DOM XML Library for Palm OS

Version 1.1.5
Copyright © 2003-2006 PDADevelopers.com

PDADevelopers.com

DOM XML Library for Palm OS
Table of Contents

Introduction

This is a lightweight library designed for C/C++ developers who needs to manage XML documents consuming as little resources as possible.

Features

- Simplified DOM-style parser.
- Pure C library.
- C++ Object Wrapper Interface included (class Wrapper).
- Traverse XML documents using a similar DOM syntax.
- Search children and attributes by name.
- Modify/Write capabilities.

Limitations
- No DTD or XML-Schema validation is made.
- XML is not strictly checked.
- 64KB capacity for the XML Document text (Palm OS limitation).
- !DOCTYPE, ENTITY, NOTATION not fully parsed.
- XPath notation no implemented.

Coming Soon Features
- !DOCTYPE fully parsed, including ENTITY, NOTATION

General Notes

This library assumes that the input document has a valid format. This means that if the format is not valid, you can obtain unpredictable result because the error can or cannot be detected. If the error is detected document parsing is aborted, FALSE is returned and DOM tree isn't created. Otherwise you will obtain a DOM tree but its content will be not representative.

All pointers returned by this library are managed by the library and must not be explicitly released.
All nodes created from the document tree will be released when you execute **xml_release** or **xml_deleteFreeNodes** so you are free to ignore internal node references and release them with these two functions.

Beginners Tutorial

Creating a new XML DOM Tree by Parsing a Text

Let's suppose you have a *char *source* string containing the text representation of an XML document, and you want to parse it to create the DOM Tree representation of the document. Let's suppose the text contains this:

<?xml version="1.0"?>
<!--Clients example-->
<rows recordset="clients">
        <record id="1">
                <name>Bill Gates</name>
        </record>
</rows>

In order to create the tree, you will need first to define the variable that will reference to the root node of the document, and initialize it:

```
HXML h = xml_create();
```

Next, you will need to call the parsing routine:

```
xml_processDocument(h, source);
```

At this point, the XML DOM tree is already created. Now you can navigate the structure using the utility functions. For example, lets get the first node and print the number of siblings:

```
XMLNode rows = xml_getElementsByTagName(xml_getRoot(h),"rows");
if (rows) {
    print("Recordset : ");
    println(xml_getAttribute(rows, "recordset", valueTemp, valueTempS));

    print("RecordCount : ");
    println(xml_childCount(rows));
}
```

We also loop through the node structure:

```
XMLNode record = xml_firstChild(rows);
while (record){
    println("--------------------------------- -");
    print("Record id : ");
    println(xml_getAttribute(record, "id", valueTemp, valueTempSize));

    print("name : ");
    println( xml_text(xml_getElementsByTagName(record, "name"),
              valueTemp, valueTempSize));

    record = xml_nextSibling(record);
}
```

Finally, don't forget to release the document:
```
xml_release(h);
```

Creating a new XML DOM Tree by Inserting Nodes

First, you must define the variables that will hold your information:

```
char *buffer = sampleBuffer;
HXML h = xml_create();
XMLNode root;
```

Then, you can initialize the root element, which will become the handle of the main XML DOM tree representation:

```
root = xml_createRoot(h);
```

Next, you can start adding your tree, node by node. We will construct the following (extremely simple) XML document:

<?xml version="1.0"?>
<!--Clients example-->
<rows recordset="clients">
<record id="1">
<name>Bill Gates</name>
</record>
</rows>

```
xml_appendChild(root, xml_createProcessingInstruction(h));
xml_appendChild(root, xml_createComment(h, "Clients example"));

// Create rows node and initialize attributes
XMLNode rows = xml_createElement(h, "rows");
xml_appendChild(root, rows);
xml_appendAttribute(rows, xml_createAttribute(h, "recordset","clients"));

// Create record node and initialize attributes
XMLNode record = xml_createElement(h, "record");
xml_appendChild( rows, record );
xml_appendAttribute( record, xml_createAttribute(h, "id", 1));

// Create name node and initialize attributes
XMLNode address = xml_createElement(h, "name");
xml_appendChild( record, name );
xml_appendChild( name, xml_createTextNode(h, "Bill Gates"));
```

Congratulations! Now we have the document ready, referenced by the **h** handle. In case you want to convert it to string (for storing or sending through any communication channel) you can call the conversion routine:

```
xml_toXML(h, buffer, SAMPLE_BUFFER_SIZE);
```

Finally, don't forget to release the document:

```
xml_release(h);
```

Main Routines

HXML xml_create()
This function creates a new XMLParser and returns a handle to it. You must call **xml_release** with the handle returned by **xml_create** to release all the memory allocated by the parser.
There are no limits in the number of parsers you can create.

void xml_release(HXML h)
Release the XMLParser pointed by *h*. This function does not release the memory allocated by the document text passed as parameter in the **xml_processDocument** function, so you must free the memory on your own. All the **XMLNodes** obtained from *h* become invalid references.

char* xml_getDocument(HXML h)
Return a pointer to the Document text.

BOOL xml_processDocument(HXML h, char *XMLDoc)
Parses the document *XMLDoc*. *XMLDoc* is an ASCII null terminated string containing the XML content to parse. Returns TRUE if the document was successfully processed and FALSE otherwise. If FALSE is returned you can get the error detail calling the functions **xml_getErrorID**, **xml_getErrorMsg** and **xml_getErrorPos**.

Remarks
When calling **xml_processDocument** keep in mind the following things:
- Every time you call **xml_processDocument** all references to previously parsed documents and all structures instantiated in previous calls are deleted. The parser never releases the document passed as argument. You must free the memory in use by the document by your own.
- If **xml_processDocument** returns FALSE only the pointer **XMLDoc** is stored internally. No additional structures are kept so you can release the document.
- Otherwise if TRUE is returned all DOM tree information keeps references to the document so you MUST NOT release the memory allocated by it. If you do that **xml_nodeValue** and **xml_nodeName** will get undefined.

Navigation Routines

XMLNode xml_getRoot(HXML h)
Returns the Root node of the document. If there isn't a root element it returns NULL.

XMLNode xml_parentNode(XMLNode node)
Returns the parent node. If *node* is the ROOT node, it returns NULL.

XMLNode xml_firstChild(XMLNode node)
Returns the first child node. If there are no such children, returns NULL.

XMLNode xml_lastChild(XMLNode node)
Returns the last child node. If there are no such children, returns NULL.

XMLNode xml_firstAttribute(XMLNode node)
Returns the first attribute of the node. If there is no such attribute, returns NULL.

XMLNode xml_lastAttribute(XMLNode node)
Returns the last attribute of the node. If there is no such attribute, returns NULL.

XMLNode xml_nextSibling(XMLNode node)
Returns the next sibling of the node in the parent's (children/attributes) list. If there is no such sibling, returns NULL.

XMLNode xml_previousSibling(XMLNode node)
Returns the previous sibling of the node in the parent's (children/attributes) list. If there is no such sibling, returns NULL.

XMLNode xml_getElementsByTagName_first (XMLNode node, const char*name)
Returns the first child in the **node** children list for which the result of **xml_nodeName** is exactly equal to **name.** If no such children exits, NULL is returned.

XMLNode xml_getElementsByTagName_next (XMLNode node, const char*name)
Returns the next sibling of the node in the parent's children list for which the result of **xml_nodeName** is exactly equal to **name.** If no such sibling exits NULL is returned.

XMLNode xml_getAttributeNode(XMLNode node, const char*name)
Returns the attribute node from the **node** attributes list for which the result of **xml_nodeName** is exactly equal to **name.** If no such attribute exits NULL is returned.

XMLNode xml_getAttribute(XMLNode node, const char* name, char *buffer, ushort size)

Returns the attribute node from the **node** attributes list for which the result of **xml_nodeName** is exactly equal to **name.** If no such attribute exits NULL is returned. In case of the existence of the node, **buffer** is filled with the result of **xml_nodeValue** otherwise is filled with an empty string.

uint   xml_attributeCount(XMLNode node)
Returns the number of items in the attributes list of the node

uint   xml_childCount(XMLNode node)
Returns the number of items in the children list of the node. If the node doesn't have children it returns 0.

BOOL   xml_hasChildNodes(XMLNode node)
Return TRUE if the node has child items.

XMLNode xml_getAttributeNode(XMLNode node, const char*name)
Returns the attribute node from the **node** attributes list for which the result of **xml_nodeName** is exactly equal to **name.** If no such attribute exits NULL is returned.

XMLNode xml_getAttribute(XMLNode node, const char* name, char *buffer, ushort size)
Returns the attribute node from the **node** attributes list for which the result of **xml_nodeName** is exactly equal to **name.** If no such attribute exits NULL is returned. In case of the existence of the node, **buffer** is filled with the result of **xml_nodeValue** otherwise is filled with an empty string.

Error routines

ushort xml_getErrorID(HXML h)
Returns the *code* of the last Error occurred:

ERR_NONE (0)	No error occurred.
ERR_CLOSE_TAG (1)	End tag does not match the start tag.
ERR_INV_CHAR (2)	Expecting a char while another char or 'end of file' found.
ERR_INV_SINTAX (3)	Incorrect syntax.
ERR_ELEM_NCLOSED (4)	Element not closed.
ERR_ELEM_NREC (5)	Element not recognized. E.g. <!NOTATION>

char *xml_getErrorMsg(HXML h)
Returns a string containing a detailed message of the error. NULL is returned if no error occurred.

char *xml_getErrorPos(HXML h)
Returns the offset where the parser stopped processing the document. 0 is returned if no error occurred.

Node Information Routines

uint xml_nodeType(XMLNode node)

Specifies the XML Document Object Model (DOM) node type, which determines valid values and whether the node can have child nodes.

NODE_ELEMENT (1)	The node represents an element. An element node can have the following child node types: Element, Text, Comment, ProcessingInstruction, CDATASection. An element node can be the child of the Document and Element nodes.
NODE_ATTRIBUTE (2)	The node represents an attribute of an element.
NODE_TEXT (3)	The node represents the text content of a tag. A text node cannot have any child nodes. A text node can appear as the child node of the Attribute, DocumentFragment, Element, and EntityReference nodes.
NODE_CDATA_SECTION (4)	The node represents a CDATA section in the XML source. CDATA sections are used to escape blocks of text that would otherwise be recognized as markup. A CDATA section node cannot have any child nodes. A CDATA section node can appear as the child of the DocumentFragment, EntityReference, and Element nodes.
NODE_PROCESSING_INSTRUCTION (7)	The node represents a processing instruction from the XML document. A processing instruction node cannot have any child nodes. A processing instruction node can appear as the child of the Document, DocumentFragment, and Element nodes.
NODE_COMMENT (8)	The node represents a comment in the XML document. A comment node cannot have any child nodes. A comment node can appear as the child of Document, DocumentFragment, and Element nodes.
NODE_DOCUMENT (9)	The node represents a document object, which, as the root of the document tree, provides access to the entire XML document. A document node can have the following child node types: Element, ProcessingInstruction, Comment, and DocumentType. A document node cannot appear as the child of any node types.
NODE_DOCUMENT_TYPE (10)	The node represents the document type declaration, indicated by the <!DOCTYPE > tag. A document type node can have the following child node types: Notation and Entity. A document type node can appear as the child of the Document node.

char* xml_nodeName(XMLNode node, char *buffer, uint size)

Returns the qualified name of the element, attribute or a fixed string for other node types.

Remarks

Always returns a non-empty string. The **xml_nodeName** returns the qualified name for the element, or attribute. For example, it returns xxx:yyy for the element <xxx:yyy>.

The node name value varies, depending on the **xml_nodeType** result.

NODE_ATTRIBUTE	Contains the name of the attribute.
NODE_CDATA_SECTION	Contains the literal string "#cdata-section".
NODE_COMMENT	Contains the literal string "#comment".
NODE_DOCUMENT	Contains the literal string "#document".
NODE_DOCUMENT_TYPE	Contains the name of the document type; for example, xxx in <!DOCTYPE xxx …>.
NODE_ELEMENT	Contains the name of the XML tag, with any namespace prefix included if present.
NODE_PROCESSING_INSTRUCTION	Contains the target; the first token following the <? characters.
NODE_TEXT	Contains the literal string "#text".

char* xml_nodeValue(XMLNode node, char *buffer, uint size)

Contains the text associated with the node.

This value depends of the **xml_nodeType** result.

NODE_ATTRIBUTE	Contains a string representing the value of the attribute.
NODE_CDATA_SECTION	Contains a string representing the text stored in the CDATA section.
NODE_COMMENT	Contains the content of the comment, exclusive of the comment's start and end sequence.
NODE_DOCUMENT, NODE_ELEMENT	Contains Null. Note that attempting to set the value of nodes of these types generates an error.
NODE_DOCUMENT_TYPE	Contains all extra data in <!DOCTYPE name ….> tag.
NODE_PROCESSING_INSTRUCTION	Contains the content of the processing instruction, excluding the target
NODE_TEXT	Contains a string representing the text stored in the text node.

char* xml_text(XMLNode node, char *buffer, uint size)

Contains the text content of the node and its subtrees.

Edition Routines

XMLNode   xml_createRoot(HXML h)

Creates a document tree and returns a reference to its root node. If there was already a document tree in *h* it is released.
Returns NULL if the root of the new document can't be created.

XMLNode   xml_createAttribute(HXML h, const char *name, const char *value = "")

Creates a attribute node and returns a reference to it. Set its name and value to the supplied data.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.
Returns NULL if the node can't be created.

XMLNode   xml_createCDATASection(HXML h, const char* data = "")

Creates a CDATA section node that contains the supplied data, and returns a reference to it.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.
Returns NULL if the node can't be created.

XMLNode   xml_createTextNode(HXML h, const char *text = "")

Creates a text node that contains the supplied data and returns a reference to it.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.
Returns NULL if the node can't be created.

XMLNode   xml_createComment(HXML h, const char *comment = "")

Creates a comment node that contains the supplied data and returns a reference to it.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.
Returns NULL if the node can't be created.

XMLNode   xml_createDocumentFragment(HXML h)

Creates a new document fragment node and returns a reference to it.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.
Returns NULL if the node can't be created.

XMLNode   xml_createElement(HXML h, const char *tagName)

Creates an element node using the specified name and returns a reference to it.
Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.
Returns NULL if the node can't be created.

XMLNode   xml_createProcessingInstruction(HXML h, const char *target = "xml", const char *data="version=\1.0")

Creates a processing instruction node that contains the supplied target and data and returns a reference to it.

Although this method creates the new object in the context of this document, it does not automatically add the new object to the document tree. It is a *free node*.

Returns NULL if the node can't be created.

BOOL   xml_appendChild(XMLNode parent, XMLNode node)

Appends *node* as the last child of the *parent. Node* must be a *free node. Node* must be of a child type valid of *parent.*

Returns FALSE if the operation can't be carried out.

BOOL   xml_appendAttribute (XMLNode parent, XMLNode attr)

Appends the attribute node as the last node in parent attribute list. No validation is made. *Attr* must be a *free node* and *parent* must accept attributes. *Attr* must be of type attribute.

Returns FALSE if the operation can't be carried out.

XMLNode   xml_cloneNode(XMLNode node, BOOL deep)

Clones a node and returns a reference to the new node, attributes are copied if it had them. If the *deep* flag is set all children nodes are cloned too.

Returns FALSE if the operation can't be carried out.

BOOL   xml_insertBefore(XMLNode newNode, XMLNode before)

Inserts *newNode* as the previous sibling of the *before* node.

Returns FALSE if the operation can't be carried out.

BOOL   xml_remove(XMLNode node);

Removes the node from the document tree. The node is not destroyed, it can be put back on the tree using xml_insertBefore, xml_appendChild or xml_Replace. *Node* cannot be the root of the document or a free node.

Returns FALSE if the operation can't be carried out.

BOOL   xml_replace(XMLNode newNode, XMLNode oldNode)

Replaces the *oldNode* with the *newNode.* The *newNode* must be a *free node* and *oldNode* cannot be a *free node*. After this function completes the *oldNode* is *free node* and the *newNode* is set.

Returns FALSE if the operation can't be carried out.

BOOL   xml_delete(XMLNode node)

Deletes the *node* releasing all resources in use by it, all its children and attributes. If it isn't a *free node* it is first removed from the tree and then deleted.

Returns FALSE if the operation can't be carried out.

BOOL   xml_deleteFreeNodes(HXML h);

Releases all free nodes. All references to then get invalid.

Returns FALSE if the operation can't be carried out.

uint   xml_toXML(HXML h, char *buffer, uint size)

Fill the *buffer* with the XML representation of document tree. The representation is cut-off to feet in the *size* specified.

Returns the count of characters written.

Purchasing the source code

If you are interested in purchasing the library source code, please contact **sales@pdadevelopers.com** for more information.

Revision History

Version 1.0
Initial Version.

Version 1.1
New Targets: Expanded Mode (A4/A5-relative data) and Expanded with A5-based Jumptable.

Version 1.1.5
New Targets: All modes with 4 bytes 'int'.