise.

10.19.1.4 char * mdh_entry_get_text(GtkEntry *e, char * txt)

Returns the text contents of the specified entry box. The return pointer points to the string pointed to by *txt*. The user is responsible for providing a character array pointed to by *txt* large enough to contain the text retrieved.

10.19.1.5 void mdh_toggle_button_set_active(GtkToggleButton *b, int v)

The named toggle button will be set to active if the value of *v* is non-zero; inactive otherwise. Triggers a toggle signal.

10.19.1.6 void mdh_entry_set_text(GtkEntry *e, char * txt)

Sets the contents of the named entry box. Triggers a entry changed signal.

10.19.1.7 void mdh_text_buffer_set_text(GtkTextBuffer *t, char * txt)

Sets the contents of the named text buffer.

10.19.1.8 void mdh_label_set_text(GtkLabel *l, char * txt)

Sets the contents of the named label.

10.19.1.9 void mdh_widget_hide(GtkWidget *w)

Hides the named widget.

10.19.1.10 void mdh_widget_show(GtkWidget *w)

Displays the named widget.

10.19.1.11 char * mdh_tree_selection_get_selected(GtkTreeSelection *t, int col, char *txt)

Returns the value in column 1 of the named tree.

10.19.1.12 void mdh_tree_store_clear(GtkTreeStore *t)

Clears the named tree store.

10.19.1.13 double mdh_spin_button_get_value(GtkSpinButton *s)

Returns the value in the named spin button.

10.19.1.14 void mdh_spin_button_set_value(GtkSpinButton *s, double v)

Sets the value of the named spin button.

10.20 Boyer-Moore-Gosper Functions

int bmg_fullsearch(mstring search_string, mstring buffer_base)

Returns the number of non-overlapping instances of "search_string" in "buffer_base". See Figure 79.

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

int main() {

    mstring a = "now is the time for all good men to come to the aid of the
party";
    mstring b = "to";
    cout << bmg_fullsearch(b, a) << endl;
    return EXIT_SUCCESS;
}

yields:

    2
```

Figure 79 Boyer-Moore Example

These functions are publically available from:

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

and are believed to be contributed source and are unrestricted with respect to use and redistribution, and, that most, if not all, the code was written by employee(s) of the United States and thus in the public domain. The distribution contains, in part, the following notes:

Here are routines to perform fast string searches using the Boyer-Moore-Gosper algorithm; they can be used in any Unix program (and should be portable to non-Unix systems). You can search either a file or a buffer in memory.

The code is mostly due to James A. Woods (jaw@ames-aurora.arpa) although I have modified it heavily, so all bugs are my fault. The original code is from his sped-up version of egrep, recently posted on mod.sources and available via anonymous FTP from ames-aurora.arpa as pub/egrep.one and pub/egrep.two. That code handles regular expressions; mine does not.

These have only been tested on 4.2BSD Vax systems.

-Jeff Mogul

mogul@navajo.stanford.edu
decwrl!glacier!navajo!mogul
BMGSUBS(3L)

BMGSUBS(3L)

NAME

(bmgsbubs) bmg_setup, bmg_search, bmg_fsearch - Boyer-Moore-Gosper string search routines

SYNOPSIS

```
bmg_setup(search_string, case_fold_flag)
char *search_string;
int case_fold_flag;

bmg_fsearch(file_des, action_func)
int file_des;
int (*action_func)();

bmg_search(buffer_base, buffer_length, action_func)
char *buffer_base;
int buffer_length;
int (*action_func)();
```

DESCRIPTION

These routines perform fast searches for strings, using the Boyer-Moore-Gosper algorithm. No meta-characters (such as '*' or '.') are interpreted, and the search string cannot contain newlines.

Bmg_setup must be called as the first step in performing a search. The search_string parameter is the string to be searched for. Case_fold_flag should be false (zero) if characters should match exactly, and true (non-zero) if case should be ignored when checking for matches.

Once a search string has been specified using bmg_setup, one or more searches for that string may be performed.

Bmg_fsearch searches a file, open for reading on file descriptor file_des (this is not a stdio file.) For each line that contains the search string, bmg_fsearch will call the action_func function specified by the caller as action_func(matching_line, byte_offset). The matching_line parameter is a (char *) pointer to a temporary copy of the line; byte_offset is the offset from the beginning of the file to the first occurrence of the search string in that line. Action_func should return true (non-zero) if the search should continue, or false (zero) if the search should terminate at this point.

Bmg_search is like bmg_fsearch, except that instead of searching a file, it searches the buffer pointed to by buffer_base; buffer_length specifies the number of bytes in the buffer. The byte_offset parameter to action_func gives the offset from the beginning of the buffer.

If the user merely wants the matching lines printed on the standard output, the action_func parameter to bmg_fsearch or bmg_search can be NULL.

AUTHOR

Jeffrey Mogul (Stanford University), based on code written by James A. Woods (NASA Ames)

BUGS

Might be nice to have a version of this that handles regular expressions.

There are large, but finite, limits on the length of both pattern strings and text lines. When these limits are exceeded, all bets are off.

The string pointer passed to `action_func` points to a temporary copy of the matching line, and must be copied elsewhere before `action_func` returns.

`Bmg_search` does not permanently modify the buffer in any way, but during its execution (and therefore when `action_func` is called), the last byte of the buffer may be temporarily changed.

The Boyer-Moore algorithm cannot find lines that do not contain a given pattern (like `"grep -v"`) or count lines (`"grep -n"`). Although it is fast even for short search strings, it gets faster as the search string length increases.

16 May 1986

BMGSUBS(3L)

10.21 `cvt()`

```
char *cvt(long i)
char *cvt(double i)
char *cvt(float i)
char *cvt(int i)
```

These functions return a null terminated varying length character string containing in printable version of the argument. The functions contain short static character arrays and, consequently, are not threadsafe.

10.22 `xecute()` and `command()`

`command()` is a macro that takes a quoted string constant argument. The macro surrounds the string with an extra set of quotes and processes any embedded quotes to backslash-quote. It then invokes a function (`__command__()`) which strips the extra surrounding quotes. The net effect of this is that you can pass a quoted string containing quotes without the need for "leaning toothpick" notation. Example:

```
xecute(command("for i=1:1:10 "test ",i,!"));
strcpy(target, command("for i=1:1:10 write "test ",i,!"))
```

The argument must be a character string constant.

10.23 `ErrorMessage()`

```
void ErrorMessage(char * message, int line_number)
```

This function (written in C and part of the underlying legacy library) will print an error message, close the global array files and terminate the program. The integer "line_number" will be printed with the message. The pre-processor predefined macro `__LINE__` can be used here. Example:

```
ErrorMessage("Cannot locate patient", __LINE__);
```

10.24 Error Exceptions

The toolkit generates (throws) exceptions for certain conditions. For example, when you access global arrays with the toolkit, the accesses may result in the thrown error exceptions:

1. *ConversionException.*
2. *GlobalNotFoundException*
3. *MumpsSymbolTableException.*
4. *NumericRangeException.*

The first can occur in any context that attempts to retrieve data from a global array where none exists. The second occurs if you attempt to convert the contents of a global to a numeric type where the contents of the global are not valid data for the conversion.

If uncaught, both exceptions will result in program termination.

The following are the exceptions thrown by the toolkit:

1. `ConversionException()` - usually occurs when you attempt to store a value from a global array into a numeric variable but the string in the global is not a valid number.
2. `GlobalNotFoundException()` - thrown by an attempt to reference non-existent global array data.
3. `MumpsSymbolTableException()` - thrown by an attempt to fetch the value of a non-existent variable from the Mumps runtime symbol table.
4. `NumericRangeException()` - thrown by attempts to divide by zero or using arguments with values less than or equal to zero to log functions.

See Figure 80.

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

global a("a");

int main() {
    long i;
    a().Kill();
    mstring A;
    a("1") = "now is the time";
    try {
        i = a("1");
    }

    catch ( ConversionException ce) {
        cout << ce.what() << endl;
    }
    try {
        i = a("22");
    }
    catch (GlobalNotFoundException nf) {
        cout << nf.what() << endl;
    }

    try {
        A=SymGet("abc");
    }
    catch (MumpsSymbolTableException st) {
        cout << nf.what() << endl;
    }

    return 0;
}
```

Figure 80 Exceptions Examples

10.25 HitRatio()

double HitRatio(void)

Calculates the native global array processor cache hit ratio since the beginning of the program or the last call to *HitRatio()* The native global array file processor, as opposed to the Berkeley Data Base, keeps track of how many file I/O requests are satisfied from data already in the file system's cache. This function gives the percentage of cache hits. It only works with the native global array processor.

10.26 Hashing functions

```
char * hash(char * str)
long lhash(char * str)
```

hash() returns either a null terminated character string up to 10 characters in length containing a numeric hash code of the string passed as an argument. The argument may be up to *STR_MAX* characters in length. *lhash()* returns an **unsigned long** value of the hash value.

10.27 Dump Global Array Database

```
void Dump(char * filename)
void Dump(mstring filename)
void Dump(string filename)

void Restore(char * filename)
void Restore(mstring filename)
void Restore(string filename)
```

The global array data base is dumped (written in its entirety) to filename or read and restored from filename (null terminated array of chars). Both operations must not be done from the same program.

10.28 Stream Output

```
friend ostream & operator << (ostream&, global)
```

A global array may participate in stream output. For example:

```
gbl("A", "B", "C") << "test test test";
cout << gbl("A", "B", "C") << endl;
```

The above will print "test test test" (without quotes) followed by the newline character. Alternatively:

```
cout << gbl("A", "B", "C").Get() << endl;
```

will do the same thing (the Get() function returns "char *").

10.29 Smith-Waterman Alignment Function

```
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 gap=-1, int
mismatch=-1, int match=2])
```

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:

```
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 gap and mismatch penalties are -1 and the match reward is +2. The parameters "gap", "mismatch" and "match" are the gap and mismatch penalties (negative integers) and

the match reward (a positive integer). These values default to -1, -1 and 2 respectively. If insufficient memory is available, a segmentation violation will be raised.]

The first character of each sequence string MUST be blank.

See Figure 81.

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

```
int main() {
```

```
    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);
```

```
    return 0;
```

```
}
```

results in:

S-W Alignments for:

64 now is the time for all good men to come to the aid of the party

22 time for good men

29 men 32

::::

19 men 22

score=12

29 - men 32

::::

18 men 22

score=11

23 l good-- men 32

::::: ::::

11 good men 22

score=24

22 ll good-- men 32

::::: ::::

11 - good men 22

score=23

16 for all good-- men 32

::::: ::::: ::::

6 for -- good men 22

score=37

12 time- for all good-- men 32

::::: ::::: ::::: ::::

1 time for -- good men 22

score=48

Figure 81 Smith-Waterman Example

10.30 Stop list functions: StopINIT(), StopLookup()

```
void StopInit(mstring file)
void StopInit(string file)
void StopInit(char * file)

int StopLookup(mstring word)
int StopLookup(string word)
int StopLookup(char * word)
```

StopInit() reads the sorted file "file" of stoplist words into the stoplist container (one word per line). *StopLookup()* returns 0 if "word" is not found and 1 if "word" is found in the stoplist.

10.31 Synonym Functions: SymInit(), SYN()

```
int SynInit(mstring filename)
int SynInit(string filename)
int SynInit(char * filename)

mstring SYN(mstring word)
string SYN(string word)
char * SYN(char * word)
```

SynInit() opens and reads a synonym file and returns the number of lines read. The maximum number of synonyms permitted is determined by "SYNMAX" in *libmpscpp.h* (default is 20,000). Each line of the synonym file consists of multiple words, in lower case, separated from one another by a single blank. The first word is the root alias and the remaining words are alternative synonyms. The function *SYN()* looks up a word. If the word is an alternative synonym, the root alias is returned. If not, the original word is returned.

10.32 int \$test

Returns integer 1 or 0 indicating the success or failure of certain previous commands. Some, but not all, commands set "\$test".

10.33 Xecute()

```
int Xecute(char * command)
int Xecute(mstring command)
int Xecute(string command)
int Xecute(char * command)
```

These functions invoke the Mumps interpreter which executes *command*. Returns 1 if successful, 0 otherwise.

The macro *Xecute()* is a special case. It is used with character string constants. It will pre-process a character string constant command and insert the backslash escape character prior to any embedded quotes thus permitting more normal appearing text (see similar macro *command()*).

Examples:

```
mstring c;
Xecute("for i=$Order(^a(i)) q:i="" s sum=sum+^a(i)");

c = "for i=1:1:10 write i,!";
xecute(c);

c = command("for i=1:1:10 write \"ans=\",i,!");
xecute(c);
```

10.34 Zseek() Ztell()

```
bool Zseek(FILE *file, offset)
bool Ztell(FILE *file, offset)
```

These functions are used in connection with direct access files opened with FILE pointers (see: *fopen()*). They are compatible with 64 bit file pointer 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* into the **mstring** or global variable represented 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 in connection with *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 relative to the start of the file.

10.35 MDH Functions

10.35.1 \$z~mdh~toggle~button~get~active(ToggleButtonReference)

Returns 0 if the button is inactive, 1 if active

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

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

10.35.3 \$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.

10.35.4 \$z~mdh~entry~get~text(EntryReference)

Returns the current string contents of the referenced Entry box.

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

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

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

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

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

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

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

Returns value in designated column of referenced TreeModel.

10.35.9 \$z~mdh~tree~store~clear(TreeStoreReference)

Clears (deletes) the contents of the referenced TreeStore.

10.35.10 \$z~mdh~tree~level~add(TreeStoreReference,treeDepth,index,data,...)

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

10.35.11 \$z~mdh~spin~button~get~value(SpinButtonReference)

Returns the current value of the referenced SpinButton.

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

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

10.35.13 \$z~mdh~widget~hide(widgetReference)

Hides the widget from view.

10.35.14 \$z~mdh~widget~show(widgetReference)

Displays (un-hides) the widget.

11 GTK Desktop GUI Apps

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

11.1 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 82 gives an example that includes several widget types.

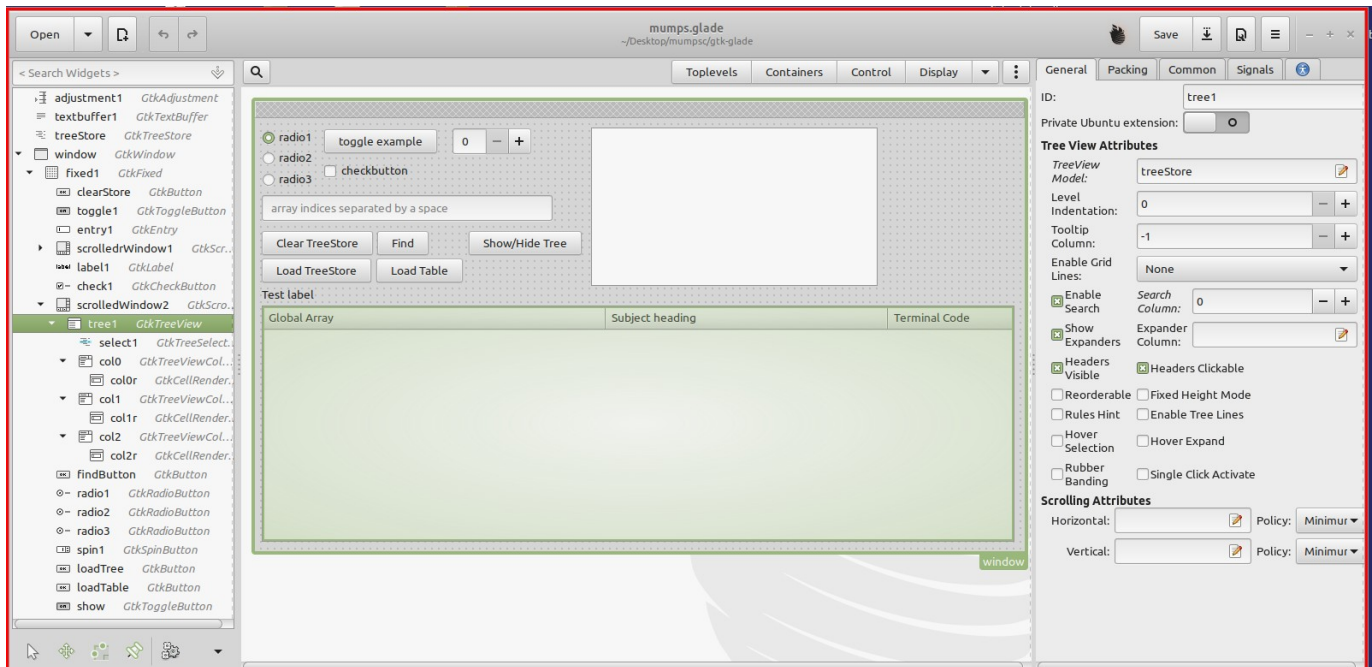


Figure 82 Glade Canvas

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).

11.2 GTK Example

11.2.1 Glade Design Tool

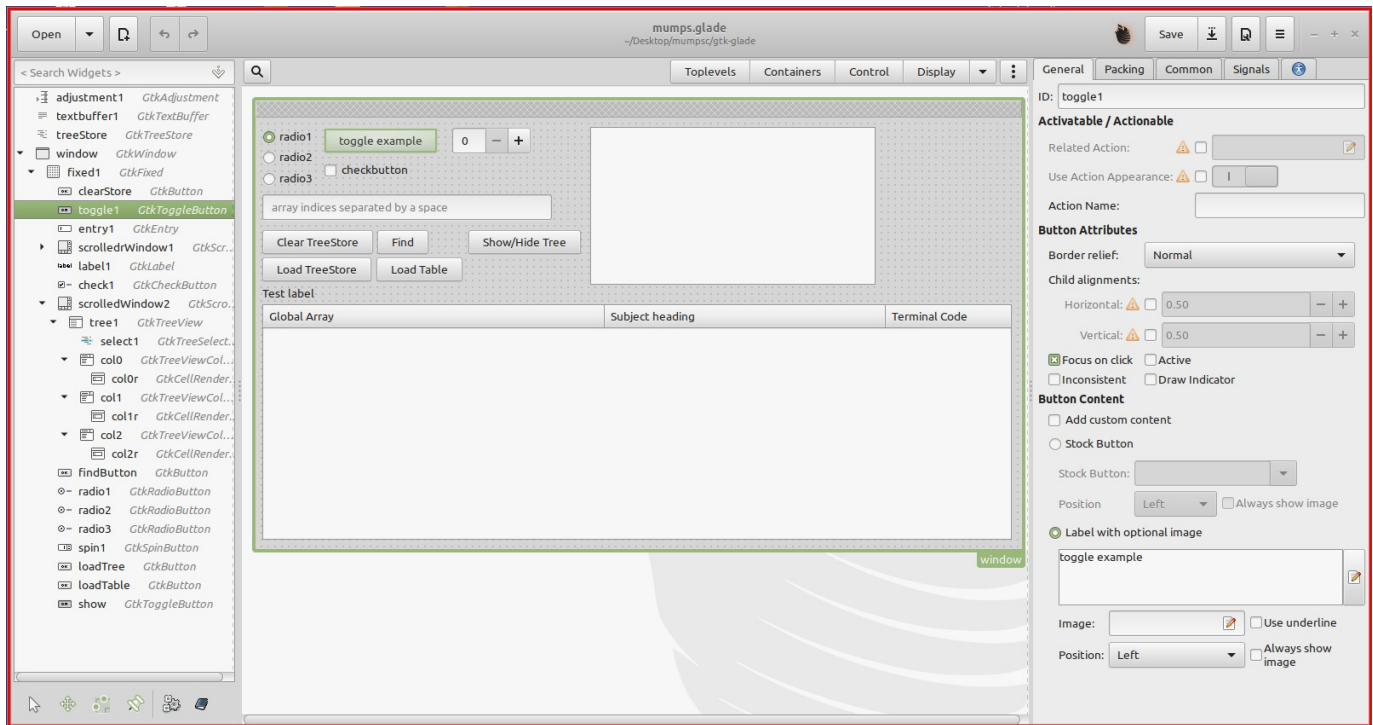


Figure 83 Toggle Button Screen 1

In Figure 83 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.

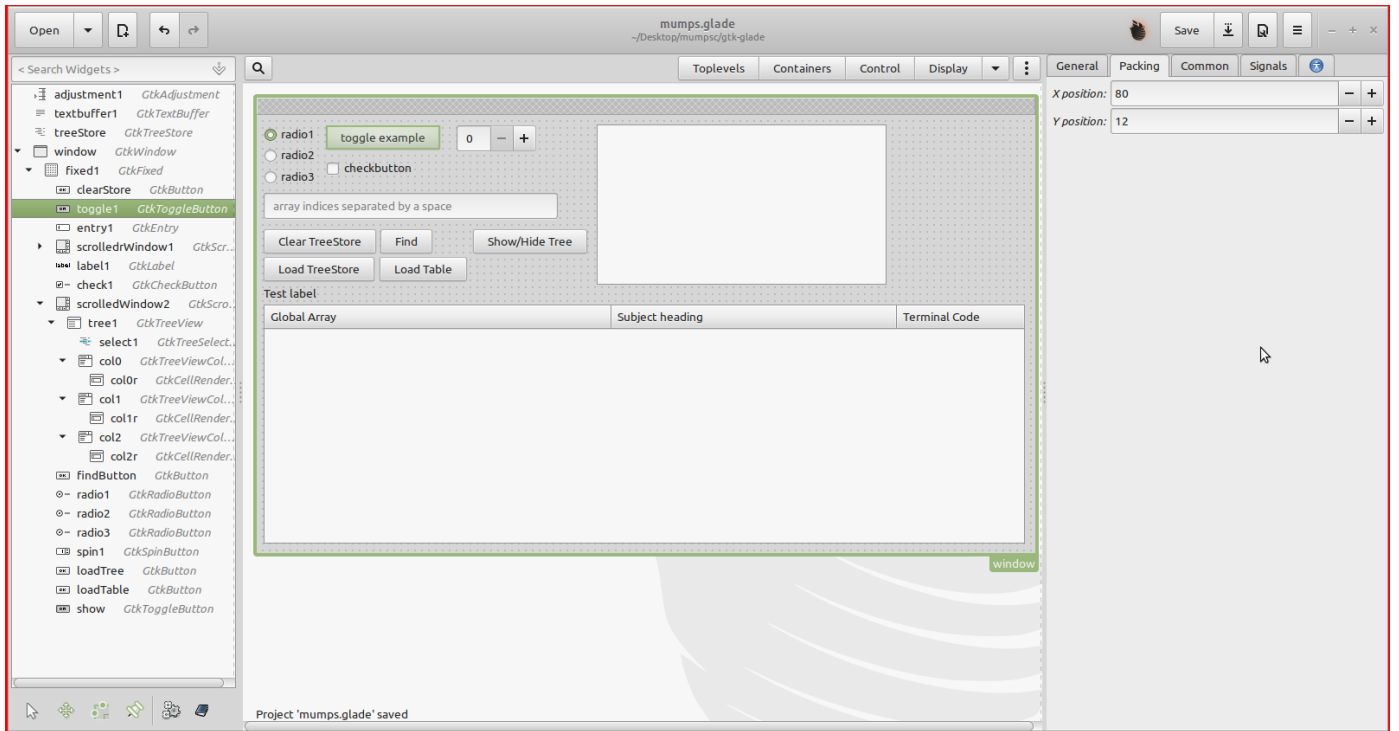


Figure 84 Toggle Button Screen 2

In Figure 84 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.

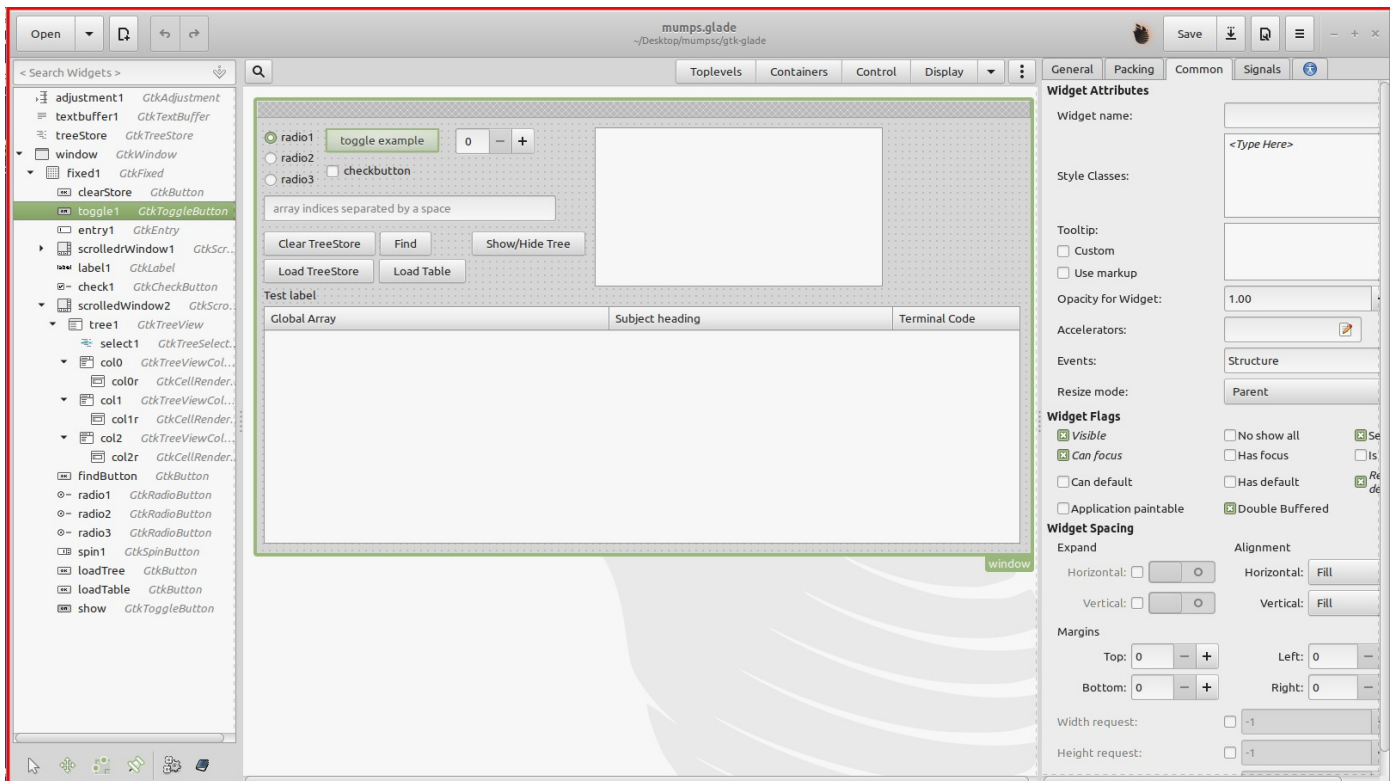


Figure 85 Toggle Button Screen 3

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.

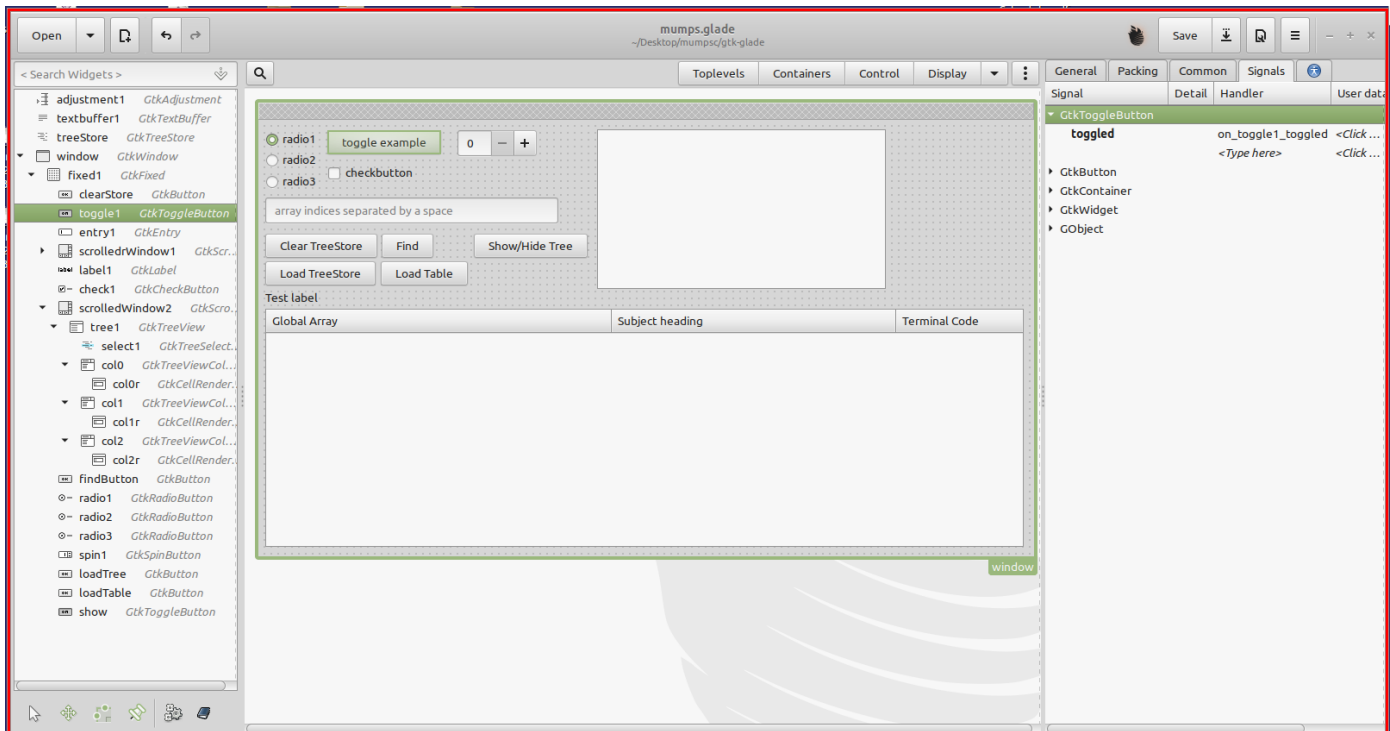


Figure 86 Toggle Button Screen 4

In Figure 86 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`.

11.2.2 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 togg1 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:

11.2.2.1 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 `toggle1` widget, the line:

```
GtkToggleButton *toggle1;
```

appears, among others.

11.2.2.2 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*.

11.2.2.3 gtk3.h

This file contains the basic signal handlers (written in C) which are used to invoke the corresponding Mumps programs which will actually 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.

11.2.2.4 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.

11.2.2.5 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.

11.2.2.6 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 $~mdh~toggle~button~get~active(toggle1),!
```

The function `$~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.

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. Secondly, 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

absolute value.....	37	Exponential base 2.....	37
ACID.....	9, 11	exponential format.....	20
alignment.....	51	Extended Arithmetic Precision.....	20
arc cosine.....	37	Extended Precision Math.....	13
Arc sine.....	37	extended precision software.....	12
Arc tangent.....	37	Extract Function.....	74
Array Index Collating Sequence.....	28	file.....	39
ASCII.....	39	File Names.....	28
Ascii Function.....	73	File Names Containing Directory Information.....	28
astyle.....	20	file pointer.....	40
backreferences.....	47	files.....	35
Base 10 log.....	37	Find Function.....	74
Base 2 log.....	37	floating point numbers.....	13
basename.....	36	For Command Extensions.....	24
Bash Functions.....	36	fractional precision.....	20
Begin Transaction.....	9	ftello.....	41
BEGIN TRANSACTION;.....	19	Full List of Configure Options.....	14
begins Function.....	73	Function Calls.....	29
blanks.....	39, 40	General Relational Database Options.....	14
bmj_fullsearch.....	116	Global arrays.....	8
Boyer-Moore-Gosper Function.....	47	GNU GPL/LGPL.....	60
Break and Quit.....	25	GNU MPFR library3.....	13
btree.....	39	GNU multiple precision arithmetic library2.....	13
Btree block size.....	15	Goto.....	20
buffer.....	41	Gregorian.....	38
buffers.....	39	Gregorian date.....	38
c_str Function.....	73	GTK Desktop GUI Apps.....	125
C++ container.....	53	Hardware Math.....	13
cache hit ratio.....	40	Horolog Function.....	75
cache size.....	15	HTML.....	40
centroid vector.....	44	Implementation Notes.....	20
cgi-bin.....	40	include.....	6
Checkout.....	6	Inline C++.....	31
Class mstring.....	70	int.....	13
collating sequence.....	28	Integer arithmetic.....	13
command line interpreter.....	17	internal buffer.....	41, 42
Commit.....	9	Interpreting a Mumps Program.....	17
COMMIT TRANSACTION;.....	19	Inverse Document Frequency score.....	51
Compiler Error Messages.....	58	Jaccard.....	45
Compiling Large Programs.....	30	January 1, 1970.....	38
configure prefix.....	14	Job command.....	28
Correlation Functions.....	52	Julian date.....	38
Cosine.....	37, 45	Justify Function.....	75
data.dat.....	17	key.dat.....	17
database.....	35	Kill Command.....	24
Date functions.....	38	left justifies.....	40
Debian.....	9	Legacy-Mumps-Interpreter.....	6
decorate Function.....	73	Length Functio.....	75
Dice.....	45	libmpscpp.h.....	61
DocCorrelate().....	107	limit to integer precision.....	21
document vectors.....	45	Linux time.....	38
Document-Document matrix.....	53	local_install.....	6
Documentation.....	6	Lock Command.....	27
dump.....	39	logarithm.....	37
dump file.....	39	logarithms.....	37
EncodeHTML Function.....	74	Math Functions.....	37
ends Function.....	74	mcvt Function.....	75
ength Function.....	75	mdh_dialog_new_with_buttons.....	115
Eval Function.....	74	mdh_entry_get_text.....	115
Examples.....	6	mdh_entry_set_text.....	115
exponent.....	38	mdh_label_set_text.....	116
exponential.....	37	mdh_spin_button_get_value.....	116
Exponential.....	37	mdh_spin_button_set_value.....	116
Exponential base 10.....	37	mdh_text_buffer_set_text.....	115

mdh_toggle_button_get_active.....	115	similarity coefficients.....	45
mdh_toggle_button_set_active.....	115	Similarity Functions.....	45
mdh_tree_level_add.....	115	sine.....	38
mdh_tree_selection_get_selected.....	116	Sine function.....	38
mdh_tree_store_clear.....	116	Smith Waterman.....	51
mdh_widget_show.....	116	source code programs.....	17
Modulo Operator.....	20	SQL.....	9
mstring.....	61	SQL functions.....	54
mstring Example.....	80	Sqlite3 Database Configuration.....	10
mstring Functions and Methods.....	73	square root.....	38
mstring Operations.....	70	Square root.....	38
mstring Operator Overloads.....	73	stdin.....	42
Multi-Dimensional and Hierarchical Database Toolkit.....	60	Stem Function.....	79
multiple blanks.....	39	Stop and Synonym Function.....	53
Mumps Global Array Database.....	7	stop words.....	53
Mumps-Interpreter-Compiler-Library.....	6	STR_MAX.....	51
Mumps-Language-Processors.....	6	strfmon().....	30
mumps-native-single-user-amd64.....	6	string alignment.....	51
Mumps-Projects.....	6	string replacement.....	49
mumps-sql-db-create.....	10	string search.....	47
mumps-sqlite3-amd64.....	6	sum.....	45
mumps.sqlite.....	10, 17	SymGet SymPut Functions.....	79
mumps2c.....	58	synonym.....	54
mumpsc.....	18	tangent.....	38
Naked indicator.....	28	Tangent function.....	38
National Library of Medicine.....	101	Term-Term matrix.....	52
native global arrays.....	35	TermCorrelate().....	107
Natural log.....	37	Text Processing Functions.....	45
natural logarithm.....	37	time().....	39
Navigating Globals.....	66	Token Function.....	79
New Command.....	21	Translate Function.....	79
NLM.....	101	Tree Structured Medical Record.....	8
offset.....	41	ulimit.....	51
open.....	19, 55	Vector and Matrix Functions.....	43
padding.....	40	void mdh_widget_hide.....	116
PAGE_SHIFT.....	15	with-float_digits.....	13
pattern.....	49	word stem.....	41
Pattern Function.....	75	xecute.....	18
Perl.....	47	Z Functions.....	36
Perl Compatible Regular Expression Library License.....	150	zbasename.....	36
Perl Function.....	76	zmain.....	18
Piece Function.....	76	--with-hardware-math=no.....	13
Power function.....	38	--with-locale.....	16
PRAGMA.....	19	--with-.....	14
precision.....	13	--with-block.....	15
radians.....	37	--with-cache.....	15
random number generator.....	41	--with-data_size.....	14
readline.....	17	--with-dbfile.....	14
ReadLine Function.....	77	--with-dbname.....	14
replace Function.....	77	--with-float-bits.....	13, 16
restore.....	39	--with-float-bits=val.....	20
Rollback.....	9	--with-float-digits.....	14, 16, 20
ROLLBACK TRANSACTION;.....	19	--with-hardware-math.....	13, 16
Rounding.....	21	--with-ibuf.....	16
Running a Mumps Program.....	17	--with-index_size.....	14
Running the Mumps CLI Interpreter.....	17	--with-int-32.....	13
s_str Functio.....	79	--with-long-double.....	13
SAVEPOINT.....	19	--with-maxglobal.....	16
Scan Functions.....	42	--with-no-inline.....	16
ScanAlnum Function.....	77	--with-profile.....	16
seed.....	41	--with-strmax.....	16
shell.....	41	--with-terminate-on-error.....	16
Shell Commands.....	34	#!/usr/bin/mumps.....	17
shred Function.....	77	\$atan.....	37
ShredQuery Function.....	78	\$fnumber().....	20
Sim1.....	45	\$Fnumber() Function.....	30

\$justify()	21	\$zpad	40
\$random	41	\$zPerlMatch()	47
\$select()	30	\$zpow	38
\$Select() Function	30	\$zReplace	49
\$test	42	\$zrestore	39
\$z	19, 38, 39, 55	\$zrestore[39
\$z~mdh~dialog~new~with~buttons	123	\$zseek	40, 41
\$z~mdh~entry~get~text	123	\$zShred	49
\$z~mdh~entry~set~text	123	\$zShredQuery	49
\$z~mdh~label~set~text	123	\$zsin	38
\$z~mdh~spin~button~get~value	123	\$zSmithWaterman	51
\$z~mdh~spin~button~set~value	124	\$zSqlite	19
\$z~mdh~text~buffer~set~text	123	\$zSqlite("begin transaction")	19
\$z~mdh~toggle~button~get~active	123	\$zSqlite("commit transaction")	19
\$z~mdh~toggle~button~set~active	123	\$zSqlite("pragma",option)	19
\$z~mdh~tree~level~add	123	\$zSqlite("rollback",[savepoint])	19
\$z~mdh~tree~selection~get~selected	123	\$zSqlite("savepoint",[savepoint_name])	19
\$z~mdh~tree~store~clear	123	\$zSqlite("SQL",sql_command)	19
\$z~mdh~widget~hide	124	\$zsqlOpen	19, 55
\$z~mdh~widget~show	124	\$zsqr	38
\$zabs	37	\$zsrand	41
\$zacos	37	\$zstem	41
\$zasin	37	\$zStopInit	53
\$zb	39	\$zStopLookup	53
\$zchdir	39	\$zSynInit	53
\$zcos	37	\$zSynLookup	53
\$zCurrentFile	39	\$zsystem	41
\$zd1	38	\$ztan	38
\$zd2	38	\$ztell	41
\$zd3	38	\$zu	41
\$zd4	38	\$zwi	41
\$zd5	38	\$zwn	41
\$zd6	38	\$zwp	41, 42
\$zd7	38	\$zws	42
\$zd8	38	\$zws(string)	42
\$zdate	38	\$zz	44
\$zdump	39	\$zzAvg	43
\$zexp	37	\$zzBMGSearch	47
\$zexp10	37	\$zzCosine	45
\$zexp2(arg)	37	\$zzCount	44
\$zfile	39	\$zzDice	45
\$zflush	39	\$zzDocCorrelate	53
\$zfunctions	36	\$zzInput(var)	42, 43
\$zgetenv(arg)	40	\$zzJaccard	45
\$zhit	40	\$zzMax	44
\$zhtml	40	\$zzMin	45
\$zlog	37	\$zzMultiply	45
\$zlog10	37	\$zzScan	42
\$zlog2	37	\$zzScanAlnum	42
\$zlower	40	\$zzSim1	45
\$znative	19, 55	\$zzSoundex(s1)	50
\$zNative	19, 55	\$zzSum	45
\$zNoBlanks(arg)	40	\$zzTermCorrelate	52
\$znatural	40	\$zzTranspose	45
\$zp	40		