libxml Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purpose and Features

libxml is a fast and light XML parser library. libxml was created by extracting functions from the XML library used by Sony-made embedded devices including PlayStation®, and porting them to the PlayStation®Vita environment. Applications can easily access the information in XML documents by using this library.

Main Functions

The libxml library provides the following main features.

- Function to sequentially acquire the contents of XML documents via the SAX interface (SAX1.0)
- Function to generate the tree structure of a document from the XML document contents
- Function to operate arbitrary tree structure of XML documents (DOM Level 1 Core)
- Function to serialize a created tree structure to XML documents

Consumed Resources

The system resources used by the libxml library are listed below.

Resource	Description
Footprint	Approx. 100 KiB when PRX is loaded
Work memory	1.5 KiB or more from memory allocator given by the application
Threads	Threads are not generated internally libxml operates with threads that call API functions
Processor time	Negligible

The above listed work memory is for when SAX APIs are used. If DOM APIs are used, the work memory is increased compared to when SAX is used in order to maintain the tree structure. However, since only unique element names and attribute names are maintained, the increase in required memory is minimized to the extent that repetitions of the same name are numerous.

Embedding in Programs

The following files are required in order to use libxml.

File Name	Description
xml/xml.h	Header file
libSceXml_stub.a	Stub library file

Include xml/xml.h in the source program. (A number of header files are automatically included.)

Load also a PRX module as follows in the program.

```
if ( sceSysmoduleLoadModule(SCE_SYSMODULE_XML) != SCE_OK ) {
   //Error processing
}
```

When building a program, link libSceXml_stub.a.

Sample Program

The following programs are provided as sample programs that use the libxml library. Refer to them as needed.

sample_code/system/api_xml/sax/

This is a sample of the basic usage of the SAX interface of libxml.

sample_code/system/api_xml/dom/

This is a sample of the basic usage of the DOM interface of libxml.

Reference Materials

For details about the SAX interface, refer to the following websites.

- http://en.wikipedia.org/wiki/Simple_API_for_XML
- http://www.saxproject.org/

For details on the DOM interface, refer to the following websites.

- http://en.wikipedia.org/wiki/Document_Object_Model
- http://www.w3.org/TR/REC-DOM-Level-1/

libxml does not fully implement these specifications. Refer to the "Complied with Specifications" chapter for details.

(The above reference destination has been confirmed as of November 28, 2011. Note that pages may have been subsequently moved or its contents modified.)

2 Usage Procedure

Basic Processing Procedure

This section explains the basic processing procedure to parse XML documents. XML documents can be accessed either through the SAX interface or through the DOM interface. The basic processing flow is as follows.

(1) Specify `SCE_SYSMODULE_XML` to load the PRX module.
(2) Pass the memory allocator to generate the `Initializer` object.
(3) Generate the required objects according to the intended use and performing initialization by passing the `Initializer` object.
(4) Access the required information through the objects.
(5) Discard the objects.
(6) Discard the `Initializer` object.
(7) Unload the PRX module.

Preparations

The memory allocator must be passed when using a library. Implement `sce::Xml::MemAllocator`. After loading the PRX module, first set the memory allocator to `InitParameter`. Provide type variables and appropriately set *`mode`* and *`bufSize`* as needed.

(1) Initialization of library

(1) Load the PRX module.
(2) Generate the `sce::Xml::Initializer` object. If the object is to be retained during the utilization period, it can be generated either on a stack or on a heap.
(3) Set the implemented memory allocator to `allocator` of `InitParameter`. Set an arbitrary value for `userData`. This value is passed as an argument to the passed memory allocator.
(4) Pass `InitParameter` to `Initializer::initialize()` to perform initialization.

(2) Interface usage procedure

SAX

Perform steps (1) through (4) in the above "Initialization of library".

(5) Generate the `Sax::Parser` instance. If it is to be retained during the utilization period, it can be generated either on a stack or on a heap.
(6) Pass the `Initializer` object generated in step (2) to `Parser::initialize()` to execute initialization.
(7) Set the SAX event handler (the event handler that implements the class inheriting `Sax::DocumentHandler`) with `Parser::setDocumentHandler()`.
(8) Set the behavior of the parser with `Parser::setResolveEntity()`, etc., according to the intended use.
(9) Load the XML document to be parsed into memory. The entire document need not be loaded at one time into memory.
(10) Pass the buffer to `Parser::parse()` to start parsing. If the buffer to be passed contains the end of the XML document (including when the entire XML document is passed at one time), set the *`isFinal`* argument of `parse()` to true. If it is set to false, this indicates that the document is not yet complete.

(11) The set SAX event handler is called for each component of the XML document.

(12) Once `parse()` is called with *`isFinal`* `= false`, go on to call `parse()`.

(13) Once the processing is completed, discard the SAX event handler and the `Parser`.

DOM (to generate DOM tree from XML document)

Perform steps (1) through (4) in the above "Initialization of library".

(5) Generate the `Dom::DocumentBuilder` instance. If it is to be retained during the utilization period, it can be generated either on a stack or on a heap.

(6) Pass the `Initializer` object generated in step (2) to `DocumentBuilder::initialize()` to execute initialization.

(7) Set the behavior of the parser with `Parser::setResolveEntity()`, etc., according to the intended use.

(8) Load the XML document to be parsed into memory. The entire document need not be loaded at one time into memory.

(9) Pass the buffer to `DocumentBuilder::parse()` to start DOM tree building. If the buffer to be passed contains the end of the XML document (including when the entire XML document is passed at one time), set the *`isFinal`* argument of `parse()` to `true`. If it is set to `false`, this indicates that the document is not yet complete.

(10) Get the pointer to the `Document` with `DocumentBuilder::getDocument()`.The `Document` itself is retained in `DocumentBuilder` even after `getDocument()`.

(11) Access the DOM tree with the APIs of the `Document`.

(12) After the processing ends, discard `DocumentBuilder`.

DOM (to generate a new DOM tree)

Perform steps (1) through (4) in the above "Initialization of library".

(5) Generate the `Dom::Document` instance. If it is to be retained during the utilization period, it can be generated either on a stack or on a heap.

(6) Pass the `Initializer` object generated in step (2) to `Document::initialize()` to execute initialization.

(7) Access the DOM tree with the APIs of the `Document`.

(8) After the processing ends, discard the `Document`.

(3) Terminating

(1) After completing the use of all the libxml objects and discarding them, lastly discard the `initializer`.

(2) Unload the PRX module.

3 Complied with Specifications

Text Encoding

Only UTF-8 is complied with.

SAX

A subset of the SAX 1.0 specification is used.

DOM

A subset of the DOM Level 1 Core specification is used.

However, whereas in the DOM Level 1 specification, the base class is `Node` and operations are done on its derived classes, in libxml all operations are done on the `Document`. In libxml, `NodeId`, a 64-bit integer, is used as the method to specify `Node`. The `Node` class is offered as an interface for wrapping operations on `Document` by generating `Node` from `NodeId`.

Moreover, there is no `NamedNodeMap`. Attribute groups are referenced using `NodeList`.

Entity Resolution

`Sax::Parser` and `Dom::DocumentBuilder` include `setResolveEntity(bool)`. This is the XML parser of libxml and controls whether to process entity reference.

The default is false. In the false state, entity reference is handled as regular text.

True must be set to comply with the XML specification.

(The reason why false is set by default is that this is convenient for embedded devices that often process fixed format documents.)

Unsupported Specifications

Namespaces are not supported.

Namespace prefixes are handled as element names, and namespace URI declarations are handled as normal attributes.

DTD is not supported.

DTD in input XML documents are skipped.

4 Reference Information

Memory Allocator

Inherit and implement the `sce::Xml::MemAllocator` class at the library user level. The memory allocator is called each time memory allocation/deallocation is required. The `allocate` and `deallocate` specifications are equivalent to the standard libc functions `malloc` and `free`. If an `allocate` request is not acknowledged, the library does not function, but it is possible to optimize `deallocate` to release all the memory areas at one time following library use.

Interface to be Implemented	Description
`allocate`	This function receives the size, allocates a memory area of that size and returns the beginning pointer. When no memory area can be allocated, NULL is returned.
`deallocate`	This function releases the memory area of the pointer passed with `allocate`.

Character Strings

In libxml, that character string handlers are the `sce::xml::String` type.

This type has the `const char*` type member *`buffer`* and `size_t` type member *`size`*.

For the application to pass the `String` type to libxml, the character string entity indicated by `String::`*`buffer`* must be secured in a different area. Moreover, this entity must not be modified until returning from the API. Since there is `String::`*`size`*, the entity does not have to end with '\0'.

Regarding the `String` types received by the application, the *`buffer`* character string does not always end with '\0'. Moreover, the contents of the memory pointed by *`buffer`* must not be modified. If necessary, copy the character string of the entity pointed by *`buffer`* using *`size`*.

`String` holds only the character string pointer and length. Its use will not cause a transfer of responsibility to release the memory of the entity which contains the character string.

Errors

libxml does not use C++ exceptions. Errors are basically returned from the API functions as return values.

Initialization

Initialization processing (memory allocation, etc.) that may cause errors is not executed in the object constructors. Since each interface class has an `initialize()` API, pass `initializer` objects to this API for initialization.

Discarding Objects

Call `terminate()` for each object. Since the interface class destructor calls `terminate()`, it is not necessary to explicitly call `terminate()` when deleting objects immediately after the processing (when leaving the scope if interface class entity has been secured on a stack).

Storage Duration of Objects

Information passed by arguments to the SAX event handler (class that implements `Sax::DocumentHandler`) can be referenced only within this function. To reference values even after the processing returns from the handler, make copies as needed.

`Document` generated from an XML document with `Dom::DocumentBuilder` is held inside `DocumentBuilder`. This `Document` is held until `DocumentBuilder` is discarded or `parse()` is newly executed.

The `NodeId` acquired from `Dom::Document` is valid while the acquisition source `Document` exists.

Object Copy

Even when an object is copied by the copy constructor or by substitution (`operator=`), the held contents are not copied. The same object is referenced. The responsibility to discard the object is not transferred by copying.

SAX Event Handler

Implement by the class inheriting the `sce::Xml::Sax::DocumentHandler` class defined in the header file xml_sax_document_handler.h.

For items whose return value type is `int`, the parse processing at that point in time can be halted by returning a value other than 0. At this time, `resultXmlParseInProgress` is returned for `Parser::parse()` call.

DOM Tree Mode

libxml has two DOM tree modes, read only and read/write. `Document` generated with `Dom::DocumentBuilder` are read-only.

When a `Document` is initialized with `Document::initialize()`, it becomes a read/write `Document`. The read-only mode can be changed to the read/write mode with `Document::setWritable()`.

5 Precautions

Limitations

- To perform segmented processing of an XML document with `Sax::Parser::parse()` or `Dom::DocumentBuilder::parse()` (when using the *`isFinal`* argument as false), the text is segmented even when the input buffer boundary differs from the (UTF-8 multi-byte) character boundary. In SAX, 1 character is notified by multiple `characters` events, and DOM, it is mapped to multiple text nodes.
- In the SAX APIs, XML declarations in XML documents are PI tokens.
- In the DOM APIs, the part from the beginning of an XML document until the first element is not recognized and is skipped.
- In the DOM APIs, comments in XML documents are not recognized and are skipped.