Mini-XML Programmers Manual
Version 2.9

MICHAEL R. SWEET

Mini-XML Programmers Manual, Version 2.9

Table of Contents

Table of Contents

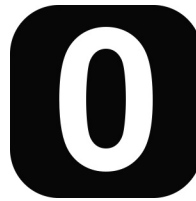
Table of Contents

Library Reference

Table of Contents

Table of Contents

Introduction



This programmers manual describes Mini-XML version 2.9, a small XML parsing library that you can use to read and write XML data files in your C and C++ applications.

Mini-XML was initially developed for the Gutenprint project to replace the rather large and unwieldy `libxml2` library with something substantially smaller and easier-to-use. It all began one morning in June of 2003 when Robert posted the following sentence to the developer's list:

> *It's bad enough that we require libxml2, but rolling our own XML parser is a bit more than we can handle.*

I then replied with:

> *Given the limited scope of what you use in XML, it should be trivial to code a mini-XML API in a few hundred lines of code.*

I took my own challenge and coded furiously for two days to produced the initial public release of Mini-XML, total lines of code: 696. Robert promptly integrated Mini-XML into Gutenprint and removed libxml2.

Thanks to lots of feedback and support from various developers, Mini-XML has evolved since then to provide a more complete XML implementation and now stands at a whopping 3,792 lines of code, compared to 140,410 lines of code for libxml2 version 2.9.1.

Aside from Gutenprint, Mini-XML is used for the following projects/software applications:

- CUPS
- ZynAddSubFX

Please file a bug on msweet.org if you would like your project added or removed from this list, or if you have any comments/quotes you would like me to publish about your experiences with Mini-XML.

Organization of This Document

This manual is organized into the following chapters and appendices:

- Chapter 1, "Building, Installing, and Packaging Mini-XML", provides compilation, installation, and packaging instructions for Mini-XML.
- Chapter 2, "Getting Started with Mini-XML", shows how to use the Mini-XML library in your programs.
- Chapter 3, "More Mini-XML Programming Techniques", shows additional ways to use the Mini-XML library.
- Chapter 4, "Using the mxmldoc Utility", describes how to use the `mxmldoc(1)` program to generate software documentation.
- Appendix A, "Mini-XML License", provides the terms and conditions for using and distributing Mini-XML.
- Appendix B, "Release Notes", lists the changes in each release of Mini-XML.
- Appendix C, "Library Reference", contains a complete reference for Mini-XML, generated by `mxmldoc`.
- Appendix D, "XML Schema", shows the XML schema used for the XML files produced by `mxmldoc`.

Notation Conventions

Various font and syntax conventions are used in this guide. Examples and their meanings and uses are explained below:

`mxmldoc`
`mxmldoc(1)`
: The names of commands; the first mention of a command or function in a chapter is followed by a manual page section number.

/var
/etc/hosts
: File and directory names.

```
Request ID is Printer-123
```
: Screen output.

`lp -d printer filename ENTER`
: Literal user input; special keys like **`ENTER`** are in ALL CAPS.

12.3
: Numbers in the text are written using the period (.) to indicate the decimal point.

Abbreviations

The following abbreviations are used throughout this manual:

Gb
: Gigabytes, or 1073741824 bytes

kb
: Kilobytes, or 1024 bytes

Mb
: Megabytes, or 1048576 bytes

UTF-8, UTF-16
: Unicode Transformation Format, 8-bit or 16-bit

W3C
: World Wide Web Consortium

XML
: Extensible Markup Language

Other References

The Unicode Standard, Version 4.0, Addison-Wesley, ISBN 0-321-18578-1

The definition of the Unicode character set which is used for XML.

Extensible Markup Language (XML) 1.0 (Third Edition)

The XML specification from the World Wide Web Consortium (W3C)

Legal Stuff

Building, Installing, and Packaging Mini-XML

This chapter describes how to build, install, and package Mini-XML on your system from the source archive. You will need an ANSI/ISO-C compatible compiler to build Mini-XML - GCC works, as do most vendors' C compilers. If you are building Mini-XML on Windows, we recommend using the Visual C++ environment with the supplied solution file. For other operating systems, you'll need a POSIX-compatible shell and `make` program in addition to the C compiler.

Compiling Mini-XML

Mini-XML comes with both an autoconf-based configure script and a Visual C++ solution that can be used to compile the library and associated tools.

Compiling with Visual C++

Open the *mxml.sln* solution in the *vcnet* folder. Choose the desired build configuration, "Debug" (the default) or "Release", and then choose *Build Solution* from the *Build* menu.

Compiling with Command-Line Tools

Type the following command to configure the Mini-XML source code for your system:

```
./configure ENTER
```

The default install prefix is */usr/local*, which can be overridden using the **--prefix** option:

```
./configure --prefix=/foo ENTER
```

Other configure options can be found using the **--help** option:

```
./configure --help ENTER
```

Once you have configured the software, use the `make(1)` program to do the build and run the test program to verify that things are working, as follows:

```
make ENTER
```

Installing Mini-XML

If you are using Visual C++, copy the *mxml.lib* and and *mxml.h* files to the Visual C++ *lib* and *include directories, respectively.*

Otherwise, use the `make` command with the **`install`** target to install Mini-XML in the configured directories:

```
make install ENTER
```

Creating Mini-XML Packages

Mini-XML includes two files that can be used to create binary packages. The first file is *mxml.spec* which is used by the `rpmbuild(8)` software to create Red Hat Package Manager ("RPM") packages which are commonly used on Linux. Since `rpmbuild` wants to compile the software on its own, you can provide it with the Mini-XML tar file to build the package:

```
rpmbuild -ta mxml-version.tar.gz ENTER
```

The second file is *mxml.list* which is used by the `epm(1)` program to create software packages in a variety of formats. The `epm` program is available from the following URL:

```
http://www.epmhome.org/
```

Use the `make` command with the **`epm`** target to create portable and native packages for your system:

```
make epm ENTER
```

The packages are stored in a subdirectory named *dist* for your convenience. The portable packages utilize scripts and tar files to install the software on the target system. After extracting the package archive, use the *mxml.install* script to install the software.

The native packages will be in the local OS's native format: RPM for Red Hat Linux, DPKG for Debian Linux, PKG for Solaris, and so forth. Use the corresponding commands to install the native packages.

Getting Started with Mini-XML

This chapter describes how to write programs that use Mini-XML to access data in an XML file. Mini-XML provides the following functionality:

- Functions for creating and managing XML documents in memory.
- Reading of UTF-8 and UTF-16 encoded XML files and strings.
- Writing of UTF-8 encoded XML files and strings.
- Support for arbitrary element names, attributes, and attribute values with no preset limits, just available memory.
- Support for integer, real, opaque ("CDATA"), and text data types in "leaf" nodes.
- "Find", "index", and "walk" functions for easily accessing data in an XML document.

Mini-XML doesn't do validation or other types of processing on the data based upon schema files or other sources of definition information, nor does it support character entities other than those required by the XML specification.

The Basics

Mini-XML provides a single header file which you include:

```
#include <mxml.h>
```

The Mini-XML library is included with your program using the **-lmxml** option:

```
gcc -o myprogram myprogram.c -lmxml ENTER
```

If you have the `pkg-config(1)` software installed, you can use it to determine the proper compiler and linker options for your installation:

```
pkg-config --cflags mxml ENTER
pkg-config --libs mxml ENTER
```

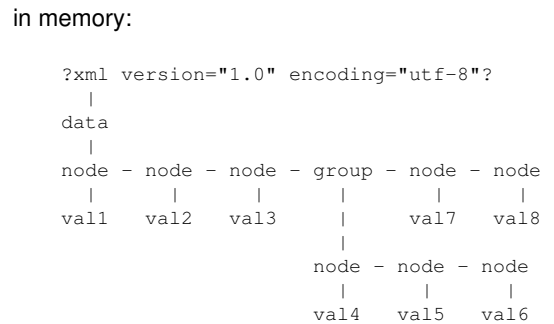
Nodes

Every piece of information in an XML file is stored in memory in "nodes". Nodes are defined by the `mxml_node_t` structure. Each node has a typed value, optional user data, a parent node, sibling nodes (previous and next), and potentially child nodes.

For example, if you have an XML file like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<data>
    <node>val1</node>
    <node>val2</node>
    <node>val3</node>
    <group>
        <node>val4</node>
        <node>val5</node>
        <node>val6</node>
    </group>
    <node>val7</node>
    <node>val8</node>
</data>
```

the node tree for the file would look like the following in memory:



where "-" is a pointer to the sibling node and "|" is a pointer to the first child or parent node.

The `mxmlGetType` function gets the type of a node, one of `MXML_CUSTOM`, `MXML_ELEMENT`, `MXML_INTEGER`, `MXML_OPAQUE`, `MXML_REAL`, or `MXML_TEXT`. The parent and sibling nodes are accessed using the `mxmlGetParent`, `mxmlGetNext`, and `mxmlGetPrevious` functions. The `mxmlGetUserData` function gets any user data associated with the node.

CDATA Nodes

CDATA (`MXML_ELEMENT`) nodes are created using the `mxmlNewCDATA` function. The `mxmlGetCDATA` function retrieves the CDATA string pointer for a node.

> **Note:**
>
> CDATA nodes are currently stored in memory as special elements. This will be changed in a future major release of Mini-XML.

Custom Nodes

Custom (`MXML_CUSTOM`) nodes are created using the `mxmlNewCustom` function or using a custom load callback specified using the `mxmlSetCustomHandlers` function. The `mxmlGetCustom` function retrieves the custom value pointer for a node.

Comment Nodes

Comment (`MXML_ELEMENT`) nodes are created using the `mxmlNewElement` function. The `mxmlGetElement` function retrieves the comment string pointer for a node, including the surrounding "!--" and "--" characters.

> **Note:**
>
> Comment nodes are currently stored in memory as special elements. This will be changed in a future major release of Mini-XML.

Element Nodes

Element (MXML_ELEMENT) nodes are created using the mxmlNewElement function. The mxmlGetElement function retrieves the element name, the mxmlElementGetAttr function retrieves the value string for a named attribute associated with the element, and the mxmlGetFirstChild and mxmlGetLastChild functions retrieve the first and last child nodes for the element, respectively.

Integer Nodes

Integer (MXML_INTEGER) nodes are created using the mxmlNewInteger function. The mxmlGetInteger function retrieves the integer value for a node.

Opaque Nodes

Opaque (MXML_OPAQUE) nodes are created using the mxmlNewOpaque function. The mxmlGetOpaque function retrieves the opaque string pointer for a node. Opaque nodes are like string nodes but preserve all whitespace between nodes.

Text Nodes

Text (MXML_TEXT) nodes are created using the mxmlNewText and mxmlNewTextf functions. Each text node consists of a text string and (leading) whitespace value - the mxmlGetText function retrieves the text string pointer and whitespace value for a node.

Processing Instruction Nodes

Processing instruction (`MXML_ELEMENT`) nodes are created using the `mxmlNewElement` function. The `mxmlGetElement` function retrieves the processing instruction string for a node, including the surrounding "?" characters.

> **Note:**
>
> Processing instruction nodes are currently stored in memory as special elements. This will be changed in a future major release of Mini-XML.

Real Number Nodes

Real number (`MXML_REAL`) nodes are created using the `mxmlNewReal` function. The `mxmlGetReal` function retrieves the CDATA string pointer for a node.

XML Declaration Nodes

XML declaration (`MXML_ELEMENT`) nodes are created using the `mxmlNewXML` function. The `mxmlGetElement` function retrieves the XML declaration string for a node, including the surrounding "?" characters.

> **Note:**
>
> XML declaration nodes are currently stored in memory as special elements. This will be changed in a future major release of Mini-XML.

Creating XML Documents

You can create and update XML documents in memory using the various `mxmlNew` functions. The following code will create the XML document described in the previous section:

```
mxml_node_t *xml;    /* <?xml ... ?> */
mxml_node_t *data;   /* <data> */
mxml_node_t *node;   /* <node> */
mxml_node_t *group;  /* <group> */

xml = mxmlNewXML("1.0");

data = mxmlNewElement(xml, "data");

    node = mxmlNewElement(data, "node");
    mxmlNewText(node, 0, "val1");
    node = mxmlNewElement(data, "node");
    mxmlNewText(node, 0, "val2");
    node = mxmlNewElement(data, "node");
    mxmlNewText(node, 0, "val3");

    group = mxmlNewElement(data, "group");

        node = mxmlNewElement(group, "node");
        mxmlNewText(node, 0, "val4");
        node = mxmlNewElement(group, "node");
        mxmlNewText(node, 0, "val5");
        node = mxmlNewElement(group, "node");
        mxmlNewText(node, 0, "val6");

    node = mxmlNewElement(data, "node");
    mxmlNewText(node, 0, "val7");
    node = mxmlNewElement(data, "node");
    mxmlNewText(node, 0, "val8");
```

We start by creating the declaration node common to all XML files using the `mxmlNewXML` function:

```
xml = mxmlNewXML("1.0");
```

We then create the `<data>` node used for this document using the `mxmlNewElement` function. The first argument specifies the parent node (`xml`) while the second specifies the element name (`data`):

```
data = mxmlNewElement(xml, "data");
```

Each `<node>...</node>` in the file is created using the `mxmlNewElement` and `mxmlNewText` functions. The first argument of `mxmlNewText` specifies the parent node (`node`). The second argument specifies whether whitespace appears before the text - 0 or false in this case. The last argument specifies the actual text to add:

```
node = mxmlNewElement(data, "node");
mxmlNewText(node, 0, "val1");
```

The resulting in-memory XML document can then be saved or processed just like one loaded from disk or a string.

Loading XML

You load an XML file using the `mxmlLoadFile` function:

```
FILE *fp;
mxml_node_t *tree;

fp = fopen("filename.xml", "r");
tree = mxmlLoadFile(NULL, fp,
                    MXML_TEXT_CALLBACK);
fclose(fp);
```

The first argument specifies an existing XML parent node, if any. Normally you will pass `NULL` for this argument unless you are combining multiple XML sources. The XML file must contain a complete XML document including the `?xml` element if the parent node is `NULL`.

The second argument specifies the stdio file to read from, as opened by `fopen()` or `popen()`. You can also use `stdin` if you are implementing an XML filter program.

The third argument specifies a callback function which returns the value type of the immediate children for a new element node: `MXML_CUSTOM`, `MXML_IGNORE`, `MXML_INTEGER`, `MXML_OPAQUE`, `MXML_REAL`, or `MXML_TEXT`. Load callbacks are described in detail in Chapter 3. The example code uses the `MXML_TEXT_CALLBACK` constant which specifies that all data nodes in the document contain whitespace-separated text values. Other standard callbacks include `MXML_IGNORE_CALLBACK`, `MXML_INTEGER_CALLBACK`, `MXML_OPAQUE_CALLBACK`, and `MXML_REAL_CALLBACK`.

The `mxmlLoadString` function loads XML node trees from a string:

```
char buffer[8192];
mxml_node_t *tree;

...
tree = mxmlLoadString(NULL, buffer,
                      MXML_TEXT_CALLBACK);
```

The first and third arguments are the same as used for `mxmlLoadFile()`. The second argument specifies the string or character buffer to load and must be a complete XML document including the `?xml` element if the parent node is `NULL`.

Saving XML

You save an XML file using the `mxmlSaveFile` function:

```
FILE *fp;
mxml_node_t *tree;

fp = fopen("filename.xml", "w");
mxmlSaveFile(tree, fp, MXML_NO_CALLBACK);
fclose(fp);
```

The first argument is the XML node tree to save. It should normally be a pointer to the top-level `?xml` node in your XML document.

The second argument is the stdio file to write to, as opened by `fopen()` or `popen()`. You can also use `stdout` if you are implementing an XML filter program.

The third argument is the whitespace callback to use when saving the file. Whitespace callbacks are covered in detail in Chapter 3. The previous example code uses the `MXML_NO_CALLBACK` constant to specify that no special whitespace handling is required.

The `mxmlSaveAllocString`, and `mxmlSaveString` functions save XML node trees to strings:

```
char buffer[8192];
char *ptr;
mxml_node_t *tree;

...
mxmlSaveString(tree, buffer, sizeof(buffer),
               MXML_NO_CALLBACK);

...
ptr = mxmlSaveAllocString(tree, MXML_NO_CALLBACK);
```

The first and last arguments are the same as used for `mxmlSaveFile()`. The `mxmlSaveString` function takes pointer and size arguments for saving the XML document to a fixed-size buffer, while `mxmlSaveAllocString()` returns a string buffer that was allocated using `malloc()`.

Controlling Line Wrapping

When saving XML documents, Mini-XML normally wraps output lines at column 75 so that the text is readable in terminal windows. The `mxmlSetWrapMargin` function overrides the default wrap margin:

```
/* Set the margin to 132 columns */
mxmlSetWrapMargin(132);

/* Disable wrapping */
mxmlSetWrapMargin(0);
```

Memory Management

Once you are done with the XML data, use the `mxmlDelete` function to recursively free the memory that is used for a particular node or the entire tree:

```
mxmlDelete(tree);
```

You can also use reference counting to manage memory usage. The `mxmlRetain` and `mxmlRelease` functions increment and decrement a node's use count, respectively. When the use count goes to 0, `mxmlRelease` will automatically call `mxmlDelete` to actually free the memory used by the node tree. New nodes automatically start with a use count of 1.

Finding and Iterating Nodes

The `mxmlWalkPrev` and `mxmlWalkNext`functions can be used to iterate through the XML node tree:

```
mxml_node_t *node;

node = mxmlWalkPrev(current, tree,
                    MXML_DESCEND);

node = mxmlWalkNext(current, tree,
                    MXML_DESCEND);
```

In addition, you can find a named element/node using the `mxmlFindElement` function:

```
mxml_node_t *node;

node = mxmlFindElement(tree, tree, "name",
                       "attr", "value",
                       MXML_DESCEND);
```

The `name`, `attr`, and `value` arguments can be passed as `NULL` to act as wildcards, e.g.:

```
/* Find the first "a" element */
node = mxmlFindElement(tree, tree, "a",
                       NULL, NULL,
                       MXML_DESCEND);

/* Find the first "a" element with "href"
   attribute */
node = mxmlFindElement(tree, tree, "a",
                       "href", NULL,
                       MXML_DESCEND);
```

```
/* Find the first "a" element with "href"
   to a URL */
node = mxmlFindElement(tree, tree, "a",
                       "href",
                       "http://www.easysw.com/",
                       MXML_DESCEND);

/* Find the first element with a "src"
   attribute */
node = mxmlFindElement(tree, tree, NULL,
                       "src", NULL,
                       MXML_DESCEND);

/* Find the first element with a "src"
   = "foo.jpg" */
node = mxmlFindElement(tree, tree, NULL,
                       "src", "foo.jpg",
                       MXML_DESCEND);
```

You can also iterate with the same function:

```
mxml_node_t *node;

for (node = mxmlFindElement(tree, tree,
                            "name",
                            NULL, NULL,
                            MXML_DESCEND);
     node != NULL;
     node = mxmlFindElement(node, tree,
                            "name",
                            NULL, NULL,
                            MXML_DESCEND))
{
  ... do something ...
}
```

The MXML_DESCEND argument can actually be one of three constants:

- MXML_NO_DESCEND means to not to look at any child nodes in the element hierarchy, just look at siblings at the same level or parent nodes until the top node or top-of-tree is reached.

  The previous node from "group" would be the "node" element to the left, while the next node from "group" would be the "node" element to the right.

- MXML_DESCEND_FIRST means that it is OK to descend to the first child of a node, but not to descend further when searching. You'll normally use this when iterating through direct children of a parent node, e.g. all of the "node" and "group" elements under the "?xml" parent node in the example above.

  This mode is only applicable to the search function; the walk functions treat this as MXML_DESCEND since every call is a first time.

- MXML_DESCEND means to keep descending until you hit the bottom of the tree. The previous node from "group" would be the "val3" node and the next node would be the first node element under "group".

  If you were to walk from the root node "?xml" to the end of the tree with mxmlWalkNext(), the order would be:

```
?xml data node val1 node val2 node val3
group node val4 node val5 node val6 node
val7 node val8
```

If you started at "val8" and walked using `mxmlWalkPrev()`, the order would be reversed, ending at "?xml".

Finding Specific Nodes

You can find specific nodes in the tree using the `mxmlFindPath`, for example:

```
mxml_node_t *value;

value = mxmlFindPath(tree, "path/to/*/foo/bar");
```

The second argument is a "path" to the parent node. Each component of the path is separated by a slash (/) and represents a named element in the document tree or a wildcard (*) path representing 0 or more intervening nodes.

More Mini-XML Programming Techniques

This chapter shows additional ways to use the Mini-XML library in your programs.

Load Callbacks

Chapter 2 introduced the `mxmlLoadFile()` and `mxmlLoadString()` functions. The last argument to these functions is a callback function which is used to determine the value type of each data node in an XML document.

Mini-XML defines several standard callbacks for simple XML data files:

- `MXML_INTEGER_CALLBACK` - All data nodes contain whitespace-separated integers.

- MXML_OPAQUE_CALLBACK - All data nodes contain opaque strings ("CDATA").
- MXML_REAL_CALLBACK - All data nodes contain whitespace-separated floating-point numbers.
- MXML_TEXT_CALLBACK - All data nodes contain whitespace-separated strings.

You can provide your own callback functions for more complex XML documents. Your callback function will receive a pointer to the current element node and must return the value type of the immediate children for that element node: MXML_INTEGER, MXML_OPAQUE, MXML_REAL, or MXML_TEXT. The function is called *after* the element and its attributes have been read, so you can look at the element name, attributes, and attribute values to determine the proper value type to return.

The following callback function looks for an attribute named "type" or the element name to determine the value type for its child nodes:

```
mxml_type_t
type_cb(mxml_node_t *node)
{
  const char *type;

 /*
  * You can lookup attributes and/or use the
  * element name, hierarchy, etc...
  */

  type = mxmlElementGetAttr(node, "type");
  if (type == NULL)
    type = mxmlGetElement(node);

  if (!strcmp(type, "integer"))
    return (MXML_INTEGER);
  else if (!strcmp(type, "opaque"))
```

```
    return (MXML_OPAQUE);
  else if (!strcmp(type, "real"))
    return (MXML_REAL);
  else
    return (MXML_TEXT);
}
```

To use this callback function, simply use the name when you call any of the load functions:

```
FILE *fp;
mxml_node_t *tree;

fp = fopen("filename.xml", "r");
tree = mxmlLoadFile(NULL, fp, type_cb);
fclose(fp);
```

Save Callbacks

Chapter 2 also introduced the `mxmlSaveFile()`, `mxmlSaveString()`, and `mxmlSaveAllocString()` functions. The last argument to these functions is a callback function which is used to automatically insert whitespace in an XML document.

Your callback function will be called up to four times for each element node with a pointer to the node and a "where" value of `MXML_WS_BEFORE_OPEN`, `MXML_WS_AFTER_OPEN`, `MXML_WS_BEFORE_CLOSE`, or `MXML_WS_AFTER_CLOSE`. The callback function should return `NULL` if no whitespace should be added and the string to insert (spaces, tabs, carriage returns, and newlines) otherwise.

The following whitespace callback can be used to add whitespace to XHTML output to make it more readable in a standard text editor:

```
const char *
whitespace_cb(mxml_node_t *node,
              int where)
{
  const char *name;

 /*
  * We can conditionally break to a new line
  * before or after any element. These are
  * just common HTML elements...
  */

  name = mxmlGetElement(node);

  if (!strcmp(name, "html") ||
      !strcmp(name, "head") ||
      !strcmp(name, "body") ||
      !strcmp(name, "pre") ||
      !strcmp(name, "p") ||
      !strcmp(name, "h1") ||
      !strcmp(name, "h2") ||
      !strcmp(name, "h3") ||
      !strcmp(name, "h4") ||
      !strcmp(name, "h5") ||
      !strcmp(name, "h6"))
  {
   /*
    * Newlines before open and after
    * close...
    */

    if (where == MXML_WS_BEFORE_OPEN ||
        where == MXML_WS_AFTER_CLOSE)
      return ("\n");
  }
  else if (!strcmp(name, "dl") ||
           !strcmp(name, "ol") ||
           !strcmp(name, "ul"))
  {
   /*
    * Put a newline before and after list
    * elements...
    */
```

```
    return ("\n");
  }
  else if (!strcmp(name, "dd") ||
           !strcmp(name, "dt") ||
           !strcmp(name, "li"))
  {
   /*
    * Put a tab before <li>'s, * <dd>'s,
    * and <dt>'s, and a newline after them...
    */

    if (where == MXML_WS_BEFORE_OPEN)
      return ("\t");
    else if (where == MXML_WS_AFTER_CLOSE)
      return ("\n");
  }

 /*
  * Return NULL for no added whitespace...
  */

  return (NULL);
}
```

To use this callback function, simply use the name when you call any of the save functions:

```
FILE *fp;
mxml_node_t *tree;

fp = fopen("filename.xml", "w");
mxmlSaveFile(tree, fp, whitespace_cb);
fclose(fp);
```

Custom Data Types

Mini-XML supports custom data types via global load and save callbacks. Only a single set of callbacks can be active at any time, however your callbacks can store additional information in order to support multiple custom data types as needed. The `MXML_CUSTOM` node type identifies custom data nodes.

The load callback receives a pointer to the current data node and a string of opaque character data from the XML source with character entities converted to the corresponding UTF-8 characters. For example, if we wanted to support a custom date/time type whose value is encoded as "yyyy-mm-ddThh:mm:ssZ" (ISO format), the load callback would look like the following:

```
typedef struct
{
  unsigned      year,    /* Year */
                month,   /* Month */
                day,     /* Day */
                hour,    /* Hour */
                minute,  /* Minute */
                second;  /* Second */
  time_t        unix;    /* UNIX time */
} iso_date_time_t;

int
load_custom(mxml_node_t *node,
            const char *data)
{
  iso_date_time_t *dt;
  struct tm tmdata;

 /*
  * Allocate data structure...
  */
```

```
  dt = calloc(1, sizeof(iso_date_time_t));

 /*
  * Try reading 6 unsigned integers from the
  * data string...
  */

  if (sscanf(data, "%u-%u-%uT%u:%u:%uZ",
             &(dt->year), &(dt->month),
             &(dt->day), &(dt->hour),
             &(dt->minute),
             &(dt->second)) != 6)
  {
   /*
    * Unable to read numbers, free the data
    * structure and return an error...
    */

    free(dt);

    return (-1);
  }

 /*
  * Range check values...
  */

  if (dt->month <1 || dt->month > 12 ||
      dt->day  <1 || dt->day > 31 ||
      dt->hour  <0 || dt->hour > 23 ||
      dt->minute  <0 || dt->minute > 59 ||
      dt->second  <0 || dt->second > 59)
  {
   /*
    * Date information is out of range...
    */

    free(dt);

    return (-1);
  }
```

```
  /*
   * Convert ISO time to UNIX time in
   * seconds...
   */

   tmdata.tm_year = dt->year - 1900;
   tmdata.tm_mon  = dt->month - 1;
   tmdata.tm_day  = dt->day;
   tmdata.tm_hour = dt->hour;
   tmdata.tm_min  = dt->minute;
   tmdata.tm_sec  = dt->second;

   dt->unix = gmtime(&tmdata);

  /*
   * Assign custom node data and destroy
   * function pointers...
   */

   mxmlSetCustom(node, data, destroy);

  /*
   * Return with no errors...
   */

   return (0);
}
```

The function itself can return 0 on success or -1 if it is unable to decode the custom data or the data contains an error. Custom data nodes contain a `void` pointer to the allocated custom data for the node and a pointer to a destructor function which will free the custom data when the node is deleted.

The save callback receives the node pointer and returns an allocated string containing the custom data value. The following save callback could be used for our ISO date/time type:

```
char *
save_custom(mxml_node_t *node)
{
  char data[255];
  iso_date_time_t *dt;


  dt = (iso_date_time_t *)mxmlGetCustom(node);

  snprintf(data, sizeof(data),
           "%04u-%02u-%02uT%02u:%02u:%02uZ",
           dt->year, dt->month, dt->day,
           dt->hour, dt->minute, dt->second);

  return (strdup(data));
}
```

You register the callback functions using the `mxmlSetCustomHandlers()` function:

```
mxmlSetCustomHandlers(load_custom,
                      save_custom);
```

Changing Node Values

All of the examples so far have concentrated on creating and loading new XML data nodes. Many applications, however, need to manipulate or change the nodes during their operation, so Mini-XML provides functions to change node values safely and without leaking memory.

Existing nodes can be changed using the `mxmlSetElement()`, `mxmlSetInteger()`, `mxmlSetOpaque()`, `mxmlSetReal()`, `mxmlSetText()`, and `mxmlSetTextf()` functions. For example, use the following function call to change a text node to contain the text "new" with leading whitespace:

```
mxml_node_t *node;

mxmlSetText(node, 1, "new");
```

Formatted Text

The `mxmlNewTextf()` and `mxmlSetTextf()` functions create and change text nodes, respectively, using `printf`-style format strings and arguments. For example, use the following function call to create a new text node containing a constructed filename:

```
mxml_node_t *node;

node = mxmlNewTextf(node, 1, "%s/%s",
                    path, filename);
```

Indexing

Mini-XML provides functions for managing indices of nodes. The current implementation provides the same functionality as `mxmlFindElement()`. The advantage of using an index is that searching and enumeration of elements is significantly faster. The only disadvantage is that each index is a static snapshot of the XML document, so indices are not well suited to XML data that is updated more often than it is searched. The overhead of creating an index is approximately equal to walking the XML document tree. Nodes in the index are sorted by element name and attribute value.

Indices are stored in `mxml_index_t` structures. The `mxmlIndexNew()` function creates a new index:

```
mxml_node_t *tree;
mxml_index_t *ind;

ind = mxmlIndexNew(tree, "element",
                   "attribute");
```

The first argument is the XML node tree to index. Normally this will be a pointer to the `?xml` element.

The second argument contains the element to index; passing `NULL` indexes all element nodes alphabetically.

The third argument contains the attribute to index; passing `NULL` causes only the element name to be indexed.

Once the index is created, the `mxmlIndexEnum()`, `mxmlIndexFind()`, and `mxmlIndexReset()` functions

are used to access the nodes in the index. The `mxmlIndexReset()` function resets the "current" node pointer in the index, allowing you to do new searches and enumerations on the same index. Typically you will call this function prior to your calls to `mxmlIndexEnum()` and `mxmlIndexFind()`.

The `mxmlIndexEnum()` function enumerates each of the nodes in the index and can be used in a loop as follows:

```
mxml_node_t *node;

mxmlIndexReset(ind);

while ((node = mxmlIndexEnum(ind)) != NULL)
{
  // do something with node
}
```

The `mxmlIndexFind()` function locates the next occurrence of the named element and attribute value in the index. It can be used to find all matching elements in an index, as follows:

```
mxml_node_t *node;

mxmlIndexReset(ind);

while ((node = mxmlIndexFind(ind, "element",
                             "attr-value"))
             != NULL)
{
  // do something with node
}
```

The second and third arguments represent the element name and attribute value, respectively. A `NULL` pointer is used to return all elements or attributes

in the index. Passing `NULL` for both the element name and attribute value is equivalent to calling `mxmlIndexEnum`.

When you are done using the index, delete it using the `mxmlIndexDelete()` function:

```
mxmlIndexDelete(ind);
```

SAX (Stream) Loading of Documents

Mini-XML supports an implementation of the Simple API for XML (SAX) which allows you to load and process an XML document as a stream of nodes. Aside from allowing you to process XML documents of any size, the Mini-XML implementation also allows you to retain portions of the document in memory for later processing.

The `mxmlSAXLoadFd`, `mxmlSAXLoadFile`, and `mxmlSAXLoadString` functions provide the SAX loading APIs. Each function works like the corresponding `mxmlLoad` function but uses a callback to process each node as it is read.

The callback function receives the node, an event code, and a user data pointer you supply:

```
void
sax_cb(mxml_node_t *node,
       mxml_sax_event_t event,
       void *data)
{
  ... do something ...
}
```

The event will be one of the following:

- MXML_SAX_CDATA - CDATA was just read
- MXML_SAX_COMMENT - A comment was just read
- MXML_SAX_DATA - Data (custom, integer, opaque, real, or text) was just read
- MXML_SAX_DIRECTIVE - A processing directive was just read
- MXML_SAX_ELEMENT_CLOSE - A close element was just read (</element>)
- MXML_SAX_ELEMENT_OPEN - An open element was just read (<element>)

Elements are *released* after the close element is processed. All other nodes are released after they are processed. The SAX callback can *retain* the node using the mxmlRetain function. For example, the following SAX callback will retain all nodes, effectively simulating a normal in-memory load:

```
void
sax_cb(mxml_node_t *node,
       mxml_sax_event_t event,
       void *data)
{
  if (event != MXML_SAX_ELEMENT_CLOSE)
    mxmlRetain(node);
}
```

More typically the SAX callback will only retain a small portion of the document that is needed for post-processing. For example, the following SAX callback will retain the title and headings in an XHTML file. It also retains the (parent) elements like <html>, <head>, and <body>, and processing directives like <?xml ... ?> and <!DOCTYPE ... >:

```
void
sax_cb(mxml_node_t *node,
       mxml_sax_event_t event,
       void *data)
{
  if (event == MXML_SAX_ELEMENT_OPEN)
  {
   /*
    * Retain headings and titles...
    */

    char *name = mxmlGetElement(node);

    if (!strcmp(name, "html") ||
        !strcmp(name, "head") ||
        !strcmp(name, "title") ||
        !strcmp(name, "body") ||
        !strcmp(name, "h1") ||
        !strcmp(name, "h2") ||
        !strcmp(name, "h3") ||
        !strcmp(name, "h4") ||
        !strcmp(name, "h5") ||
        !strcmp(name, "h6"))
      mxmlRetain(node);
  }
  else if (event == MXML_SAX_DIRECTIVE)
    mxmlRetain(node);
  else if (event == MXML_SAX_DATA)
  {
    if (mxmlGetRefCount(mxmlGetParent(node)) > 1)
    {
     /*
      * If the parent was retained, then retain
      * this data node as well.
      */

      mxmlRetain(node);
    }
  }
}
```

The resulting skeleton document tree can then be

searched just like one loaded using the `mxmlLoad` functions. For example, a filter that reads an XHTML document from stdin and then shows the title and headings in the document would look like:

```
mxml_node_t *doc, *title, *body, *heading;

doc = mxmlSAXLoadFd(NULL, 0,
                    MXML_TEXT_CALLBACK,
                    sax_cb, NULL);

title = mxmlFindElement(doc, doc, "title",
                        NULL, NULL,
                        MXML_DESCEND);

if (title)
  print_children(title);

body = mxmlFindElement(doc, doc, "body",
                       NULL, NULL,
                       MXML_DESCEND);

if (body)
{
  for (heading = mxmlGetFirstChild(body);
       heading;
       heading = mxmlGetNextSibling(heading))
    print_children(heading);
}
```

Using the mxmldoc Utility

This chapter describes how to use `mxmldoc(1)` program to automatically generate documentation from C and C++ source files.

The Basics

Originally developed to generate the Mini-XML and CUPS API documentation, `mxmldoc` is now a general-purpose utility which scans C and C++ source files to produce HTML and man page documentation along with an XML file representing the functions, types, and definitions in those source files. Unlike popular documentation generators like Doxygen or Javadoc, `mxmldoc` uses in-line comments rather than comment headers, allowing for more "natural" code documentation.

By default, `mxmldoc` produces HTML documentation. For example, the following command will scan all of the C source and header files in the current directory and produce a HTML documentation file called *filename.html*:

```
mxmldoc *.h *.c >filename.html ENTER
```

You can also specify an XML file to create which contains all of the information from the source files. For example, the following command creates an XML file called *filename.xml* in addition to the HTML file:

```
mxmldoc filename.xml *.h *.c >filename.html ENTER
```

The `--no-output` option disables the normal HTML output:

```
mxmldoc --no-output filename.xml *.h *.c ENTER
```

You can then run `mxmldoc` again with the XML file alone to generate the HTML documentation:

```
mxmldoc filename.xml >filename.html ENTER
```

Creating Man Pages

The `--man filename` option tells `mxmldoc` to create a man page instead of HTML documentation, for example:

```
mxmldoc --man filename filename.xml \
   >filename.man ENTER

mxmldoc --man filename *.h *.c \
   >filename.man ENTER
```

Creating Xcode Documentation Sets

The `--docset directory.docset` option tells `mxmldoc` to create an Xcode documentation set containing the HTML documentation, for example:

```
mxmldoc --docset foo.docset *.h *.c foo.xml ENTER
```

Xcode documentation sets can only be built on Mac OS X with Xcode 3.0 or higher installed.

Commenting Your Code

As noted previously, `mxmldoc` looks for in-line comments to describe the functions, types, and constants in your code. `Mxmldoc` will document all public names it finds in your source files - any names starting with the underscore character (_) or names that are documented with the @private@ directive are treated as private and are not documented.

Comments appearing directly before a function or type definition are used to document that function or type. Comments appearing after argument, definition, return type, or variable declarations are used to document that argument, definition, return type, or variable. For example, the following code excerpt defines a key/value structure and a function that creates a new instance of that structure:

```
/* A key/value pair. This is used with the
   dictionary structure. */

struct keyval
{
  char *key; /* Key string */
  char *val; /* Value string */
```

```
};

/* Create a new key/value pair. */

struct keyval * /* New key/value pair */
new_keyval(
    const char *key, /* Key string */
    const char *val) /* Value string */
{
  ...
}
```

`Mxmldoc` also knows to remove extra asterisks (*) from the comment string, so the comment string:

```
/*
 * Compute the value of PI.
 *
 * The function connects to an Internet server
 * that streams audio of mathematical monks
 * chanting the first 100 digits of PI.
 */
```

will be shown as:

```
Compute the value of PI.

The function connects to an Internet server
that streams audio of mathematical monks
chanting the first 100 digits of PI.
```

Comments can also include the following special `@name ...@` directive strings:

- `@deprecated@` - flags the item as deprecated to discourage its use
- `@private@` - flags the item as private so it will not be included in the documentation
- `@since ...@` - flags the item as new since a particular release. The text following the

`@since` up to the closing `@` is highlighted in the generated documentation, e.g. `@since Mini-XML 2.7@`.

Titles, Sections, and Introductions

`Mxmldoc` also provides options to set the title, section, and introduction text for the generated documentation. The `--title text` option specifies the title for the documentation. The title string is usually put in quotes:

```
mxmldoc filename.xml \
   --title "My Famous Documentation" \
   >filename.html ENTER
```

The `--section name` option specifies the section for the documentation. For HTML documentation, the name is placed in a HTML comment such as:

```
<!-- SECTION: name -->
```

For man pages, the section name is usually just a number ("3"), or a number followed by a vendor name ("3acme"). The section name is used in the `.TH` directive in the man page:

```
.TH mylibrary 3acme "My Title" ...
```

The default section name for man page output is "3". There is no default section name for HTML output.

Finally, the `--intro filename` option specifies a file to embed after the title and section but before the generated documentation. For HTML documentation, the file must consist of valid HTML without the usual

`DOCTYPE`, `html`, and `body` elements. For man page documentation, the file must consist of valid `nroff(1)` text.

Mini-XML License

The Mini-XML library and included programs are provided under the terms of the GNU Library General Public License version 2 (LGPL2) with the following exceptions:

1. Static linking of applications to the Mini-XML library does not constitute a derivative work and does not require the author to provide source code for the application, use the shared Mini-XML libraries, or link their applications against a user-supplied version of Mini-XML.

If you link the application to a modified version of Mini-XML, then the changes to Mini-XML must be provided under the terms of the LGPL2 in sections 1, 2, and 4.

2. You do not have to provide a copy of the Mini-XML license with programs that are linked to the Mini-XML library, nor do you have to identify the Mini-XML license in your program or documentation as required by section 6 of the LGPL2.

GNU LIBRARY GENERAL PUBLIC LICENSE

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

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 Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, 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 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 a program 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.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the

original version, 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 companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker

conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the libary" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library 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 compile 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) 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.

c) 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.

d) 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 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.

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 to 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 Library 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.

> *one line to give the library's name and an idea of what it does.*
> Copyright (C) *year name of author*
>
> 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.

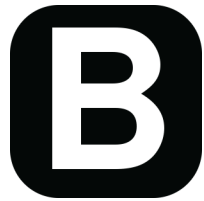
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary.

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.
>
> *signature of Ty Coon*, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

Release Notes



Changes in Mini-XML 2.9

- mxmlLoad* did not correctly load value nodes with MXML_NO_CALLBACK or MXML_TEXT_CALLBACK (Bug #502)

Changes in Mini-XML 2.8

- Now call docsetutil using xcrun on OS X (Bug #458)
- mxmldoc did not escape special HTML characters inside @code foo@ comments.
- Fixed a memory leak in mxmlElementDeleteAttr (Bug #452)
- Added MXML_MAJOR/MINOR_VERSION definitions to mxml.h (Bug $461)

- Fixed a bug reading UTF-16 characters from a file (Bug #454)
- Fixed a memory leak when loading invalid XML (Bug #496)
- Fixed an XML fragment loading problem (Bug #494)

Changes in Mini-XML 2.7

- Added 64-bit configurations to the VC++ project files (STR #129)
- Fixed conformance of mxmldoc's HTML and CSS output.
- Added data accessor ("get") functions and made the mxml_node_t and mxml_index_t structures private but still available in the Mini-XML header to preserve source compatibility (STR #118)
- Updated the source headers to reference the Mini-XML license and its exceptions to the LGPL2 (STR #108)
- Added a new mxmlFindPath() function to find the value node of a named element (STR #110)
- Building a static version of the library did not work on Windows (STR #112)
- The shared library did not include a destructor for the thread- specific data key on UNIX-based operating systems (STR #103)
- mxmlLoad* did not error out on XML with multiple root nodes (STR #101)
- Fixed an issue with the _mxml_vstrdupf function (STR #107)

- mxmlSave* no longer write all siblings of the passed node, just that node and its children (STR #109)

Changes in Mini-XML 2.6

- Documentation fixes (STR #91, STR #92)
- The mxmldoc program did not handle typedef comments properly (STR #72)
- Added support for "long long" printf formats.
- The XML parser now ignores BOMs in UTF-8 XML files (STR #89)
- The mxmldoc program now supports generating Xcode documentation sets.
- mxmlSave*() did not output UTF-8 correctly on some platforms.
- mxmlNewXML() now adds encoding="utf-8" in the ?xml directive to avoid problems with non-conformant XML parsers that assume something other than UTF-8 as the default encoding.
- Wrapping was not disabled when mxmlSetWrapMargin(0) was called, and "<?xml ... ?>" was always followed by a newline (STR #76)
- The mxml.pc.in file was broken (STR #79)
- The mxmldoc program now handles "typedef enum name {} name" correctly (STR #72)

Changes in Mini-XML 2.5

- The mxmldoc program now makes greater use of CSS and supports a --css option to embed an alternate stylesheet.

- The mxmldoc program now supports --header and --footer options to insert documentation content before and after the generated content.
- The mxmldoc program now supports a --framed option to generate framed HTML output.
- The mxmldoc program now creates a table of contents including any headings in the --intro file when generating HTML output.
- The man pages and man page output from mxmldoc did not use "\-" for dashes (STR #68)
- The debug version of the Mini-XML DLL could not be built (STR #65)
- Processing instructions and directives did not work when not at the top level of a document (STR #67)
- Spaces around the "=" in attributes were not supported (STR #67)

Changes in Mini-XML 2.4

- Fixed shared library build problems on HP-UX and Mac OS X.
- The mxmldoc program did not output argument descriptions for functions properly.
- All global settings (custom, error, and entity callbacks and the wrap margin) are now managed separately for each thread.
- Added mxmlElementDeleteAttr() function (STR #59)
- mxmlElementSetAttrf() did not work (STR #57)

- mxmlLoad*() incorrectly treated declarations as parent elements (STR #56)
- mxmlLoad*() incorrectly allowed attributes without values (STR #47)
- Fixed Visual C++ build problems (STR #49)
- mxmlLoad*() did not return NULL when an element contained an error (STR #46)
- Added support for the apos character entity (STR #54)
- Fixed whitespace detection with Unicode characters (STR #48)
- mxmlWalkNext() and mxmlWalkPrev() did not work correctly when called with a node with no children as the top node (STR #53)

Changes in Mini-XML 2.3

- Added two exceptions to the LGPL to support static linking of applications against Mini-XML
- The mxmldoc utility can now generate man pages, too.
- Added a mxmlNewXML() function
- Added a mxmlElementSetAttrf() function (STR #43)
- Added a snprintf() emulation function for the test program (STR #32)
- Added the _CRT_SECURE_NO_DEPRECATE definition when building on VC++ 2005 (STR #36)
- mxmlLoad*() did not detect missing > characters in elements (STR #41)
- mxmlLoad*() did not detect missing close tags at the end of an XML document (STR

#45)
- Added user_data and ref_count members to mxml_node_t structure
- Added mxmlReleaseNode() and mxmlRetainNode() APIs for reference-counted nodes
- Added mxmlSetWrapMargin() to control the wrapping of XML output
- Added conditional check for EINTR error code for certain Windows compilers that do not define it (STR #33)
- The mxmldoc program now generates correct HTML 4.0 output - previously it generated invalid XHTML
- The mxmldoc program now supports "@deprecated@, "@private@", and "@since version@" comments
- Fixed function and enumeration type bugs in mxmldoc
- Fixed the XML schema for mxmldoc
- The mxmldoc program now supports --intro, --section, and --title options
- The mxmlLoad*() functions could leak a node on an error (STR #27)
- The mxml_vsnprintf() function could get in an infinite loop on a buffer overflow (STR #25)
- Added new mxmlNewCDATA() and mxmlSetCDATA() functions to create and set CDATA nodes, which are really just special element nodes
- Added new MXML_IGNORE type and MXML_IGNORE_CB callback to ignore non-element nodes, e.g. whitespace
- mxmlLoad*() did not treat custom data as opaque, so whitespace characters would be lost

Changes in Mini-XML 2.2.2

- mxmlLoad*() did not treat custom data as opaque, so whitespace characters would be lost.

Changes in Mini-XML 2.2.1

- mxmlLoadFd(), mxmlLoadFile(), and mxmlLoadString() now correctly return NULL on error (STR #21)
- mxmlNewInteger(), mxmlNewOpaque(), mxmlNewReal(), mxmlNewText(), and mxmlNewTextf() incorrectly required a parent node (STR #22)
- Fixed an XML output bug in mxmldoc.
- The "make install" target now uses the install command to set the proper permissions on UNIX/Linux/OSX.
- Fixed a MingW/Cygwin compilation problem (STR #18)

Changes in Mini-XML 2.2

- Added shared library support (STR #17)
- mxmlLoad*() now returns an error when an XML stream contains illegal control characters (STR #10)
- mxmlLoad*() now returns an error when an element contains two attributes with the same name in conformance with the XML spec (STR #16)
- Added support for CDATA (STR #14, STR #15)

- Updated comment and processing instruction handling - no entity support per XML specification.
- Added checking for invalid comment termination ("--->" is not allowed)

Changes in Mini-XML 2.1

- Added support for custom data nodes (STR #6)
- Now treat UTF-8 sequences which are longer than necessary as an error (STR #4)
- Fixed entity number support (STR #8)
- Fixed mxmlLoadString() bug with UTF-8 (STR #7)
- Fixed entity lookup bug (STR #5)
- Added mxmlLoadFd() and mxmlSaveFd() functions.
- Fixed multi-word UTF-16 handling.

Changes in Mini-XML 2.0

- New programmers manual.
- Added Visual C++ project files for Microsoft Windows users.
- Added optimizations to mxmldoc, mxmlSaveFile(), and mxmlIndexNew() (STR #2)
- mxmlEntityAddCallback() now returns an integer status (STR #2)
- Added UTF-16 support (input only; all output is UTF-8)
- Added index functions to build a searchable index of XML nodes.

- Added character entity callback interface to support additional character entities beyond those defined in the XHTML specification.
- Added support for XHTML character entities.
- The mxmldoc utility now produces XML output which conforms to an updated XML schema, described in the file "doc/mxmldoc.xsd".
- Changed the whitespace callback interface to return strings instead of a single character, allowing for greater control over the formatting of XML files written using Mini-XML. THIS CHANGE WILL REQUIRE CHANGES TO YOUR 1.x CODE IF YOU USE WHITESPACE CALLBACKS.
- The mxmldoc utility now produces XML output which conforms to an updated XML schema, described in the file "doc/mxmldoc.xsd".
- Changed the whitespace callback interface to return strings instead of a single character, allowing for greater control over the formatting of XML files written using Mini-XML. THIS CHANGE WILL REQUIRE CHANGES TO YOUR 1.x CODE IF YOU USE WHITESPACE CALLBACKS.
- The mxmldoc utility is now capable of documenting C++ classes, functions, and structures, and correctly handles C++ comments.
- Added new modular tests for mxmldoc.
- Updated the mxmldoc output to be more compatible with embedding in manuals produced with HTMLDOC.
- The makefile incorrectly included a "/" separator between the destination path and

install path. This caused problems when building and installing with MingW.

Changes in Mini-XML 1.3

- Fixes for mxmldoc.
- Added support for reading standard HTML entity names.
- mxmlLoadString/File() did not decode character entities in element names, attribute names, or attribute values.
- mxmlLoadString/File() would crash when loading non- conformant XML data under an existing parent (top) node.
- Fixed several bugs in the mxmldoc utility.
- Added new error callback function to catch a variety of errors and log them to someplace other than stderr.
- The mxmlElementSetAttr() function now allows for NULL attribute values.
- The load and save functions now properly handle quoted element and attribute name strings properly, e.g. for !DOCTYPE declarations.

Changes in Mini-XML 1.2

- Added new "set" methods to set the value of a node.
- Added new formatted text methods mxmlNewTextf() and mxmlSetTextf() to create/set a text node value using printf-style formats.
- Added new standard callbacks for use with the mxmlLoad functions.

- Updated the HTML documentation to include examples of the walk and load function output.
- Added --with/without-ansi configure option to control the strdup() function check.
- Added --with/without-snprintf configure option to control the snprintf() and vsnprintf() function checks.

Changes in Mini-XML 1.1.2

- The mxml(3) man page wasn't updated for the string functions.
- mxmlSaveString() returned the wrong number of characters.
- mxml_add_char() updated the buffer pointer in the wrong place.

Changes in Mini-XML 1.1.1

- The private mxml_add_ch() function did not update the start-of-buffer pointer which could cause a crash when using mxmlSaveString().
- The private mxml_write_ws() function called putc() instead of using the proper callback which could cause a crash when using mxmlSaveString().
- Added a mxmlSaveAllocString() convenience function for saving an XML node tree to an allocated string.

Changes in Mini-XML 1.1

- The mxmlLoadFile() function now uses dynamically allocated string buffers for element names, attribute names, and attribute values. Previously they were capped at 16383, 255, and 255 bytes, respectively.
- Added a new mxmlLoadString() function for loading an XML node tree from a string.
- Added a new mxmlSaveString() function for saving an XML node tree to a string.
- Add emulation of strdup() if the local platform does not provide the function.

Changes in Mini-XML 1.0

- The mxmldoc program now handles function arguments, structures, unions, enumerations, classes, and typedefs properly.
- Documentation provided via mxmldoc and more in-line comments in the code.
- Added man pages and packaging files.

Changes in Mini-XML 0.93

- New mxmldoc example program that is also used to create and update code documentation using XML and produce HTML reference pages.
- Added mxmlAdd() and mxmlRemove() functions to add and remove nodes from a tree. This provides more flexibility over

where the nodes are inserted and allows nodes to be moved within the tree as needed.
- mxmlLoadFile() now correctly handles comments.
- mxmlLoadFile() now supports the required "gt", "quot", and "nbsp" character entities.
- mxmlSaveFile() now uses newlines as whitespace when valid to do so.
- mxmlFindElement() now also takes attribute name and attribute value string arguments to limit the search to specific elements with attributes and/or values.

  NULL pointers can be used as "wildcards".
- Added uninstall target to makefile, and auto-reconfig if Makefile.in or configure.in are changed.
- mxmlFindElement(), mxmlWalkNext(), and mxmlWalkPrev() now all provide "descend" arguments to control whether they descend into child nodes in the tree.
- Fixed some whitespace issues in mxmlLoadFile().
- Fixed Unicode output and whitespace issues in mxmlSaveFile().
- mxmlSaveFile() now supports a whitespace callback to provide more human-readable XML output under program control.

Changes in Mini-XML 0.92

- mxmlSaveFile() didn't return a value on success.

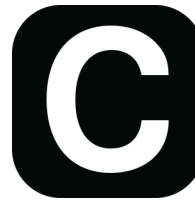
Changes in Mini-XML 0.91

- mxmlWalkNext() would go into an infinite loop.

Changes in Mini-XML 0.9

- Initial public release.

Library Reference



Contents

Functions

mxmlAdd

Add a node to a tree.

```
void mxmlAdd (
   mxml_node_t *parent,
   int where,
   mxml_node_t *child,
   mxml_node_t *node
);
```

Parameters

parent
: Parent node

where
: Where to add, MXML_ADD_BEFORE or MXML_ADD_AFTER

child
: Child node for where or MXML_ADD_TO_PARENT

node
: Node to add

Discussion

Adds the specified node to the parent. If the child argument is not NULL, puts the new node before or after the specified child depending on the value of the where argument. If the child argument is NULL, puts the new node at the beginning of the child list (MXML_ADD_BEFORE) or at the end of the child list (MXML_ADD_AFTER). The constant

MXML_ADD_TO_PARENT can be used to specify a NULL child pointer.

mxmlDelete

Delete a node and all of its children.

```
void mxmlDelete (
    mxml_node_t *node
);
```

Parameters

node
: Node to delete

Discussion

If the specified node has a parent, this function first removes the node from its parent using the mxmlRemove() function.

mxmlElementDeleteAttr

Delete an attribute.

```
void mxmlElementDeleteAttr (
    mxml_node_t *node,
    const char *name
);
```

Parameters

node
: Element

name
: Attribute name

mxmlElementGetAttr

Get an attribute.

```
const char *mxmlElementGetAttr (
   mxml_node_t *node,
   const char *name
);
```

Parameters

node
: Element node

name
: Name of attribute

Return Value

Attribute value or NULL

Discussion

This function returns NULL if the node is not an element or the named attribute does not exist.

mxmlElementSetAttr

Set an attribute.

```
void mxmlElementSetAttr (
   mxml_node_t *node,
   const char *name,
```

```
    const char *value
);
```

Parameters

node
: Element node

name
: Name of attribute

value
: Attribute value

Discussion

If the named attribute already exists, the value of the attribute is replaced by the new string value. The string value is copied into the element node. This function does nothing if the node is not an element.

mxmlElementSetAttrf

Set an attribute with a formatted value.

```
void mxmlElementSetAttrf (
    mxml_node_t *node,
    const char *name,
    const char *format,
    ...
);
```

Parameters

node
: Element node

name
: Name of attribute

format
: Printf-style attribute value

...
: Additional arguments as needed

Discussion

If the named attribute already exists, the value of the attribute is replaced by the new formatted string. The formatted string value is copied into the element node. This function does nothing if the node is not an element.

mxmlEntityAddCallback

Add a callback to convert entities to Unicode.

```
int mxmlEntityAddCallback (
    mxml_entity_cb_t cb
);
```

Parameters

cb
: Callback function to add

Return Value

0 on success, -1 on failure

mxmlEntityGetName

Get the name that corresponds to the character value.

```
const char *mxmlEntityGetName (
    int val
);
```

Parameters

val
: Character value

Return Value

Entity name or NULL

Discussion

If val does not need to be represented by a named entity, NULL is returned.

mxmlEntityGetValue

Get the character corresponding to a named entity.

```
int mxmlEntityGetValue (
    const char *name
);
```

Parameters

name
: Entity name

Return Value

Character value or -1 on error

Discussion

The entity name can also be a numeric constant. -1 is returned if the name is not known.

mxmlEntityRemoveCallback

Remove a callback.

```
void mxmlEntityRemoveCallback (
    mxml_entity_cb_t cb
);
```

Parameters

cb
: Callback function to remove

mxmlFindElement

Find the named element.

```
mxml_node_t *mxmlFindElement (
    mxml_node_t *node,
    mxml_node_t *top,
    const char *name,
    const char *attr,
    const char *value,
    int descend
);
```

Parameters

node
: Current node

top
: Top node

name
: Element name or NULL for any

attr
: Attribute name, or NULL for none

value
: Attribute value, or NULL for any

descend
: Descend into tree - MXML_DESCEND, MXML_NO_DESCEND, or MXML_DESCEND_FIRST

Return Value

Element node or NULL

Discussion

The search is constrained by the name, attribute name, and value; any NULL names or values are treated as wildcards, so different kinds of searches can be implemented by looking for all elements of a given name or all elements with a specific attribute. The descend argument determines whether the search descends into child nodes; normally you will use MXML_DESCEND_FIRST for the initial search and MXML_NO_DESCEND to find additional direct descendents of the node. The top node argument constrains the search to a particular node's children.

mxmlFindPath

Find a node with the given path.

```
mxml_node_t *mxmlFindPath (
    mxml_node_t *top,
    const char *path
);
```

Parameters

top
: Top node

path
: Path to element

Return Value

Found node or NULL

Discussion

The "path" is a slash-separated list of element names. The name "*" is considered a wildcard for one or more levels of elements. For example, "foo/one/two", "bar/two/one", "*/one", and so forth.

The first child node of the found node is returned if the given node has children and the first child is a value node.

mxmlGetCDATA

Get the value for a CDATA node.

```
const char *mxmlGetCDATA (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

CDATA value or NULL

Discussion

`NULL` is returned if the node is not a CDATA element.

mxmlGetCustom

Get the value for a custom node.

```
const void *mxmlGetCustom (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Custom value or NULL

Discussion

`NULL` is returned if the node (or its first child) is not a custom value node.

mxmlGetElement

Get the name for an element node.

```
const char *mxmlGetElement (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Element name or NULL

Discussion

`NULL` is returned if the node is not an element node.

mxmlGetFirstChild

Get the first child of an element node.

```
mxml_node_t *mxmlGetFirstChild (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

First child or NULL

Discussion

`NULL` is returned if the node is not an element node or if the node has no children.

mxmlGetInteger

Get the integer value from the specified node or its first child.

```
int mxmlGetInteger (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Integer value or 0

Discussion

0 is returned if the node (or its first child) is not an integer value node.

mxmlGetLastChild

Get the last child of an element node.

```
mxml_node_t *mxmlGetLastChild (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Last child or NULL

Discussion

`NULL` is returned if the node is not an element node or if the node has no children.

mxmlGetNextSibling

Return the node type...

```
mxml_node_t *mxmlGetNextSibling (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Get the next node for the current parent.

NULL is returned if this is the last child for the current parent.

mxmlGetOpaque

Get an opaque string value for a node or its first child.

```
const char *mxmlGetOpaque (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Opaque string or NULL

Discussion

NULL is returned if the node (or its first child) is not an opaque value node.

mxmlGetParent

Get the parent node.

```
mxml_node_t *mxmlGetParent (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Parent node or NULL

Discussion

`NULL` is returned for a root node.

mxmlGetPrevSibling

Get the previous node for the current parent.

```
mxml_node_t *mxmlGetPrevSibling (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Previous node or NULL

Discussion

`NULL` is returned if this is the first child for the current parent.

mxmlGetReal

Get the real value for a node or its first child.

```
double mxmlGetReal (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Real value or 0.0

Discussion

0.0 is returned if the node (or its first child) is not a real value node.

mxmlGetRefCount

Get the current reference (use) count for a node.

```
int mxmlGetRefCount (
    mxml_node_t *node
);
```

Parameters

node
: Node

Return Value

Reference count

Discussion

The initial reference count of new nodes is 1. Use the `mxmlRetain` and `mxmlRelease` functions to increment and decrement a node's reference count. .

mxmlGetText

Get the text value for a node or its first child.

```
const char *mxmlGetText (
    mxml_node_t *node,
    int *whitespace
);
```

Parameters

node
: Node to get

whitespace
: 1 if string is preceded by whitespace, 0 otherwise

Return Value

Text string or NULL

Discussion

`NULL` is returned if the node (or its first child) is not a text node. The "whitespace" argument can be NULL.

mxmlGetType

Get the node type.

```
mxml_type_t mxmlGetType (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

Type of node

Discussion

`MXML_IGNORE` is returned if "node" is `NULL`.

mxmlGetUserData

Get the user data pointer for a node.

```
void *mxmlGetUserData (
    mxml_node_t *node
);
```

Parameters

node
: Node to get

Return Value

User data pointer

mxmlIndexDelete

Delete an index.

```
void mxmlIndexDelete (
   mxml_index_t *ind
);
```

Parameters

ind
: Index to delete

mxmlIndexEnum

Return the next node in the index.

```
mxml_node_t *mxmlIndexEnum (
   mxml_index_t *ind
);
```

Parameters

ind
: Index to enumerate

Return Value

Next node or NULL if there is none

Discussion

Nodes are returned in the sorted order of the index.

mxmlIndexFind

Find the next matching node.

```
mxml_node_t *mxmlIndexFind (
    mxml_index_t *ind,
    const char *element,
    const char *value
);
```

Parameters

ind
: Index to search

element
: Element name to find, if any

value
: Attribute value, if any

Return Value

Node or NULL if none found

Discussion

You should call mxmlIndexReset() prior to using this function for the first time with a particular set of "element" and "value" strings. Passing NULL for both "element" and "value" is equivalent to calling mxmlIndexEnum().

mxmlIndexGetCount

Get the number of nodes in an index.

```
int mxmlIndexGetCount (
    mxml_index_t *ind
);
```

Parameters

ind
: Index of nodes

Return Value

Number of nodes in index

mxmlIndexNew

Create a new index.

```
mxml_index_t *mxmlIndexNew (
    mxml_node_t *node,
    const char *element,
    const char *attr
);
```

Parameters

node
: XML node tree

element
: Element to index or NULL for all

attr
: Attribute to index or NULL for none

Return Value

New index

Discussion

The index will contain all nodes that contain the named element and/or attribute. If both "element" and "attr" are NULL, then the index will contain a sorted list of the elements in the node tree. Nodes are sorted by element name and optionally by attribute value if the "attr" argument is not NULL.

mxmlIndexReset

Reset the enumeration/find pointer in the index and return the first node in the index.

```
mxml_node_t *mxmlIndexReset (
    mxml_index_t *ind
);
```

Parameters

ind
: Index to reset

Return Value

First node or NULL if there is none

Discussion

This function should be called prior to using mxmlIndexEnum() or mxmlIndexFind() for the first time.

mxmlLoadFd

Load a file descriptor into an XML node tree.

```
mxml_node_t *mxmlLoadFd (
    mxml_node_t *top,
    int fd,
    mxml_load_cb_t cb
);
```

Parameters

top
: Top node

fd
: File descriptor to read from

cb
: Callback function or MXML_NO_CALLBACK

Return Value

First node or NULL if the file could not be read.

Discussion

The nodes in the specified file are added to the specified top node. If no top node is provided, the XML file MUST be well-formed with a single parent node like <?xml> for the entire file. The callback function returns the value type that should be used for child nodes. If MXML_NO_CALLBACK is specified then all child nodes will be either MXML_ELEMENT or MXML_TEXT nodes.

The constants MXML_INTEGER_CALLBACK, MXML_OPAQUE_CALLBACK,

MXML_REAL_CALLBACK, and MXML_TEXT_CALLBACK are defined for loading child nodes of the specified type.

mxmlLoadFile

Load a file into an XML node tree.

```
mxml_node_t *mxmlLoadFile (
    mxml_node_t *top,
    FILE *fp,
    mxml_load_cb_t cb
);
```

Parameters

top
: Top node

fp
: File to read from

cb
: Callback function or MXML_NO_CALLBACK

Return Value

First node or NULL if the file could not be read.

Discussion

The nodes in the specified file are added to the specified top node. If no top node is provided, the XML file MUST be well-formed with a single parent node like <?xml> for the entire file. The callback function returns the value type that should be used for child nodes. If MXML_NO_CALLBACK is specified then all child nodes will be either MXML_ELEMENT

or MXML_TEXT nodes.

The constants MXML_INTEGER_CALLBACK, MXML_OPAQUE_CALLBACK, MXML_REAL_CALLBACK, and MXML_TEXT_CALLBACK are defined for loading child nodes of the specified type.

mxmlLoadString

Load a string into an XML node tree.

```
mxml_node_t *mxmlLoadString (
    mxml_node_t *top,
    const char *s,
    mxml_load_cb_t cb
);
```

Parameters

top
: Top node

s
: String to load

cb
: Callback function or MXML_NO_CALLBACK

Return Value

First node or NULL if the string has errors.

Discussion

The nodes in the specified string are added to the specified top node. If no top node is provided, the XML string MUST be well-formed with a single parent

node like <?xml> for the entire string. The callback function returns the value type that should be used for child nodes. If MXML_NO_CALLBACK is specified then all child nodes will be either MXML_ELEMENT or MXML_TEXT nodes.

The constants MXML_INTEGER_CALLBACK, MXML_OPAQUE_CALLBACK, MXML_REAL_CALLBACK, and MXML_TEXT_CALLBACK are defined for loading child nodes of the specified type.

mxmlNewCDATA

Create a new CDATA node.

```
mxml_node_t *mxmlNewCDATA (
   mxml_node_t *parent,
   const char *data
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

data
: Data string

Return Value

New node

Discussion

The new CDATA node is added to the end of the specified parent's child list. The constant

MXML_NO_PARENT can be used to specify that the new CDATA node has no parent. The data string must be nul-terminated and is copied into the new node. CDATA nodes use the MXML_ELEMENT type.

mxmlNewCustom

Create a new custom data node.

```
mxml_node_t *mxmlNewCustom (
    mxml_node_t *parent,
    void *data,
    mxml_custom_destroy_cb_t destroy
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

data
: Pointer to data

destroy
: Function to destroy data

Return Value

New node

Discussion

The new custom node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the new element node has no parent. NULL can be passed when the data in the node is not dynamically allocated or is separately managed.

mxmlNewElement

Create a new element node.

```
mxml_node_t *mxmlNewElement (
   mxml_node_t *parent,
   const char *name
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

name
: Name of element

Return Value

New node

Discussion

The new element node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the new element node has no parent.

mxmlNewInteger

Create a new integer node.

```
mxml_node_t *mxmlNewInteger (
   mxml_node_t *parent,
   int integer
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

integer
: Integer value

Return Value

New node

Discussion

The new integer node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the new integer node has no parent.

mxmlNewOpaque

Create a new opaque string.

```
mxml_node_t *mxmlNewOpaque (
    mxml_node_t *parent,
    const char *opaque
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

opaque
: Opaque string

Return Value

New node

Discussion

The new opaque node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the new opaque node has no parent. The opaque string must be nul-terminated and is copied into the new node.

mxmlNewReal

Create a new real number node.

```
mxml_node_t *mxmlNewReal (
   mxml_node_t *parent,
   double real
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

real
: Real number value

Return Value

New node

Discussion

The new real number node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the new real number node has no parent.

mxmlNewText

Create a new text fragment node.

```
mxml_node_t *mxmlNewText (
    mxml_node_t *parent,
    int whitespace,
    const char *string
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

whitespace
: 1 = leading whitespace, 0 = no whitespace

string
: String

Return Value

New node

Discussion

The new text node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the new text node has no parent. The whitespace

parameter is used to specify whether leading whitespace is present before the node. The text string must be nul-terminated and is copied into the new node.

mxmlNewTextf

Create a new formatted text fragment node.

```
mxml_node_t *mxmlNewTextf (
    mxml_node_t *parent,
    int whitespace,
    const char *format,
    ...
);
```

Parameters

parent
: Parent node or MXML_NO_PARENT

whitespace
: 1 = leading whitespace, 0 = no whitespace

format
: Printf-style frmat string

...
: Additional args as needed

Return Value

New node

Discussion

The new text node is added to the end of the specified parent's child list. The constant MXML_NO_PARENT can be used to specify that the

new text node has no parent. The whitespace parameter is used to specify whether leading whitespace is present before the node. The format string must be nul-terminated and is formatted into the new node.

mxmlNewXML

Create a new XML document tree.

```
mxml_node_t *mxmlNewXML (
    const char *version
);
```

Parameters

version
: Version number to use

Return Value

New ?xml node

Discussion

The "version" argument specifies the version number to put in the ?xml element node. If NULL, version 1.0 is assumed.

mxmlRelease

Release a node.

```
int mxmlRelease (
    mxml_node_t *node
```

```
);
```

Parameters

node
: Node

Return Value

New reference count

Discussion

When the reference count reaches zero, the node (and any children) is deleted via mxmlDelete().

mxmlRemove

Remove a node from its parent.

```
void mxmlRemove (
    mxml_node_t *node
);
```

Parameters

node
: Node to remove

Discussion

Does not free memory used by the node - use mxmlDelete() for that. This function does nothing if the node has no parent.

mxmlRetain

Retain a node.

```
int mxmlRetain (
    mxml_node_t *node
);
```

Parameters

node
: Node

Return Value

New reference count

mxmlSAXLoadFd

Load a file descriptor into an XML node tree using a SAX callback.

```
mxml_node_t *mxmlSAXLoadFd (
    mxml_node_t *top,
    int fd,
    mxml_load_cb_t cb,
    mxml_sax_cb_t sax_cb,
    void *sax_data
);
```

Parameters

top
: Top node

fd

File descriptor to read from

cb
: Callback function or MXML_NO_CALLBACK

sax_cb
: SAX callback or MXML_NO_CALLBACK

sax_data
: SAX user data

Return Value

First node or NULL if the file could not be read.

Discussion

The nodes in the specified file are added to the specified top node. If no top node is provided, the XML file MUST be well-formed with a single parent node like <?xml> for the entire file. The callback function returns the value type that should be used for child nodes. If MXML_NO_CALLBACK is specified then all child nodes will be either MXML_ELEMENT or MXML_TEXT nodes.

The constants MXML_INTEGER_CALLBACK, MXML_OPAQUE_CALLBACK, MXML_REAL_CALLBACK, and MXML_TEXT_CALLBACK are defined for loading child nodes of the specified type.

The SAX callback must call mxmlRetain() for any nodes that need to be kept for later use. Otherwise, nodes are deleted when the parent node is closed or after each data, comment, CDATA, or directive node.

mxmlSAXLoadFile

Load a file into an XML node tree using a SAX callback.

```
mxml_node_t *mxmlSAXLoadFile (
    mxml_node_t *top,
    FILE *fp,
    mxml_load_cb_t cb,
    mxml_sax_cb_t sax_cb,
    void *sax_data
);
```

Parameters

top
: Top node

fp
: File to read from

cb
: Callback function or MXML_NO_CALLBACK

sax_cb
: SAX callback or MXML_NO_CALLBACK

sax_data
: SAX user data

Return Value

First node or NULL if the file could not be read.

Discussion

The nodes in the specified file are added to the specified top node. If no top node is provided, the XML file MUST be well-formed with a single parent node like <?xml> for the entire file. The callback

function returns the value type that should be used for child nodes. If MXML_NO_CALLBACK is specified then all child nodes will be either MXML_ELEMENT or MXML_TEXT nodes.

The constants MXML_INTEGER_CALLBACK, MXML_OPAQUE_CALLBACK, MXML_REAL_CALLBACK, and MXML_TEXT_CALLBACK are defined for loading child nodes of the specified type.

The SAX callback must call mxmlRetain() for any nodes that need to be kept for later use. Otherwise, nodes are deleted when the parent node is closed or after each data, comment, CDATA, or directive node.

mxmlSAXLoadString

Load a string into an XML node tree using a SAX callback.

```
mxml_node_t *mxmlSAXLoadString (
    mxml_node_t *top,
    const char *s,
    mxml_load_cb_t cb,
    mxml_sax_cb_t sax_cb,
    void *sax_data
);
```

Parameters

top
: Top node

s
: String to load

cb

Callback function or MXML_NO_CALLBACK

sax_cb

SAX callback or MXML_NO_CALLBACK

sax_data

SAX user data

Return Value

First node or NULL if the string has errors.

Discussion

The nodes in the specified string are added to the specified top node. If no top node is provided, the XML string MUST be well-formed with a single parent node like <?xml> for the entire string. The callback function returns the value type that should be used for child nodes. If MXML_NO_CALLBACK is specified then all child nodes will be either MXML_ELEMENT or MXML_TEXT nodes.

The constants MXML_INTEGER_CALLBACK, MXML_OPAQUE_CALLBACK, MXML_REAL_CALLBACK, and MXML_TEXT_CALLBACK are defined for loading child nodes of the specified type.

The SAX callback must call mxmlRetain() for any nodes that need to be kept for later use. Otherwise, nodes are deleted when the parent node is closed or after each data, comment, CDATA, or directive node.

mxmlSaveAllocString

Save an XML tree to an allocated string.

```
char *mxmlSaveAllocString (
    mxml_node_t *node,
    mxml_save_cb_t cb
);
```

Parameters

node
: Node to write

cb
: Whitespace callback or MXML_NO_CALLBACK

Return Value

Allocated string or NULL

Discussion

This function returns a pointer to a string containing the textual representation of the XML node tree. The string should be freed using the free() function when you are done with it. NULL is returned if the node would produce an empty string or if the string cannot be allocated.

The callback argument specifies a function that returns a whitespace string or NULL before and after each element. If MXML_NO_CALLBACK is specified, whitespace will only be added before MXML_TEXT nodes with leading whitespace and before attribute names inside opening element tags.

mxmlSaveFd

Save an XML tree to a file descriptor.

```
int mxmlSaveFd (
    mxml_node_t *node,
    int fd,
    mxml_save_cb_t cb
);
```

Parameters

node
: Node to write

fd
: File descriptor to write to

cb
: Whitespace callback or MXML_NO_CALLBACK

Return Value

0 on success, -1 on error.

Discussion

The callback argument specifies a function that returns a whitespace string or NULL before and after each element. If MXML_NO_CALLBACK is specified, whitespace will only be added before MXML_TEXT nodes with leading whitespace and before attribute names inside opening element tags.

mxmlSaveFile

Save an XML tree to a file.

```
int mxmlSaveFile (
    mxml_node_t *node,
    FILE *fp,
    mxml_save_cb_t cb
);
```

Parameters

node
: Node to write

fp
: File to write to

cb
: Whitespace callback or MXML_NO_CALLBACK

Return Value

0 on success, -1 on error.

Discussion

The callback argument specifies a function that returns a whitespace string or NULL before and after each element. If MXML_NO_CALLBACK is specified, whitespace will only be added before MXML_TEXT nodes with leading whitespace and before attribute names inside opening element tags.

mxmlSaveString

Save an XML node tree to a string.

```
int mxmlSaveString (
    mxml_node_t *node,
    char *buffer,
    int bufsize,
    mxml_save_cb_t cb
);
```

Parameters

node
: Node to write

buffer
: String buffer

bufsize
: Size of string buffer

cb
: Whitespace callback or MXML_NO_CALLBACK

Return Value

Size of string

Discussion

This function returns the total number of bytes that would be required for the string but only copies (bufsize - 1) characters into the specified buffer.

The callback argument specifies a function that returns a whitespace string or NULL before and after each element. If MXML_NO_CALLBACK is specified,

whitespace will only be added before MXML_TEXT nodes with leading whitespace and before attribute names inside opening element tags.

mxmlSetCDATA

Set the element name of a CDATA node.

```
int mxmlSetCDATA (
    mxml_node_t *node,
    const char *data
);
```

Parameters

node
: Node to set

data
: New data string

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not a CDATA element node.

mxmlSetCustom

Set the data and destructor of a custom data node.

```
int mxmlSetCustom (
    mxml_node_t *node,
```

```
    void *data,
    mxml_custom_destroy_cb_t destroy
);
```

Parameters

node
: Node to set

data
: New data pointer

destroy
: New destructor function

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not a custom node.

mxmlSetCustomHandlers

Set the handling functions for custom data.

```
void mxmlSetCustomHandlers (
    mxml_custom_load_cb_t load,
    mxml_custom_save_cb_t save
);
```

Parameters

load
: Load function

save

Save function

Discussion

The load function accepts a node pointer and a data string and must return 0 on success and non-zero on error.

The save function accepts a node pointer and must return a malloc'd string on success and NULL on error.

mxmlSetElement

Set the name of an element node.

```
int mxmlSetElement (
    mxml_node_t *node,
    const char *name
);
```

Parameters

node
: Node to set

name
: New name string

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it is not an element node.

mxmlSetErrorCallback

Set the error message callback.

```
void mxmlSetErrorCallback (
    mxml_error_cb_t cb
);
```

Parameters

cb
: Error callback function

mxmlSetInteger

Set the value of an integer node.

```
int mxmlSetInteger (
    mxml_node_t *node,
    int integer
);
```

Parameters

node
: Node to set

integer
: Integer value

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not an integer node.

mxmlSetOpaque

Set the value of an opaque node.

```
int mxmlSetOpaque (
    mxml_node_t *node,
    const char *opaque
);
```

Parameters

node
: Node to set

opaque
: Opaque string

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not an opaque node.

mxmlSetReal

Set the value of a real number node.

```
int mxmlSetReal (
    mxml_node_t *node,
    double real
);
```

Parameters

node
: Node to set

real
: Real number value

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not a real number node.

mxmlSetText

Set the value of a text node.

```
int mxmlSetText (
    mxml_node_t *node,
    int whitespace,
    const char *string
);
```

Parameters

node
: Node to set

whitespace

1 = leading whitespace, 0 = no whitespace
string
String

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not a text node.

mxmlSetTextf

Set the value of a text node to a formatted string.

```
int mxmlSetTextf (
    mxml_node_t *node,
    int whitespace,
    const char *format,
    ...
);
```

Parameters

node
Node to set
whitespace
1 = leading whitespace, 0 = no whitespace
format
Printf-style format string
...
Additional arguments as needed

Return Value

0 on success, -1 on failure

Discussion

The node is not changed if it (or its first child) is not a text node.

mxmlSetUserData

Set the user data pointer for a node.

```
int mxmlSetUserData (
   mxml_node_t *node,
   void *data
);
```

Parameters

node
: Node to set

data
: User data pointer

Return Value

0 on success, -1 on failure

mxmlSetWrapMargin

Set the wrap margin when saving XML data.

```
void mxmlSetWrapMargin (
   int column
```

```
);
```

Parameters

column
: Column for wrapping, 0 to disable wrapping

Discussion

Wrapping is disabled when "column" is 0.

mxmlWalkNext

Walk to the next logical node in the tree.

```
mxml_node_t *mxmlWalkNext (
    mxml_node_t *node,
    mxml_node_t *top,
    int descend
);
```

Parameters

node
: Current node

top
: Top node

descend
: Descend into tree - MXML_DESCEND, MXML_NO_DESCEND, or MXML_DESCEND_FIRST

Return Value

Next node or NULL

Discussion

The descend argument controls whether the first child is considered to be the next node. The top node argument constrains the walk to the node's children.

mxmlWalkPrev

Walk to the previous logical node in the tree.

```
mxml_node_t *mxmlWalkPrev (
   mxml_node_t *node,
   mxml_node_t *top,
   int descend
);
```

Parameters

node
: Current node

top
: Top node

descend
: Descend into tree - MXML_DESCEND, MXML_NO_DESCEND, or MXML_DESCEND_FIRST

Return Value

Previous node or NULL

Discussion

The descend argument controls whether the previous node's last child is considered to be the previous node. The top node argument constrains the walk to

the node's children.

Data Types

mxml_custom_destroy_cb_t

Custom data destructor

typedef void (*mxml_custom_destroy_cb_t)(void *);

mxml_custom_load_cb_t

Custom data load callback function

typedef int (*mxml_custom_load_cb_t)(mxml_node_t *, const char *);

mxml_custom_save_cb_t

Custom data save callback function

typedef char *(*mxml_custom_save_cb_t)(mxml_node_t *);

mxml_entity_cb_t

Entity callback function

typedef int (*mxml_entity_cb_t)(const char *);

mxml_error_cb_t

Error callback function

typedef void (*mxml_error_cb_t)(const char *);

mxml_index_t

An XML node index.

typedef struct mxml_index_s mxml_index_t;

mxml_load_cb_t

Load callback function

typedef mxml_type_t
(*mxml_load_cb_t)(mxml_node_t *);

mxml_node_t

An XML node.

typedef struct mxml_node_s mxml_node_t;

mxml_save_cb_t

Save callback function

typedef const char *(*mxml_save_cb_t)(mxml_node_t
*, int);

mxml_sax_cb_t

SAX callback function

typedef void (*mxml_sax_cb_t)(mxml_node_t *,
mxml_sax_event_t, void *);

mxml_sax_event_t

SAX event type.

typedef enum mxml_sax_event_e mxml_sax_event_t;

mxml_type_t

The XML node type.

typedef enum mxml_type_e mxml_type_t;

Constants

mxml_sax_event_e

SAX event type.

Constants

MXML_SAX_CDATA
: CDATA node

MXML_SAX_COMMENT
: Comment node

MXML_SAX_DATA
: Data node

MXML_SAX_DIRECTIVE
: Processing directive node

MXML_SAX_ELEMENT_CLOSE
: Element closed

MXML_SAX_ELEMENT_OPEN
: Element opened

mxml_type_e

The XML node type.

Constants

MXML_CUSTOM Mini-XML 2.1
: Custom data

MXML_ELEMENT
: XML element with attributes

MXML_IGNORE Mini-XML 2.3
: Ignore/throw away node

MXML_INTEGER
: Integer value

MXML_OPAQUE
: Opaque string

MXML_REAL
: Real value

MXML_TEXT
: Text fragment

XML Schema

This appendix provides the XML schema that is used for the XML files produced by `mxmldoc`. This schema is available on-line at:

`http://www.msweet.org/schema/mxmldoc.xsd`

mxmldoc.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Mini-XML 2.9 documentation schema for mxmldoc output.
      Copyright 2003-2014 by Michael Sweet.
    </xsd:documentation>
  </xsd:annotation>

  <!-- basic element definitions -->
  <xsd:element name="argument" type="argumentType"/>
  <xsd:element name="class" type="classType"/>
  <xsd:element name="constant" type="constantType"/>
  <xsd:element name="description" type="xsd:string"/>
```

```
<xsd:element name="enumeration" type="enumerationType"/>
<xsd:element name="function" type="functionType"/>
<xsd:element name="mxmldoc" type="mxmldocType"/>
<xsd:element name="namespace" type="namespaceType"/>
<xsd:element name="returnvalue" type="returnvalueType"/>
<xsd:element name="seealso" type="identifierList"/>
<xsd:element name="struct" type="structType"/>
<xsd:element name="typedef" type="typedefType"/>
<xsd:element name="type" type="xsd:string"/>
<xsd:element name="union" type="unionType"/>
<xsd:element name="variable" type="variableType"/>

<!-- descriptions of complex elements -->
<xsd:complexType name="argumentType">
  <xsd:sequence>
    <xsd:element ref="type" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="default" type="xsd:string" use="optional"/>
  <xsd:attribute name="name" type="identifier" use="required"/>
  <xsd:attribute name="direction" type="direction" use="optional"
   default="I"/>
</xsd:complexType>

<xsd:complexType name="classType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="class"/>
      <xsd:element ref="enumeration"/>
      <xsd:element ref="function"/>
      <xsd:element ref="struct"/>
      <xsd:element ref="typedef"/>
      <xsd:element ref="union"/>
      <xsd:element ref="variable"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="identifier" use="required"/>
  <xsd:attribute name="parent" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:complexType name="constantType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="identifier" use="required"/>
</xsd:complexType>

<xsd:complexType name="enumerationType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="constant" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="identifier" use="required"/>
</xsd:complexType>

<xsd:complexType name="functionType">
  <xsd:sequence>
```

```
    <xsd:element ref="returnvalue" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="argument" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element ref="seealso" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="identifier" use="required"/>
  <xsd:attribute name="scope" type="scope" use="optional"/>
</xsd:complexType>

<xsd:complexType name="mxmldocType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="class"/>
    <xsd:element ref="enumeration"/>
    <xsd:element ref="function"/>
    <xsd:element ref="namespace"/>
    <xsd:element ref="struct"/>
    <xsd:element ref="typedef"/>
    <xsd:element ref="union"/>
    <xsd:element ref="variable"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="namespaceType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="class"/>
      <xsd:element ref="enumeration"/>
      <xsd:element ref="function"/>
      <xsd:element ref="struct"/>
      <xsd:element ref="typedef"/>
      <xsd:element ref="union"/>
      <xsd:element ref="variable"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="identifier" use="required"/>
</xsd:complexType>

<xsd:complexType name="returnvalueType">
  <xsd:sequence>
    <xsd:element ref="type" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="structType">
  <xsd:sequence>
    <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="variable"/>
      <xsd:element ref="function"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="identifier" use="required"/>
</xsd:complexType>

<xsd:complexType name="typedefType">
  <xsd:sequence>
```

```
      <xsd:element ref="type" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="identifier" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="unionType">
    <xsd:sequence>
      <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="variable" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="identifier" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="variableType">
    <xsd:sequence>
      <xsd:element ref="type" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="description" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="identifier" use="required"/>
  </xsd:complexType>

  <!-- data types -->
  <xsd:simpleType name="direction">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="I"/>
      <xsd:enumeration value="O"/>
      <xsd:enumeration value="IO"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="identifier">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[a-zA-Z_(.]([a-zA-Z_(.,)* 0-9])*"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="identifierList">
    <xsd:list itemType="identifier"/>
  </xsd:simpleType>

  <xsd:simpleType name="scope">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value=""/>
      <xsd:enumeration value="private"/>
      <xsd:enumeration value="protected"/>
      <xsd:enumeration value="public"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```