**Mini-XML 入门**

这一章描述了如何写一个程序使用Mini-XML来访问XML文件中的数据.Mini-XML提供了以下功能:

* 在内存中创建和管理XML文档的函数.
* 读UTF-8和UTF-16 编码的XML文件和字符串.
* 写UTF-8 编码的XML文件和字符串.
* 支持任意的元素名称，属性以及属性值，没有任何其他限制，仅受限于有效内存.
* 支持整形、浮点、自定义("CDATA")和文本数据类型在"叶"节点.
* 提供"查找"、"索引"、以及"步进"函数可以很简单的访问XML文档中的数据.

Mini-XML 不进行基于"XML方案(SCHEMA)"文件或者其他内容源定义信息的校验和其他类型的处理 ,也不支持其他组织所要求的XML规范.

**基础知识**

Mini-XML 提供的一个你需要包含的头文件:

#include <mxml.h>

把Mini-XML库连接到你的应用程序使用-lmxml 选项:

gcc -o myprogram myprogram.c -lmxml ENTER

如果你已经安装pkg-config(1) 软件, 你可以使用它来为你的安装确定适当的编译和连接选项:

pkg-config --cflags mxml ENTER

pkg-config --libs mxml ENTER

**节点**

每一块XML文件中的信息片断(元素、文本、数字)是一个存储在内存中的"节点(nodes)" .节点使用[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) 结构进行定义. 它的[type](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_type_t) 成员变量定义了节点类型（element, integer, opaque, real, or text) 决定了需要从联合(union)类型的成员变量[value](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_value_t) 中获取的值.

|  |  |  |
| --- | --- | --- |
| *表 2-1: Mini-XML 节点值的成员变量***值** | **类型** | **节点成员** |
| 用户定义 | void \* | node->value.custom.data |
| XML元素 | char \* | node->value.element.name |
| 整数 | int | node->value.integer |
| 不透明字符串 | char \* | node->value.opaque |
| 浮点数 | double | node->value.real |
| 文本 | char \* | node->value.text.string |

译者：节点类型定义枚举参见：[mxml\_type\_e](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_type_e)。 Mini-XML中的节点类型定义和其他有些解析器有些不同，其中整数、浮点、和文本节点是指在一个XML元素中一系列的使用空格作为分割的值，每个元素 可以拥有多个以上节点，并可以选择使用空格分开，如：<abc>aa bb cc</abc>，Mini-MXML在使用参数：MXML\_TEXT\_CALLBACK进行载入时，将在abc元素下面生成3个text类 型的子节点。在创建时也可以使用同样的方式创建节点。整数和浮点也是同样方式，但如果转换失败则MiniXML报错。而不透明字符串类型(OPAQUE) 则不进行字符串分割，在载入时需要使用MXML\_OPAQUE\_CALLBACK参数，将所有字符串形成一个子节点。详情见：[使用加载回调函数](file:///D:/我的文档/Desktop/minixml-cn.html#ADVANCED)。 Z.F

每一个节点总是有一个成员变量:user\_data 可以允许你为每一个节点关联你需要的应用数据.

新的节点可以使用以下函数进行创建 [mxmlNewElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewElement), [mxmlNewInteger](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewInteger), [mxmlNewOpaque](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewOpaque), [mxmlNewReal](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewReal), [mxmlNewText](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewText) [mxmlNewTextf](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewTextf) [mxmlNewXML](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewXML) . 只有 elements 可以拥有子节点，顶级节点必须是一个 element , 通常是<?xml version="1.0"?> 使用mxmlNewXML()函数创建的节点.

每个节点都有一些关联节点的指针，上(parent), 下(child), 左(prev), and 右(next) 相对应于当前节点. 如果你有一个XML文件如下所示:

<?xml version="1.0"?>

<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>

那么在内存中的文件节点树看上去如下所示:

?xml

|

data

|

node - node - node - group - node - node

| | | | | |

val1 val2 val3 | val7 val8

|

node - node - node

| | |

val4 val5 val6

这里"-"指向下一个节点，"|"指向第一个子节点。

当你使用完毕这些XML数据后，使用函数[mxmlDelete](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlDelete) 来释放指定节点或者整个XML树节点和它下面所有子节点的内存:

mxmlDelete(tree);

**创建 XML 文档**

你可以在内存中创建和更新XML文档，使用mxmlNew 一系列函数. 下面的代码将创建上一章描述的XML文档:

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");

我们首先使用[mxmlNewXML](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewXML)函数来创建所有XML文件都需要的标头 <?xml version="1.0"?> :

xml = mxmlNewXML("1.0");

然后我们使用[mxmlNewElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewElement)函数来创建本文件使用的<data>节点.第一个参数指定了父节点(xml) ，第二个参数是元素名 (data):

data = mxmlNewElement(xml, "data");

每个在本文件中<node>...</node>之间的部分使用函数[mxmlNewElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewElement) 和 [mxmlNewText](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewText)来创建. mxmlNewText 的第一个参数指定了父节点 (node).第二个参数指定了是否在文本之前添加空白字符，在本例中使用0或者false.最后一个参数指定了需要添加的实际文本:

node = mxmlNewElement(data, "node");

mxmlNewText(node, 0, "val1");

在内存中的XML结果可以被保存或者进行其他处理，就像一个从磁盘或者字符串中读取的文档一样.

**加载XML**

你可以加载一个XML文件使用函数[mxmlLoadFile](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadFile) :

FILE \*fp;

mxml\_node\_t \*tree;

fp = fopen("filename.xml", "r");

tree = mxmlLoadFile(NULL, fp,

MXML\_TEXT\_CALLBACK);

fclose(fp);

第一个参数如果有的话则指定了一个存在的XML父节点.一般你将这个参数等于NULL,除非你想要连接多个XML源. 如果此参数等于NULL，那么指定的XML文件必须是一个完整的XML文档，文档头部要包含?xml元素.

第二个参数指定了一个标准的文件流，使用 fopen() 或者 popen()进行打开. 你也可以使用stdin，如果你想要实现一个XML过滤器程序.

第三个参数指定了一个回调函数用于一个新的XML元素节点直接返回的子节点的值类型: MXML\_CUSTOM, MXML\_IGNORE, MXML\_INTEGER, MXML\_OPAQUE, MXML\_REAL, or MXML\_TEXT. 加载回调函数的细节在[第三章](file:///D:/我的文档/Desktop/minixml-cn.html#LOAD_CALLBACKS)做了详细描述. 示例代码使用 MXML\_TEXT\_CALLBACK 常量指定文档中所有的数据节点都包含使用以空格字符分割的文本的值. 其他标准的回调还有MXML\_IGNORE\_CALLBACK, MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, 和MXML\_REAL\_CALLBACK.

函数[mxmlLoadString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadString) 可以从一个字符串中载入XML节点树:

char buffer[8192];

mxml\_node\_t \*tree;

...

tree = mxmlLoadString(NULL, buffer,

MXML\_TEXT\_CALLBACK);

第一个和第三个参数和mxmlLoadFile()用法一样. 第二个参数指定了指定了字符串或者字符缓冲区用于加载XML，当父节点参数为NULL时内容必须为完整的XML文档，包括XML头?xml元素.

**保存 XML**

你可以保存XML文件使用[mxmlSaveFile](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveFile) 函数:

FILE \*fp;

mxml\_node\_t \*tree;

fp = fopen("filename.xml", "w");

mxmlSaveFile(tree, fp, MXML\_NO\_CALLBACK);

fclose(fp);

第一个参数为想要保存的XML节点树,一般应该是一个指向你的XML文档顶级节点?xml的节点指针.

第二个单数是一个标准文件流，使用fopen() 或者 popen()来打开. 你也可以使用stdout 如果你想要实现一个XML过滤器程序.

第三个参数是一个空白回调函数用来控制保存文件时插入的"空白"字符."空白回调"的详细信息参见[第三章](file:///D:/我的文档/Desktop/minixml-cn.html#SAVE_CALLBACKS). 以上的示例代码使用了MXML\_NO\_CALLBACK常量来指定不需要特别的空白处理.

函数[mxmlSaveAllocString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveAllocString), 和[mxmlSaveString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveString) 保存XML节点树到一个字符串中:

char buffer[8192];

char \*ptr;

mxml\_node\_t \*tree;

...

mxmlSaveString(tree, buffer, sizeof(buffer),

MXML\_NO\_CALLBACK);

...

ptr = mxmlSaveAllocString(tree, MXML\_NO\_CALLBACK);

第一个和最后一个参数的用法和函数mxmlSaveFile()一样. 函数mxmlSaveString 给出了一个指针和长度的参数来保存XML文档到一个固定大小的缓冲区中。, mxmlSaveAllocString() 返回了使用malloc分配的一个字符串缓冲区malloc().

**自动折行控制**

当我们保存XML文档时, Mini-XML一般在第75列进行折行，因为这样在终端下最易读. 函数[mxmlSetWrapMargin](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetWrapMargin) 可以覆盖缺省的折行界限:

/\* 设置自动折行到 132 列\*/

mxmlSetWrapMargin(132);

/\* 取消自动折行\*/

mxmlSetWrapMargin(0);

**搜索和遍历节点**

函数[mxmlWalkPrev](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlWalkPrev) and [mxmlWalkNext](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlWalkNext)可以被用来遍历XML节点树:

mxml\_node\_t \*node;

node = mxmlWalkPrev(current, tree,

MXML\_DESCEND);

node = mxmlWalkNext(current, tree,

MXML\_DESCEND);

另外,你可以搜索一个命名的XML元素/节点,使用函数[mxmlFindElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlFindElement):

mxml\_node\_t \*node;

node = mxmlFindElement(tree, tree, "name",

"attr", "value",

MXML\_DESCEND);

参数name, attr, 和value 可以被设置为NULL作为全部匹配, e.g.:

/\* 搜索第一个 "a" 元素\*/

node = mxmlFindElement(tree, tree, "a",

NULL, NULL,

MXML\_DESCEND);

/\* 搜索第一个"a" 元素并包含"href"属性\*/

node = mxmlFindElement(tree, tree, "a",

"href", NULL,

MXML\_DESCEND);

/\* 搜索第一个"a" 元素并且包含"href"属性等于给出的URL \*/

node = mxmlFindElement(tree, tree, "a",

"href",

"http://www.easysw.com/",

MXML\_DESCEND);

/\* 搜索第一个包含"src"属性的XML元素\*/

node = mxmlFindElement(tree, tree, NULL,

"src", NULL,

MXML\_DESCEND);

/\* 搜索第一个包含"src"= "foo.jpg"属性的XML元素 \*/

node = mxmlFindElement(tree, tree, NULL,

"src", "foo.jpg",

MXML\_DESCEND);

你也可以使用同样的功能进行遍历:

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 ...

}

参数MXML\_DESCEND可以是下面三个常量之一:

* MXML\_NO\_DESCEND含义是不查看任何的子节点在XML元素层次中，仅查看同级的伙伴节点或者父节点直到到达顶级节点或者给出的树的顶级节点.

"group"节点的上一个节点时它左面的"node"子节点,下一个节点是"group"右面的"node"子节点..

* MXML\_DESCEND\_FIRST含义是向下搜索到一个节点的第一个匹配子节点，但不再继续向下搜索。你一般使用于遍历一个父节点的直接的子节点。 ,如：在上面的例子中的所有在"?xml"父节点下的所有的"node"和"group"子节点。.

这个模式仅适用于搜索(search)功能，遍历功能(walk)对待它和 MXML\_DESCEND 一样，因为每次调用都是首次调用。

* MXML\_DESCEND含义是一直向下直到树的根部. "group"节点的上一个节点将是"val3"节点，下一个节点将是"group"的下面的第一个子节点。

如果你要使用函数mxmlWalkNext()从根结点"?xml" 遍历到整个树的结束, 那么这个顺序将如下所示：

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

如果你从"val8"开始并使用函数mxmlWalkPrev()进行遍历, 这个顺序将是反的，结束于"?xml"节点.

**更多的 Mini-XML 编程技术**

这一章显示了更多的在你的应用程序中使用Mini-XML的方法。

**Load Callbacks**

[第二章](file:///D:/我的文档/Desktop/minixml-cn.html#LOAD_XML) 介绍了函数[mxmlLoadFile()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadFile) 和[mxmlLoadString()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadString) . 这些函数的最后一个参数是一个回调函数，决定了在一个XML文档中每个数据节点的值的类型。

Mini-XML 为简单XML数据文件定义了几个标准的回调函数：

* MXML\_INTEGER\_CALLBACK- 所有的数据节点包含以空格分割的整数。
* MXML\_OPAQUE\_CALLBACK- 所有的数据节点包含"不透明"字符串（CDATA）。
* MXML\_REAL\_CALLBACK- 所有的数据节点包含以空格分割的浮点数。
* MXML\_TEXT\_CALLBACK- 所有的数据节点包含以空格分割的文本字符串。

你可以为更复杂的XML文档提供你自己的回调函数。你的回调函数将接收到一个到当前XML元素节点的指针并且必须为这个XML元素节点返回一个直接子节点的值类型: MXML\_INTEGER, MXML\_OPAQUE, MXML\_REAL, 或MXML\_TEXT.这个函数在这个XML元素和它的全部属性被读取*以后*被调用，所以你可以查看这个XML元素的名称、属性以及属性的值来决定适当的返回值类型。

下面的回调函数查看一个名称为"type"的属性或者XML元素的名字来决定它的子节点的值类型：

mxml\_type\_t

type\_cb(mxml\_node\_t \*node)

{

const char \*type;

/\*

\* 你可以查看属性和/或使用XML元素名，所在层次，等等

\*/

type = mxmlElementGetAttr(node, "type");

if (type == NULL)

type = node->value.element.name;

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);

}

要使用这个回调函数，只需要在你调用任何加载函数时简单的使用它的名字：

FILE \*fp;

mxml\_node\_t \*tree;

fp = fopen("filename.xml", "r");

tree = mxmlLoadFile(NULL, fp, **type\_cb**);

fclose(fp);

**保存回调**

[第二章](file:///D:/我的文档/Desktop/minixml-cn.html#LOAD_XML) 也介绍了[mxmlSaveFile()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveFile), [mxmlSaveString()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveString), 和[mxmlSaveAllocString()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveAllocString) 函数。这些函数的最后一个参数是一个回调函数被用来自动在一个XML文档中添加空白字符。

你的回调函数将在每个XML元素被调用四次，传入参数为一个到这个节点的指针和一个"where"的值：MXML\_WS\_BEFORE\_OPEN, MXML\_WS\_AFTER\_OPEN, MXML\_WS\_BEFORE\_CLOSE, 或者MXML\_WS\_AFTER\_CLOSE。如果不需要插入空白字符这个回调函数将返回 NULL，否则返回字符串（空白、跳格、回车和换行）将被插入。

下面的空白回调可以被用来为XHTML输出添加空白字符，来使它在一般的文本编辑器中更加易读：

const char \*

whitespace\_cb(mxml\_node\_t \*node,

int where)

{

const char \*name;

/\*

\* 我们可以在任何XML元素之前或之后有条件的添加换行。这些是一些常见的HTML元素。

\*/

name = node->value.element.name;

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"))

{

/\*

\* 在打开之前和关闭之后时换行。

\*/

if (where == MXML\_WS\_BEFORE\_OPEN ||

where == MXML\_WS\_AFTER\_CLOSE)

return ("\n");

}

else if (!strcmp(name, "dl") ||

!strcmp(name, "ol") ||

!strcmp(name, "ul"))

{

/\*

\* 在列表元素前后都添加换行。

\*/

return ("\n");

}

else if (!strcmp(name, "dd") ||

!strcmp(name, "dt") ||

!strcmp(name, "li"))

{

/\*

\* 添加一个"跳格"在<li>, \* <dd>,

\* 和 <dt>之前, 以及一个换行在他们后面...

\*/

if (where == MXML\_WS\_BEFORE\_OPEN)

return ("\t");

else if (where == MXML\_WS\_AFTER\_CLOSE)

return ("\n");

}

/\*

\* 如果不需要添加空白字符则返回NULL。

\*/

return (NULL);

}

要使用这些回调函数，只需要在你调用任何保存函数时简单使用它的名字：

FILE \*fp;

mxml\_node\_t \*tree;

fp = fopen("filename.xml", "w");

mxmlSaveFile(tree, fp, **whitespace\_cb**);

fclose(fp);

**用户定义数据类型**

Mini-XML 支持通过全局的载入和保存回调函数使用自定义数据类型。 每次只能有一组回调函数被同时激活，然而你的回调函数可以为支持多种所需要的自定义数据类型来保存更多的信息。 节点类型 MXML\_CUSTOM 表示一个自定义数据内容的节点。

加载回调接收一个到当前数据节点的指针和一个不透明字符串数据从XML源中并且将字符集转换为UTF-8编码。例如：如果我们想要支持定制的日期/时间类型并且编码为"yyyy-mm-ddThh:mm:ssZ" (ISO 格式), 那么加载回调函数如下所示：

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;

/\*

\* 分配数据结构...

\*/

dt = calloc(1, sizeof(iso\_date\_time\_t));

/\*

\* 尝试从数据字符串中读取6个无符号整数..

\*/

if (sscanf(data, "%u-%u-%uT%u:%u:%uZ",

&(dt->year), &(dt->month),

&(dt->day), &(dt->hour),

&(dt->minute),

&(dt->second)) != 6)

{

/\*

\* 如果不能读取到数字，释放分配的结构并返回一个错误...

\*/

free(dt);

return (-1);

}

/\*

\* 数据范围检查...

\*/

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)

{

/\*

\* 如果日期信息超出范围...

\*/

free(dt);

return (-1);

}

/\*

\* 转换ISO时间到以秒为单位的UNIX时间...

\*/

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);

/\*

\* 设置自定义节点数据指针和销毁函数指针...

\*/

node->value.custom.data = dt;

node->value.custom.destroy = free;

/\*

\* 返回没有错误...

\*/

return (0);

}

这个函数成功时返回0，当不能正确解码自定义数据或者数据内容错误时返回-1。自定义数据节点包含一个void 指针用来保存这个节点已经分配的自定义数据，还有一个指向销毁函数的指针用于当节点被删除时释放自定义数据。

保存回调接收一个节点指针并且返回一个包含自定义数据值的已经分配的字符串。下面的保存回调函数可以被用来保存我们的ISO日期/时间类型：

char \*

save\_custom(mxml\_node\_t \*node)

{

char data[255];

iso\_date\_time\_t \*dt;

dt = (iso\_date\_time\_t \*)node->custom.data;

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));

}

你可以注册这些回调函数使用[mxmlSetCustomHandlers()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetCustomHandlers) 函数：

mxmlSetCustomHandlers(**load\_custom**,**save\_custom**);

**改变节点的值**

到现在为止所有的例子集中描述了如何创建和加载新的XML数据节点。然而，许多的应用程序在它们的工作中需要操纵或者改变节点，所以Mini-XML提供了一些函数来安全的改变节点的值并且不会发生内存泄漏。

已有的节点可以被改变通过使用函数[mxmlSetElement()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetElement), [mxmlSetInteger()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetInteger), [mxmlSetOpaque()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetOpaque), [mxmlSetReal()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetReal), [mxmlSetText()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetText), 和 [mxmlSetTextf()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetTextf)。例如：使用下面的函数调用可以改变一个文本节点到包含字符串"new"并且具有前导的空白字符：

mxml\_node\_t \*node;

mxmlSetText(node, 1, "new");

**格式化的文本**

[mxmlNewTextf()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewTextf)和[mxmlSetTextf()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetTextf) 函数分别是创建和改变文本节点，使用printf-风格的格式字符串和参数。例如：使用下面的函数调用来创建一个新的文本节点包含一个构造的文件名：

mxml\_node\_t \*node;

node = mxmlNewTextf(node, 1, "%s/%s",

path, filename);

**索引**

Mini-XML 提供了一些函数来管理节点的索引。当前的实现提供了同样的功能就像 [mxmlFindElement()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlFindElement)一样。使用索引优势是可以使搜索和枚举XML元素显著加快。唯一不利的是每个索引都是一个 关于这个XML文档的静态快照，所以索引不适合当相对于它的搜索操作，XML数据更加频繁更新时的情况。上面的创建一个索引近似相当于遍历这个XML文档 树。在索引中的节点被按照XML元素名和它的参数值进行排序。

这些索引被保存在[mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t) 结构中。用[mxmlIndexNew()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexNew)函数可以创建一个新的索引：

mxml\_node\_t \*tree;

mxml\_index\_t \*ind;

ind = mxmlIndexNew(tree, "element",

"attribute");

第一个参数是需要进行索引的XML节点树。通常是一个指向?xml元素的节点。

第二个参数包含了需要进行索引的XML元素；使用NULL值将按照字母顺序索引所有的XML元素节点。

第三个参数包含了需要进行索引的属性；使用NULL将使只有XML元素名字被索引。

当索引被建立后，函数[mxmlIndexEnum()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexEnum), [mxmlIndexFind()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexFind), and [mxmlIndexReset()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexReset)可以被用来访问索引中的节点。 函数[mxmlIndexReset()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexReset)被用来重置在索引中的"当前"节点指针，允许你在同一个索引中进行新的搜索和枚举遍历。典型应用是你将在你调用函数[mxmlIndexEnum()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexEnum) 和[mxmlIndexFind()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexFind)之前调用这个函数。

函数[mxmlIndexEnum()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexEnum) 用来枚举在索引中的每一个节点，可以在一个循环中使用，如下所示：

mxml\_node\_t \*node;

mxmlIndexReset(ind);

while ((node = mxmlIndexEnum(ind)) != NULL)

{

// do something with node

}

函数[mxmlIndexFind()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexFind) 定位下一次在索引中出现的XML元素名和属性值。它可以被用于发现在索引中的所有匹配的XML元素，如下所示：

mxml\_node\_t \*node;

mxmlIndexReset(ind);

while ((node = mxmlIndexFind(ind, "element",

"attr-value"))

!= NULL)

{

// do something with node

}

第二和第三个参数分别表示XML元素名和属性值。使用 NULL指针用来返回索引中所有的XML元素或者属性。如果XML元素名和属性值同时为NULL则相当于调用函数mxmlIndexEnum。

当我们使用完这个索引后，使用函数[mxmlIndexDelete()](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexDelete()) 来删除它:

mxmlIndexDelete(ind);

**SAX (流方式解析) 加载文档**

Mini-XML支持一个关于简单XML API (SAX)的实现，以允许你通过节点流的方式加载和处理XML文档。另外允许你处理任何大小的XML文件，Mini-XML的实现也允许你为下一步的处理而只在内存中保留XML文档的一部分。

The [mxmlSAXLoadFd](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSAXLoad), [mxmlSAXLoadFile](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSAXLoadFile), 和[mxmlSAXLoadString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSAXLoadString) 函数提供了SAX加载的API。每个函数工作起来就象 mxmlLoad 函数一样，但是使用一个回调函数来处理它读到的每一个节点。

回调函数接收到一个节点，一个事件代码和一个你提供的用户数据指针：

void

sax\_cb(mxml\_node\_t \*node,

mxml\_sax\_event\_t event,

void \*data)

{

... do something ...

}

事件(event)将是下面的其中一个：

* MXML\_SAX\_CDATA- CDATA 正在被读取
* MXML\_SAX\_COMMENT- 一个注释正在被读取
* MXML\_SAX\_DATA- 数据(custom, integer, opaque, real, or text) 正在被读取
* MXML\_SAX\_DIRECTIVE- 一个处理指令正在被读取
* MXML\_SAX\_ELEMENT\_OPEN- 一个"打开"元素节点正在被读取，如(<element>)
* MXML\_SAX\_ELEMENT\_CLOSE- 一个"关闭"元素节点正在被读取，如(</element>)

XML元素将被 *释放* 在一个"关闭"元素被处理后。所有其他的节点在他们被处理后都被释放。SAX回调函数可以 *保留* 这个节点，通过调用函数[mxmlRetain](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlRetain) 。例如：下面的SAX回调函数将保留所有的节点，就像一个普通的内存加载一样：

void

sax\_cb(mxml\_node\_t \*node,

mxml\_sax\_event\_t event,

void \*data)

{

if (event != MXML\_SAX\_ELEMENT\_CLOSE)

mxmlRetain(node);

}

更多的典型SAX回调函数将只保留后面需要处理的这个XML文档的一小部分。例如，下面的SAX回调函数将保留一个XHTML文件的标题和头部信息。它总是保留一个XML元素（父节点）就像<html>, <head>, 和 <body>,并处理指令节点就像 <?xml ... ?> 和<!DOCTYPE ... >：

void

sax\_cb(mxml\_node\_t \*node,

mxml\_sax\_event\_t event,

void \*data)

{

if (event == MXML\_SAX\_ELEMENT\_OPEN)

{

/\*

\* 保留标题和头部

\*/

char \*name = node->value.element.name;

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 &&

node->parent->ref\_count > 1)

{

/\*

\* 如果父节点被保留，那么这个数据节点最好是也被保留。

\*/

mxmlRetain(node);

}

}

这个结果框架文档树可以被搜素就像一个使用函数mxmlLoad 加载的一样。例如，一个过滤器程序用来从标准输入(stdin)中读取一个XHTML文档，并显示在文档中的标题和标头，如下所示：

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 = body->child;

heading;

heading = heading->next)

print\_children(heading);

}

**使用mxmldoc 工具**

这一章描述了如何使用mxmldoc(1) 程序自动从c和c++源文件中生成文档。

**基础知识**

最初开发是用来生成Mini-XML 和CUPS API的文档，现在mxmldoc是一个通用工具实现了扫描C和C++源文件以生成HTML和man手册页文档与一个描述这些源文件中的函数、类型和宏的定义的XML文件。不像一些流行的文档生成工具如Doxygen 或 Javadoc, mxmldoc使用"在线"注释并不是注释头文件，允许更加自然的代码文档。

缺省情况下，mxmldoc 生成HTML文档个，例如：下面的命令将扫描所有的在当前目录下的C源代码和头文件并生成一个HTML文档文件叫 *filename.html*:

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

你也可以指定创建一个包含所有源文件信息的XML文件。例如，下面的命令创建一个XML文件 *filename.xml*并添加到这个HTML文件中：

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

--no-output 选项关闭标准的HTML输出：

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

你可以再次运行mxmldoc通过这个XML文件来生成HTML文档：

mxmldoc filename.xml >filename.html ENTER

--man filename 选项来告诉mxmldoc创建一个手册页(man page)代替HTML文档，例如：

mxmldoc --man filename filename.xml \

>filename.man ENTER

mxmldoc --man filename \*.h \*.c \

>filename.man ENTER

mxmldoc --man filename filename.xml \*.h \*.c \

>filename.man ENTER

**注释你的代码**

在上面已经提到，mxmldoc 查看在线注释来描述你源文件中的函数、类型以及常量。Mxmldoc 将为你源文件中"所有"公开的命名生成文档，所有以下划线开始的名称和被使用[@private@](file:///D:/我的文档/Desktop/minixml-cn.html#ATDIRECTIVES) 指令注解的名称被认为是“私有”的而不生成文档。

出现在函数或者类型定义前面的注释被用来生出这个函数或者类型的文档。出现在参数、定义、返回类型或者变量定义被用来生成它们的注释。例如：下列代码片断定义一个包含key/vaue的结构以及一个创建这个结构的新实例的函数：

/\* 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 总是知道并从注释字符串中移除多余的(\*)号，所以下面的这个注释字符串：

/\*

\* 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.

\*/

将被显示为：

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.

注释中也可以包含@name ...@ 这样的特殊指令字符串：

* @deprecated@- 标识一个条目为不支持并阻止它的使用
* @private@- 标识一个条目为为私有的所以它将不被包含在文档中。
* @since ...@- 标识一个条目为为从一个特别重要的发布更新。从@since到@间的文本在文档中将被高亮显示，如： @since CUPS 1.3@。

**标题、分段名和简介**

Mxmldoc 也提供了一些选项来设置生成的文档中的标题、分段名和简介。The --title text 选项指定了文档的标题。标题字符串通常放在引号内：

mxmldoc filename.xml \

--title "My Famous Documentation" \

>filename.html ENTER

--section name 选项指定了文档的分段名。对于HTML文档，这个名字放在一个HTML注释块中，如下所示：

<!-- SECTION: name -->

对于手册页(man pages), 分段名通常是一个数字("3")，或者是一个数字后面跟着厂商名如("3acme")。这个分段名被使用在手册页(man pages)中的.TH指令里：

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

缺省手册页(man pages)的分段名输出是"3"。对于HTML输出没有缺省的分段名。

最后，--intro filename 选项指定了一个文件来嵌入到在标题和分段名之后和生成文档的前面。对于HTML文档，这个文件必须兼容HTML并且不能使用DOCTYPE, html, and body元素。 .对于手册页文档，这个文件必须兼容有效的nroff(1) 文本.

**库参考手册**

**内容**

[函数](file:///D:/我的文档/Desktop/minixml-cn.html#FUNCTIONS)

* + [mxmlAdd](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlAdd)
  + [mxmlDelete](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlDelete)
  + [mxmlElementDeleteAttr](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlElementDeleteAttr)
  + [mxmlElementGetAttr](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlElementGetAttr)
  + [mxmlElementSetAttr](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlElementSetAttr)
  + [mxmlElementSetAttrf](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlElementSetAttrf)
  + [mxmlEntityAddCallback](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlEntityAddCallback)
  + [mxmlEntityGetName](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlEntityGetName)
  + [mxmlEntityGetValue](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlEntityGetValue)
  + [mxmlEntityRemoveCallback](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlEntityRemoveCallback)
  + [mxmlFindElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlFindElement)
  + [mxmlIndexDelete](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexDelete)
  + [mxmlIndexEnum](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexEnum)
  + [mxmlIndexFind](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexFind)
  + [mxmlIndexNew](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexNew)
  + [mxmlIndexReset](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlIndexReset)
  + [mxmlLoadFd](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadFd)
  + [mxmlLoadFile](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadFile)
  + [mxmlLoadString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlLoadString)
  + [mxmlNewCDATA](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewCDATA)
  + [mxmlNewCustom](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewCustom)
  + [mxmlNewElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewElement)
  + [mxmlNewInteger](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewInteger)
  + [mxmlNewOpaque](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewOpaque)
  + [mxmlNewReal](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewReal)
  + [mxmlNewText](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewText)
  + [mxmlNewTextf](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewTextf)
  + [mxmlNewXML](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlNewXML)
  + [mxmlRelease](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlRelease)
  + [mxmlRemove](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlRemove)
  + [mxmlRetain](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlRetain)
  + [mxmlSAXLoadFd](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSAXLoadFd)
  + [mxmlSAXLoadFile](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSAXLoadFile)
  + [mxmlSAXLoadString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSAXLoadString)
  + [mxmlSaveAllocString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveAllocString)
  + [mxmlSaveFd](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveFd)
  + [mxmlSaveFile](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveFile)
  + [mxmlSaveString](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSaveString)
  + [mxmlSetCDATA](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetCDATA)
  + [mxmlSetCustom](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetCustom)
  + [mxmlSetCustomHandlers](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetCustomHandlers)
  + [mxmlSetElement](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetElement)
  + [mxmlSetErrorCallback](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetErrorCallback)
  + [mxmlSetInteger](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetInteger)
  + [mxmlSetOpaque](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetOpaque)
  + [mxmlSetReal](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetReal)
  + [mxmlSetText](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetText)
  + [mxmlSetTextf](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetTextf)
  + [mxmlSetWrapMargin](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlSetWrapMargin)
  + [mxmlWalkNext](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlWalkNext)
  + [mxmlWalkPrev](file:///D:/我的文档/Desktop/minixml-cn.html#mxmlWalkPrev)

[数据类型](file:///D:/我的文档/Desktop/minixml-cn.html#TYPES)

* + [mxml\_attr\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_attr_t)
  + [mxml\_custom\_destroy\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_destroy_cb_t)
  + [mxml\_custom\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_load_cb_t)
  + [mxml\_custom\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_save_cb_t)
  + [mxml\_custom\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_t)
  + [mxml\_element\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_element_t)
  + [mxml\_error\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_error_cb_t)
  + [mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t)
  + [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t)
  + [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t)
  + [mxml\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_save_cb_t)
  + [mxml\_sax\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_cb_t)
  + [mxml\_sax\_event\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_event_t)
  + [mxml\_text\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_text_t)
  + [mxml\_value\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_value_t)

[结构体](file:///D:/我的文档/Desktop/minixml-cn.html#STRUCTURES)

* + [mxml\_attr\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_attr_s)
  + [mxml\_custom\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_s)
  + [mxml\_element\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_element_s)
  + [mxml\_index\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_s)
  + [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s)
  + [mxml\_text\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_text_s)

[联合](file:///D:/我的文档/Desktop/minixml-cn.html#UNIONS)

* + [mxml\_value\_u](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_value_u)

[常量枚举](file:///D:/我的文档/Desktop/minixml-cn.html#ENUMERATIONS)

* + [mxml\_sax\_event\_e](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_event_e)
  + [mxml\_type\_e](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_type_e)

**函数**

**mxmlAdd**

添加一个节点到树中

void mxmlAdd (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 int where,  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*child,  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node  
 );

**参数**

parent

父节点

where

添加到哪里, MXML\_ADD\_BEFORE or MXML\_ADD\_AFTER

child

where的子节点或者使用MXML\_ADD\_TO\_PARENT

node

准备添加的节点

**说明**

添加一个指定的节点到父节点，如果child参数不是NULL，将这个新的节点添加到指定的"child" 的前面或者后面（由where参数决定）。如果child参数是NULL，把新节点添加到子节点列表的最前面(MXML\_ADD\_BEFORE)或者时子 节点列表的最后面(MXML\_ADD\_AFTER)。常量MXML\_ADD\_TO\_PARENT 可以被用来指定一个NULL的child指针。

**mxmlDelete**

删除一个节点和它的所有的子节点。

void mxmlDelete (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node  
 );

**参数**

node

被删除的节点

**说明**

如果这个指定的节点有一个父节点，这个函数首先使用mxmlRemove() 函数从它的父节点中移除自己。

**Mini-XML 2.4** **mxmlElementDeleteAttr**

删除一个参数

void mxmlElementDeleteAttr (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*name  
 );

**参数**

node

XML元素节点

name

属性名称

**mxmlElementGetAttr**

获取一个参数

const char \*mxmlElementGetAttr (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*name  
 );

**参数**

node

XML元素节点

name

属性名称

**返回值**

属性值或者NULL

**说明**

如果node参数不是一个XML元素或者指定的属性名不存在则返回NULL。

**mxmlElementSetAttr**

设置一个属性。

void mxmlElementSetAttr (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*name,  
 const char \*value  
 );

**参数**

node

XML元素节点

name

属性名称

value

属性值

**说明**

如果这个属性名已经存在，这个属性的值将被替换为新的字符串值。这个字符串值将被拷贝到这个XML元素节点，如果这个节点不是一个XML元素，则这个函数不做任何事。

**Mini-XML 2.3** **mxmlElementSetAttrf**

设置一个XML元素属性使用一个格式化的值。

void mxmlElementSetAttrf (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*name,  
 const char \*format,  
 ...  
 );

**参数**

node

XML元素节点

name

属性名

format

"printf"风格的属性值

...

需要的附加参数(printf)风格

**说明**

如果这个属性名已经存在，这个属性的值将被替换为新的格式化字符串值。这个格式化后字符串值将被拷贝到这个XML元素节点，如果这个节点不是一个XML元素，则这个函数不做任何事。

**mxmlEntityAddCallback**

添加一个回调函数来将XML实体转换为Unicode编码字符。

int mxmlEntityAddCallback (void);

**返回值**

0 成功, -1 失败

回调函数用来将类似于"&amp;"之类的XML实体字符串转化为用户设定的Unicode编码。此函数的原型 应该为：int mxmlEntityAddCallback(int (\*cb)(const char \*name)) 参数为自定义中的回调函数。目前仅支持将实体转换为单个字符的转换，而不支持实体字符串。 Z.F

**mxmlEntityGetName**

获取一个字符值对应的XML实体名字。

const char \*mxmlEntityGetName (  
 int val  
 );

**参数**

val

字符值

**返回值**

XML实体名字或者NULL

**说明**

如果val不需要被标识为一个命名的XML实体，返回NULL。

如：val = '&'，将返回"&amp;" Z.F

**mxmlEntityGetValue**

获取一个代表到一个XML命名实体的字符。

int mxmlEntityGetValue (  
 const char \*name  
 );

**参数**

name

XML实体名字

**返回值**

字符值或者-1代表错误

**说明**

XML实体名字总是可以被关联到一个数字常量，如果这个名字未知则返回-1。

**mxmlEntityRemoveCallback**

删除一个XML实体回调。

void mxmlEntityRemoveCallback (void);

函数原型应该是：void mxmlEntityRemoveCallback(int (\*cb)(const char \*name)); Z.F

**mxmlFindElement**

搜索一个命名的XML元素。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlFindElement (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 const char \*name,  
 const char \*attr,  
 const char \*value,  
 int descend  
 );

**参数**

node

当前节点

top

顶级节点

name

XML元素名，或者NULL匹配所有元素

attr

属性名，或者NULL表示不匹配属性

value

属性值，或者NULL表示任何值

descend

在XML树中向下搜索模式： MXML\_DESCEND, MXML\_NO\_DESCEND, 或者 MXML\_DESCEND\_FIRST

**返回值**

XMl元素节点或者NULL

**说明**

搜索可以被XML元素名，属性名和属性值所限定；任何名字或者值等于NULL被处理就相当于通配符，所以使 用不同的搜索方法可以被实现用来查看所有的指定名称的XML元素或者是所有的具有指定属性的XML元素。参数： descend 确定了是否向下搜索子节点；通常你将使用 MXML\_DESCEND\_FIRST作为第一次搜索，然后使用使用 MXML\_NO\_DESCEND来发现更多的这个节点的直接子节点。 top 节点参数约束了搜索在一个指定节点的子节点中。

**mxmlIndexDelete**

删除一个索引。

void mxmlIndexDelete (  
 [mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t) \*ind  
 );

**参数**

ind

被删除的索引

**mxmlIndexEnum**

返回索引中的下一个节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlIndexEnum (  
 [mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t) \*ind  
 );

**参数**

ind

进行枚举的索引

**返回值**

下一个节点或者NULL代表没有更多的节点

**说明**

返回节点顺序将按照索引的排序被返回。

**mxmlIndexFind**

搜索下一个匹配的节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlIndexFind (  
 [mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t) \*ind,  
 const char \*element,  
 const char \*value  
 );

**参数**

ind

进行搜索的索引

element

如不为NULL，代表想要搜索的XML元素名。

value

如不为NULL，代表想要搜索的属性值。

**返回值**

节点或者NULL代表没有发现。

**说明**

你在第一次使用一个特定的包含"element"和"value"字符串的集合来调用这个函数之前应该首先调用mxmlIndexReset()函数。如果"element"和"value"同时等于NULL则相当于调用了mxmlIndexEnum()函数。

**mxmlIndexNew**

创建一个新的索引。

[mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t) \*mxmlIndexNew (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*element,  
 const char \*attr  
 );

**参数**

node

XML 节点树

element

索引的XML元素名或者NULL代表所有元素

attr

索引的XML属性名或者NULL代表不使用。

**返回值**

新的索引

**说明**

被创建的索引将包含具备指定的元素名和/或属性所有的节点。如果"element" 和"attr"同时等于NULL，索引将包含一个被排序的完整节点树的列表。节点被按照XML元素名和选择的属性值（如果"attr"参数不等于NULL）进行排序。

**mxmlIndexReset**

重设索引中的枚举/搜索指针并且返回索引中的第一个节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlIndexReset (  
 [mxml\_index\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_t) \*ind  
 );

**参数**

ind

准备重设的索引

**返回值**

第一个节点或者NULL代表索引为空。

**说明**

这个函数需要被首先调用，在第一次使用函数mxmlIndexEnum() 或 mxmlIndexFind()之前。

**mxmlLoadFd**

载入一个文件描述符到一个XML节点树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlLoadFd (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 int fd,  
 [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t) cb  
 );

**参数**

top

顶部节点

fd

需要进行读取的文件描述符

cb

回调函数或者 MXML\_NO\_CALLBACK

**返回值**

第一个节点或者NULL代表文件不能被读取。

**说明**

在指定文件中的所有节点将被添加到所指定的顶部节点。如果没有"top"顶部节点被提供，这个XML文件必须是规范的并且整个文件只有一个父节点为<?xml> 。 回调函数返回的值类型将被使用到子节点。如果 MXML\_NO\_CALLBACK 参数被指定，那么所有的子节点将都会是MXML\_ELEMENT 或者 MXML\_TEXT 其中之一的节点。  
  
 常量 MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, MXML\_REAL\_CALLBACK, 和 MXML\_TEXT\_CALLBACK 定义了将载入指定类型的子节点。

**mxmlLoadFile**

载入一个文件到一个XML节点树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlLoadFile (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 FILE \*fp,  
 [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t) cb  
 );

**参数**

top

顶级节点

fp

准备读取的文件

cb

回调函数或MXML\_NO\_CALLBACK

**返回值**

第一个节点或者NULL代表文件不能被读取。

**说明**

在指定文件中的所有节点将被添加到所指定的顶部节点。如果没有"top"顶部节点被提供，这个XML文件必须是规范的并且整个文件只有一个父节点为<?xml> 。 回调函数返回的值类型将被使用到子节点。如果 MXML\_NO\_CALLBACK 参数被指定，那么所有的子节点将都会是MXML\_ELEMENT 或者 MXML\_TEXT 其中之一的节点.  
  
 常量 MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, MXML\_REAL\_CALLBACK, 和 MXML\_TEXT\_CALLBACK 定义了将载入指定类型的子节点。

**mxmlLoadString**

载入一个文件到一个XML节点树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlLoadString (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 const char \*s,  
 [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t) cb  
 );

**参数**

top

顶级节点

s

准备读取的字符串

cb

回调函数或MXML\_NO\_CALLBACK

**返回值**

第一个节点或者NULL代表字符串中有错误。

**说明**

在指定字符串中的所有节点将被添加到所指定的顶部节点。如果没有"top"顶部节点被提供，这个XML字符串必须是规范的并且整个文件只有一个父节点为<?xml> 。 回调函数返回的值类型将被使用到子节点。如果 MXML\_NO\_CALLBACK 参数被指定，那么所有的子节点将都会是MXML\_ELEMENT 或者 MXML\_TEXT 其中之一的节点.  
  
 T常量 MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, MXML\_REAL\_CALLBACK, 和 MXML\_TEXT\_CALLBACK 定义了将载入指定类型的子节点。

**Mini-XML 2.3** **mxmlNewCDATA**

创建一个新的CDATA 节点.

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewCDATA (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 const char \*data  
 );

**参数**

parent

父节点或者MXML\_NO\_PARENT

data

数据字符串

**返回值**

新的节点

**说明**

新的CDATA节点将被添加到指定父节点的子节点列表的最后，常量 MXML\_NO\_PARENT 可以被用来指定新的CDATA节点没有父节点。数据字符串必须是以空字符结尾，并被拷贝到新的CDATA节点。CDATA节点使用 MXML\_ELEMENT 节点类型。

**Mini-XML 2.1** **mxmlNewCustom**

创建一个新的用户自定义数据节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewCustom (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 void \*data,  
 [mxml\_custom\_destroy\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_destroy_cb_t) destroy  
 );

**参数**

parent

父节点或者 MXML\_NO\_PARENT

data

指向数据的指针

destroy

销毁数据使用的函数

**返回值**

新节点

**说明**

新的自定义节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的自定义节点没有父节点。NULL可以被通过，当数据节点不是动态分配或者是独立管理时。

最后一句NULL应该是指参数：destroy，表示不使用销毁函数。 Z.F

**mxmlNewElement**

创建一个新的XML元素节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewElement (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 const char \*name  
 );

**参数**

parent

父节点或MXML\_NO\_PARENT

name

XML元素名称

**返回值**

新节点

**说明**

新的XML元素节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的XML元素节点没有父节点。

**mxmlNewInteger**

创建一个新的整数节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewInteger (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 int integer  
 );

**参数**

parent

父节点或MXML\_NO\_PARENT

integer

整形值

**返回值**

新节点

**说明**

新的整数节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的整数节点没有父节点。

**mxmlNewOpaque**

创建一个新的不透明字符串节点

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewOpaque (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 const char \*opaque  
 );

**参数**

parent

父节点或MXML\_NO\_PARENT

opaque

不透明字符串

**返回值**

新节点

**说明**

新的不透明字符串节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的不透明字符串节点没有父节点。这个字符串必须是空字符结尾并被拷贝到新节点。

**mxmlNewReal**

创建一个新的浮点数节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewReal (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 double real  
 );

**参数**

parent

父节点或MXML\_NO\_PARENT

real

浮点数值

**返回值**

新节点

**说明**

新的浮点数节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的浮点数节点没有父节点。

**mxmlNewText**

创建新的文本分段节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewText (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 int whitespace,  
 const char \*string  
 );

**参数**

parent

父节点或MXML\_NO\_PARENT

whitespace

1 = 有前导空格, 0 = 没有空格

string

字符串

**返回值**

新节点

**说明**

新的文本节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的文本节点没有父节点。参数：whitespace被用在指定是否在这个节点前面有前导空格。文本字符串必须时以空字符结尾并被拷贝到新的节点。

**mxmlNewTextf**

创建一个新的格式化的文本分段节点

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewTextf (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*parent,  
 int whitespace,  
 const char \*format,  
 ...  
 );

**参数**

parent

父节点或MXML\_NO\_PARENT

whitespace

1 = 有前导空格, 0 = 没有空格

format

"printf"风格的格式化字符串

...

需要的附加参数

**返回值**

新节点

**说明**

新的文本节点将被添加到指定父节点的子节点列表的最后。常量 MXML\_NO\_PARENT 可以被用来指定新的文本节点没有父节点。参数：whitespace被用在指定是否在这个节点前面有前导空格。格式化字符串必须时以空字符结尾并被格式化到新的节点。

**Mini-XML 2.3** **mxmlNewXML**

创建一个新的XML文档树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlNewXML (  
 const char \*version  
 );

**参数**

version

使用的版本号

**返回值**

新的 "?xml" 节点

**说明**

参数 "version" 指定了放在"?xml" 元素节点中的版本号。如果为NULL则假定为 "version 1.0"。

**Mini-XML 2.3** **mxmlRelease**

释放一个节点。

int mxmlRelease (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node  
 );

**参数**

node

节点

**返回值**

新的引用计数

**说明**

当引用计数为0时，这个节点（以及所有子节点）被通过函数 mxmlDelete() 所删除。

**mxmlRemove**

移除一个节点从它的父节点中。

void mxmlRemove (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node  
 );

**参数**

node

被移除的节点

**说明**

不释放节点使用的内存，使用函数 mxmlDelete() 来释放。如果这个节点没有父节点则这个函数不做任何事。

**Mini-XML 2.3** **mxmlRetain**

保留一个节点

int mxmlRetain (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node  
 );

**参数**

node

节点

**返回值**

新的引用计数

**Mini-XML 2.3** **mxmlSAXLoadFd**

使用SAX回调从一个文件描述符中加载数据到一个XML节点树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlSAXLoadFd (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 int fd,  
 [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t) cb,  
 [mxml\_sax\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_cb_t) sax\_cb,  
 void \*sax\_data  
 );

**参数**

top

顶级节点

fd

进行读取的文件描述符

cb

XML节点类型回调函数或者MXML\_NO\_CALLBACK

sax\_cb

SAX 回调函数或者MXML\_NO\_CALLBACK

sax\_data

SAX 用户数据

**返回值**

第一个节点或者NULL 代表文件不能被读取。

**说明**

在指定文件中的节点将被添加到指定的顶级节点中。如果"top"节点没有提供，这个XML文件必须是规范的 并且整个文件只有一个父节点为<?xml> 。回调函数"cb"返回子节点的值类型。如果 MXML\_NO\_CALLBACK 参数被指定，那么所有的子节点将都会是MXML\_ELEMENT 或者 MXML\_TEXT 其中之一的节点。  
  
 常量 MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, MXML\_REAL\_CALLBACK, 和MXML\_TEXT\_CALLBACK 被定义用于加载指定类型的子节点。  
  
 在SAX 回调函数中("sax\_cb")，对于所有节点都必须调用mxmlRetain()函数用于保留为以后使用。否则，节点将在父节点被关闭时或者到达数据、注释、CDATA和指令节点时被删除。

**Mini-XML 2.3** **mxmlSAXLoadFile**

使用SAX回调从一个文件中加载数据到一个XML节点树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlSAXLoadFile (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 FILE \*fp,  
 [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t) cb,  
 [mxml\_sax\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_cb_t) sax\_cb,  
 void \*sax\_data  
 );

**参数**

top

顶级节点

fp

进行读取的文件

cb

XML节点类型回调函数或者MXML\_NO\_CALLBACK

sax\_cb

SAX 回调函数或者MXML\_NO\_CALLBACK

sax\_data

SAX 用户数据

**返回值**

第一个节点或者NULL 代表文件不能被读取。

**说明**

在指定文件中的节点将被添加到指定的顶级节点中。如果"top"节点没有提供，这个XML文件必须是规范的 并且整个文件只有一个父节点为<?xml> 。回调函数"cb"返回子节点的值类型。如果 MXML\_NO\_CALLBACK 参数被指定，那么所有的子节点将都会是MXML\_ELEMENT 或者 MXML\_TEXT 其中之一的节点。  
  
 常量 MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, MXML\_REAL\_CALLBACK, 和MXML\_TEXT\_CALLBACK 被定义用于加载指定类型的子节点。  
  
 在SAX 回调函数中("sax\_cb")，对于所有节点都必须调用mxmlRetain()函数用于保留为以后使用。否则，节点将在父节点被关闭时或者到达数据、注释、CDATA和指令节点时被删除。

**Mini-XML 2.3** **mxmlSAXLoadString**

使用SAX回调从一个字符串中加载数据到一个XML节点树。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlSAXLoadString (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 const char \*s,  
 [mxml\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_load_cb_t) cb,  
 [mxml\_sax\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_cb_t) sax\_cb,  
 void \*sax\_data  
 );

**参数**

top

顶级节点

s

准备加载的字符串

cb

XML节点类型回调函数或者MXML\_NO\_CALLBACK

sax\_cb

SAX 回调函数或者MXML\_NO\_CALLBACK

sax\_data

SAX 用户数据

**返回值**

第一个节点或者NULL 代表文件不能被读取。

**说明**

在指定字符串中的节点将被添加到指定的顶级节点中。如果"top"节点没有提供，这个XML字符串必须是规 范的并且整个文件只有一个父节点为<?xml> 。回调函数"cb"返回子节点的值类型。如果 MXML\_NO\_CALLBACK 参数被指定，那么所有的子节点将都会是MXML\_ELEMENT 或者 MXML\_TEXT 其中之一的节点。  
  
 常量 MXML\_INTEGER\_CALLBACK, MXML\_OPAQUE\_CALLBACK, MXML\_REAL\_CALLBACK, 和MXML\_TEXT\_CALLBACK 被定义用于加载指定类型的子节点。  
  
 在SAX 回调函数中("sax\_cb")，对于所有节点都必须调用mxmlRetain()函数用于保留为以后使用。否则，节点将在父节点被关闭时或者到达数据、注释、CDATA和指令节点时被删除。

**mxmlSaveAllocString**

保存一个XML节点树到一个内部分配的字符串。

char \*mxmlSaveAllocString (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 [mxml\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_save_cb_t) cb  
 );

**参数**

node

准备要写入的节点

cb

空白回调函数或者MXML\_NO\_CALLBACK

**返回值**

分配的字符串或者 NULL

**说明**

这个函数返回一个指向字符串的指针包含了描述整个XML节点树的文本。当你使用完这个字符串后需要使用free()函数来释放。如果这个节点产生了一个空字符或者字符串分配失败将返回NULL。  
  
 回调函数参数指定了一个函数用来在每个XML元素之前或者之后返回一个空白字符串或者NULL。如果指定了MXML\_NO\_CALLBACK，空格将仅被 添加到具有前导空格的MXML\_TEXT节点前面（node->value->text->whitespace = 1）和在一个打开XML元素标签的属性名称前面。

**mxmlSaveFd**

保存一个XML节点树到一个文件描述符。

int mxmlSaveFd (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 int fd,  
 [mxml\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_save_cb_t) cb  
 );

**参数**

node

准备要写入的节点

fd

准备写入的文件描述符

cb

空白回调函数或者MXML\_NO\_CALLBACK

**返回值**

成功返回 0，错误返回 -1。

**说明**

回调函数参数指定了一个函数用来在每个XML元素之前或者之后返回一个空白字符串或者NULL。如果指定了MXML\_NO\_CALLBACK，空格将仅被 添加到具有前导空格的MXML\_TEXT节点前面（node->value->text->whitespace = 1）和在一个打开XML元素标签的属性名称前面。

**mxmlSaveFile**

保存一个XML节点树到一个文件。

int mxmlSaveFile (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 FILE \*fp,  
 [mxml\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_save_cb_t) cb  
 );

**参数**

node

准备要写入的节点

fp

准备写入的文件。

cb

空白回调函数或者MXML\_NO\_CALLBACK

**返回值**

成功返回 0，错误返回 -1。

**说明**

回调函数参数指定了一个函数用来在每个XML元素之前或者之后返回一个空白字符串或者NULL。如果指定了MXML\_NO\_CALLBACK，空格将仅被 添加到具有前导空格的MXML\_TEXT节点前面（node->value->text->whitespace = 1）和在一个打开XML元素标签的属性名称前面。

**mxmlSaveString**

保存一个XML节点树到一个字符串。

int mxmlSaveString (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 char \*buffer,  
 int bufsize,  
 [mxml\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_save_cb_t) cb  
 );

**参数**

node

准备要写入的节点

buffer

字符串缓冲区

bufsize

字符串缓冲区大小

cb

空白回调函数或者MXML\_NO\_CALLBACK

**返回值**

字符串大小

**说明**

这个函数返回字符串需要字节总数，但是最多拷贝(bufsize-1)个字符到指定的buffer中。

如：一个XML树字符串长度为200个字节，但缓冲区只有100字节，则返回200，但只拷贝了99个字符，最后一个为空字符。Z.F

回调函数参数指定了一个函数用来在每个XML元素之前或者之后返回一个空白字符串或者NULL。如果指定了MXML\_NO\_CALLBACK，空格将仅被 添加到具有前导空格的MXML\_TEXT节点前面（node->value->text->whitespace = 1）和在一个打开XML元素标签的属性名称前面。

**Mini-XML 2.3** **mxmlSetCDATA**

设置一个CDATA元素节点的名称。

int mxmlSetCDATA (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*data  
 );

**参数**

node

准备设置的节点

data

新的数据字符串

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个CDATA节点则节点不发生改变。

**Mini-XML 2.1** **mxmlSetCustom**

对一个用户自定义数据节点设置数据和销毁回调函数。

int mxmlSetCustom (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 void \*data,  
 [mxml\_custom\_destroy\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_destroy_cb_t) destroy  
 );

**参数**

node

被设置的节点

data

新的数据指针

destroy

新的销毁回调函数

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个用户自定义节点则节点不发生改变。

**mxmlSetCustomHandlers**

设置对于自定义数据的处理回调函数。

void mxmlSetCustomHandlers (  
 [mxml\_custom\_load\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_load_cb_t) load,  
 [mxml\_custom\_save\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_save_cb_t) save  
 );

**参数**

load

加载回调函数

save

保存回调函数

**说明**

加载回调函数接收一个节点指针和数据字符串，成功时必须返回 0，错误时返回一个非0值。  
  
 保存回调函数接收一个节点指针，成功时必须返回一个使用malloc分配的字符串，错误时返回NULL。

**mxmlSetElement**

设置XML元素节点的名字。

int mxmlSetElement (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*name  
 );

**参数**

node

被设置的节点

name

新的名称字符串

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个XML元素节点则节点不发生改变。

**mxmlSetErrorCallback**

设置错误信息回调函数。

void mxmlSetErrorCallback (  
 [mxml\_error\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_error_cb_t) cb  
 );

**参数**

cb

错误回调函数

**mxmlSetInteger**

设置一个整数节点的值。

int mxmlSetInteger (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 int integer  
 );

**参数**

node

被设置的节点

integer

整数值

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个整数节点则节点不发生改变。

**mxmlSetOpaque**

设置一个不透明字符串节点的值。

int mxmlSetOpaque (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 const char \*opaque  
 );

**参数**

node

被设置的节点

opaque

不透明字符串

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个不透明字符串节点则节点不发生改变。

**mxmlSetReal**

设置一个浮点数节点的值。

int mxmlSetReal (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 double real  
 );

**参数**

node

被设置的节点

real

浮点数值

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个浮点数节点则节点不发生改变。

**mxmlSetText**

设置一个文本节点的值。

int mxmlSetText (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 int whitespace,  
 const char \*string  
 );

**参数**

node

被设置的节点

whitespace

1 = 有前导空格, 0 = 没有前导空格

string

字符串

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个文本节点则节点不发生改变。

**mxmlSetTextf**

设置一个文本节点的值为一个格式化的字符串。

int mxmlSetTextf (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 int whitespace,  
 const char \*format,  
 ...  
 );

**参数**

node

被设置的节点

whitespace

1 = 有前导空格, 0 = 没有前导空格

format

"printf"风格的格式化字符串

...

需要的附加参数

**返回值**

成功返回 0，失败返回 -1。

**说明**

如果这个节点不是一个文本节点则节点不发生改变。

**Mini-XML 2.3** **mxmlSetWrapMargin**

设置在保存XML数据时的自动折行位置。

void mxmlSetWrapMargin (  
 int column  
 );

**参数**

column

自动折行的列

**说明**

当"column" is <= 0时取消自动折行。

**mxmlWalkNext**

遍历到XML树中的下一个逻辑节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlWalkNext (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 int descend  
 );

**参数**

node

当前节点

top

顶级节点

descend

在XML树中的向下搜索模式 - MXML\_DESCEND, MXML\_NO\_DESCEND, 或者 MXML\_DESCEND\_FIRST。

**返回值**

下一个节点或者NULL

**说明**

"descend"参数控制下一个节点是否考虑第一个子节点。"top"参数约束了遍历这个节点的所有子节点。

**mxmlWalkPrev**

遍历到XML树中的上一个逻辑节点。

[mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*mxmlWalkPrev (  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*node,  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*top,  
 int descend  
 );

**参数**

node

当前节点

top

顶级节点

descend

在XML树中的向下搜索模式 - MXML\_DESCEND, MXML\_NO\_DESCEND, 或者 MXML\_DESCEND\_FIRST。

**返回值**

上一个节点或者NULL

**说明**

"descend"参数控制下一个节点是否考虑第一个子节点。"top"参数约束了遍历这个节点的所有子节点。

**数据类型**

**mxml\_attr\_t**

XML元素节点的属性值。

typedef struct [mxml\_attr\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_attr_s) mxml\_attr\_t;

**mxml\_custom\_destroy\_cb\_t**

自定义数据销毁回调函数原型

typedef void (\*mxml\_custom\_destroy\_cb\_t)(void \*);

**mxml\_custom\_load\_cb\_t**

自定义数据加载回调函数原型

typedef int (\*mxml\_custom\_load\_cb\_t)([mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*, const char \*);

**mxml\_custom\_save\_cb\_t**

自定义数据保存回调函数原型

typedef char \*(\*mxml\_custom\_save\_cb\_t)([mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*);

**Mini-XML 2.1** **mxml\_custom\_t**

自定义XML类型值

typedef struct [mxml\_custom\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_s) mxml\_custom\_t;

**mxml\_element\_t**

XML元素值

typedef struct [mxml\_element\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_element_s) mxml\_element\_t;

**mxml\_error\_cb\_t**

错误回调函数原型

typedef void (\*mxml\_error\_cb\_t)(const char \*);

**mxml\_index\_t**

XML节点索引

typedef struct [mxml\_index\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_index_s) mxml\_index\_t;

**mxml\_load\_cb\_t**

加载回调函数

typedef mxml\_type\_t (\*mxml\_load\_cb\_t)([mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*);

**mxml\_node\_t**

XML节点

typedef struct [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s) mxml\_node\_t;

**mxml\_save\_cb\_t**

保存回调函数

typedef const char \*(\*mxml\_save\_cb\_t)([mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*, int);

**mxml\_sax\_cb\_t**

SAX 回调函数

typedef void (\*mxml\_sax\_cb\_t)([mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*, mxml\_sax\_event\_t, void \*);

**mxml\_sax\_event\_t**

SAX 事件类型.

typedef enum [mxml\_sax\_event\_e](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_sax_event_e) mxml\_sax\_event\_t;

**mxml\_text\_t**

XML文本节点值

typedef struct [mxml\_text\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_text_s) mxml\_text\_t;

**mxml\_value\_t**

XML节点值

typedef union [mxml\_value\_u](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_value_u) mxml\_value\_t;

**Structures**

**mxml\_attr\_s**

XML元素的属性值

struct mxml\_attr\_s {  
 char \*name;  
 char \*value;  
 };

**成员**

name

属性名

value

属性值

**Mini-XML 2.1** **mxml\_custom\_s**

自定义XML节点值

struct mxml\_custom\_s {  
 void \*data;  
 [mxml\_custom\_destroy\_cb\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_destroy_cb_t) destroy;  
 };

**成员**

data

指向一个自定义数据的指针（已分配的）。

destroy

指向销毁回调函数的指针

**mxml\_element\_s**

XML元素值

struct mxml\_element\_s {  
 [mxml\_attr\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_attr_t) \*attrs;  
 char \*name;  
 int num\_attrs;  
 };

**成员**

attrs

包含的所有属性

name

XML元素名称

num\_attrs

包含的属性数量

**mxml\_index\_s**

XML节点索引

struct mxml\_index\_s {  
 int alloc\_nodes;  
 char \*attr;  
 int cur\_node;  
 [mxml\_node\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_t) \*\*nodes;  
 int num\_nodes;  
 };

**成员**

alloc\_nodes

在索引中的已分配的节点数

attr

节点使用的属性或者NULL

cur\_node

当前节点

nodes

包含的节点数组

num\_nodes

在索引中的节点总数

**mxml\_node\_s**

XML 节点。

struct mxml\_node\_s {  
 struct [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s) \*child;  
 struct [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s) \*last\_child;  
 struct [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s) \*next;  
 struct [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s) \*parent;  
 struct [mxml\_node\_s](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_node_s) \*prev;  
 int ref\_count;  
 mxml\_type\_t type;  
 void \*user\_data;  
 [mxml\_value\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_value_t) value;  
 };

**成员**

child

第一个子节点

last\_child

最后一个子节点

next

同级的下一个节点（在同一个父节点下）

parent

父节点

prev

同级的上一个节点（在同一个父节点下）

ref\_count

使用计数器（引用计数）

type

节点类型

user\_data

用户关联数据

value

节点值

**mxml\_text\_s**

XML文本节点值

struct mxml\_text\_s {  
 char \*string;  
 int whitespace;  
 };

**成员**

string

字符串片断

whitespace

是否有前导空格?

**联合**

**mxml\_value\_u**

XML节点值。

union mxml\_value\_u {  
 [mxml\_custom\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_custom_t) custom;  
 [mxml\_element\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_element_t) element;  
 int integer;  
 char \*opaque;  
 double real;  
 [mxml\_text\_t](file:///D:/我的文档/Desktop/minixml-cn.html#mxml_text_t) text;  
 };

**成员**

custom Mini-XML 2.1

自定义数据

element

元素

integer

整数

opaque

不透明字符串

real

浮点数

text

文本片断

**常量**

**mxml\_sax\_event\_e**

SAX 事件类型。

**常量**

MXML\_SAX\_CDATA

CDATA 节点

MXML\_SAX\_COMMENT

注释节点

MXML\_SAX\_DATA

数据节点

MXML\_SAX\_DIRECTIVE

处理指令节点

MXML\_SAX\_ELEMENT\_CLOSE

XML元素关闭节点

MXML\_SAX\_ELEMENT\_OPEN

XML元素开放节点

**mxml\_type\_e**

XML 节点类型

**常量**

MXML\_CUSTOM Mini-XML 2.1

自定义数据

MXML\_ELEMENT

XML 元素并包含属性

MXML\_IGNORE Mini-XML 2.3

忽略/抛弃的节点

MXML\_INTEGER

整数值

MXML\_OPAQUE

透明字符串值

MXML\_REAL

浮点数值

MXML\_TEXT

文本片断