inDev
SOFTWARE

SAXLib

XML parsing library for Palm OS

Reference Manual

Contents

Introduction

SAXLib is an easy to use freeware XML parsing library for Palm OS. Its API is based on the SAX ("Simple API for XML"), an event-driven interface in which the parser invokes one of several methods supplied by the caller when a "parsing event" occurs. "Events" include recognizing an XML tag, finding an error, encountering a reference to an external entity, or processing a DTD specification.

SAX interface was initially developed for Java programming language, but today you can find its implementation almost for any programming language on many computing platforms. SAXLib is an attempt to bring SAX interface to Palm OS platform.

SAXLib partially implements SAX 2 interface, as specified at SAX project homepage http://www.saxproject.org. SAXLib is based on C programming language (and SAX specification is created for Java), so not every aspect could be implemented exactly (for example, there are no "interfaces" in C language). For such cases other C-specific constructions were used keeping the whole ideology as close to original specification as possible.

This manual is divided to three parts. The first part, *Data structures*, describes all data types used with SAX Lib. The next part, *Application-defined functions*, describes callback functions which SAXLib uses to communicate with your application. It includes two major "interfaces": Content Handler and File Provider. The last part, *Interface functions*, is a reference to SAXLib functions your application calls to parse XML data. It includes functions which initialize library, start and resume parsing and two "interfaces": Document Locator and Attribute.

Data structures

CharacterEncoding

Definition:

```
typedef enum
{
   encodingCP1250=0,       /* charEncodingCP1250 */
   encodingCP1251,         /* charEncodingCP1251 */
   encodingCP1252,         /* charEncodingCP1252 */
   encodingCP1253,         /* charEncodingCP1253 */
   encodingCP1254,         /* charEncodingCP1254 */
   encodingCP1255,         /* charEncodingCP1255 */
   encodingCP1256,         /* charEncodingCP1256 */
   encodingCP1257,         /* charEncodingCP1257 */
   encodingISO8859_1,      /* charEncodingISO8859_1 */
   encodingISO8859_2,      /* charEncodingISO8859_2 */
   encodingISO8859_3,      /* charEncodingISO8859_3 */
   encodingISO8859_4,      /* charEncodingISO8859_4 */
   encodingISO8859_5,      /* charEncodingISO8859_5 */
   encodingISO8859_6,      /* charEncodingISO8859_6 */
   encodingISO8859_7,      /* charEncodingISO8859_7 */
   encodingISO8859_8,      /* charEncodingISO8859_8 */
   encodingISO8859_9       /* charEncodingISO8859_9 */
} CharacterEncoding;
```

Lists all character encodings supported by SAXLib by default. `CharacterEncoding` type is used as an argument in `SAXLibParse` function and defines the character encoding SAXLib should convert to the Unicode characters in parsed XML data.

ContentHandler

Definition:

```
typedef struct ContentHandler
{
   CH_StartDocumentFunc          *StartDocument;
   CH_EndDocumentFunc            *EndDocument;
   CH_StartElementFunc           *StartElement;
   CH_EndElementFunc             *EndElement;
   CH_StartPrefixMappingFunc     *StartPrefixMapping;
   CH_EndPrefixMappingFunc       *EndPrefixMapping;
   CH_CharactersFunc             *Characters;
   CH_IgnorableWhitespaceFunc    *IgnorableWhitespace;
   CH_SkippedEntityFunc          *SkippedEntity;
   CH_ProcessingInstructionFunc  *ProcessingInstruction;
   CH_SetDocumentLocatorFunc     *SetDocumentLocator;
} ContentHandler;
```

`ContentHandler` is a structure of pointers to all function from Content Handler interface. This type is used to set Content Handler interface with `SAXLibSetContentHandler` function.

FileProvider

Definition:

```
typedef struct FileProvider
{
   FP_OpenFileFunc              *OpenFile;
   FP_GetFileBlockSizeFunc      *GetFileBlockSize;
   FP_ReadFileFunc              *ReadFile;
   FP_CloseFileFunc             *CloseFile;
   FP_SeekFileFunc              *SeekFile;
   FP_GetFileSizeFunc           *GetFileSize;
} FileProvider;
```

`FileProvider` is a structure of pointers to all functions from File Provider interface. This type is used to set File Provider interface with `SAXLinSetFileProvider` function.

Application-defined functions

Interfaces

All callback function headers which SAXLib uses to communicate with an application are the part of some interface. The interface is a set of functions grouped by the purpose they are used for.

There are two interfaces defined by SAXLib: Content Handler and File Provider.

Content Handler interface

Application that uses SAXLib functionality may supply to the library pointers for functions from Content Handler interface (using `SAXLibSetContentHandler` function). Only the functions required by the application can be implemented. During the XML content parsing SAXLib calls application's functions from Content Handler interface to report all possible events which occur during processing (for example, start of the document, element start, character data etc.).

CH_StartDocumentFunc

Prototype:

```
typedef void CH_StartDocumentFunc();
```

Function called when the document parsing starts. Does not transfer any data and can be used only for initialization on the application side.

CH_EndDocumentFunc

Prototype:

```
typedef void CH_EndDocumentFunc();
```

Function called when document parsing ends. It can be used for freeing application memory after the end of processing.

CH_StartElementFunc

Prototype:

```
typedef void CH_StartElementFunc(Char* uri, Char* localName, Char*
qName,UInt32 attrID);
```

Function called when SAXLib encounters XML element start (start-tag or an empty element tag) during parsing. This function is called for both standard elements (with element start and element end pair) and empty elements (element start only).

Parameters:

- `(in) uri` Pointer to the null-terminated character string with element's namespace URI or the empty string (pointer to the null character) if element has no namespace URI.
- `(in) localName` Pointer to the null-terminated character string with element's local name (without prefix).
- `(in) qName` Pointer to the null-terminated character string with element's qualified name (with prefix). This argument always has non-zero length.
- `(in) attrID` Attribute list selector which can be used in calling SAXLib for browsing attribute list of this element (see Attribute interface for details).

CH_EndElementFunc

Prototype:

```
typedef void CH_EndElementFunc(Char* uri, Char* localName, Char* qName);
```

Function called when SAXLib encounters XML element end (end-tag) during parsing.

Parameters:

- `(in) uri` Pointer to the null-terminated character string with element's namespace URI or the empty string (pointer to the null character) if element has no namespace URI.
- `(in) localName` Pointer to the null-terminated character string with element's local name (without prefix).
- `(in) qName` Pointer to the null-terminated character string with element's qualified name (with prefix). This argument always has non-zero length.

CH_StartPrefixMappingFunc

Prototype:

```
typedef void CH_StartPrefixMappingFunc(Char* prefix, Char* uri);
```

Not implemented, reserved for the future use.

CH_EndPrefixMappingFunc

Prototype:

```
typedef void CH_EndPrefixMappingFunc(Char* uri);
```

Not implemented, reserved for the future use.

CH_CharactersFunc

Prototype:

```
typedef Boolean CH_CharactersFunc(Char* ch, Int16 start, Int16 length,
UInt32 currentState);
```

Function called each time SAXLib encounters any character data during parsing.

Parameters:

- `(in) ch` Pointer to the character buffer. Do not assume it to be null-terminated, but use `start` and `length` arguments to find out characters position in the buffer.
- `(in) start` Offset of the first reported character in the buffer. Its pointer will be `ch + start`. The number of characters is defined by the `length` parameter.
- `(in) length` Number of the characters reported in the buffer pointed by first three parameters.
- `(in) currentState` Current parsing state selector. If application interrupts parsing (returning false from the function) it can acquire serialized parsing status by calling `SAXLibGetCurrentState` function with `currentState` as a parameter. Afterwards application can resume parsing from the interruption point.

Result: returns `true` to continue parsing or `false` to interrupt it.

CH_IgnorableWhitespaceFunc

Prototype:

```
typedef void CH_IgnorableWhitespaceFunc(Char* ch, Int16 start, Int16
length);
```

Not implemented, reserved for the future use.

CH_SkippedEntityFunc

Prototype:

```
typedef void CH_SkippedEntityFunc(Char* name);
```

Not implemented, reserved for the future use.

CH_ProcessingInstructionFunc

Prototype:

```
typedef void CH_ProcessingInstructionFunc(Char* target, Char* data);
```

Not implemented, reserved for the future use.

CH_SetDocumentLocatorFunc

Prototype:

```
typedef void CH_SetDocumentLocatorFunc(UInt32 locatorID);
```

Function is called before parsing starts (even before `CH_StartDocumentFunc`). It provides the application with document locator selector.

Parameters:

- `(in) locatorID` Document locator selector your application should save for later access to document locator functions.

File Provider interface

Before calling SAXLib for XML data parsing application must provide the library with File Provider interface. SAXLib calls functions from this interface to get the data to parse. By implementing this interface application decides what kind of data SAXLib will parse (from file, Palm database, XML data in dynamic memory etc.).

FP_OpenFileFunc

Prototype:

```
typedef Boolean FP_OpenFileFunc(Char* uri, UInt32 *pFileRef);
```

Function called just before SAXLib starts parsing. This allows application to prepare XML data source.

Parameters:

- `(in) uri` Pointer to the null-terminated string passed to `SAXLibParse` function. This string is application-specific identifier of the data source. For example, it can be a file name if XML data will be read from the file.
- `(out) pFileRef` Pointer to the opened reference value for internal File Provider interface purposes. Your function should update `UInt32` value this parameter points to. Then all other functions in File Provide interface will receive this value as one of the parameters. For our example with XML file source, `pFileRef` can be used for saving opened file reference.

Result: returns `true` if file provider initialization was successful and `false` if not.

FP_GetFileBlockSizeFunc

Prototype:

```
typedef UInt16 FP_GetFileBlockSizeFunc(UInt32 fileRef);
```

Function called to get data source block size. This function is called by SAXLib before request for the first data portion to identify data buffer size. Please note that block size cannot change during parsing.

Parameters:

- `(in) fileRef` Reference to the opened data source returned by `FP_OpenFileFunc` function. Used for data source identification within File Provider interface implementation.

Result: returns single data block size in bytes.

FP_ReadFileFunc

Prototype:

```
typedef Boolean FP_ReadFileFunc(UInt32 fileRef, void *pBuf, UInt16
*readBytes);
```

Function called to read a single data block from the current position in the data source. The function should read not more bytes than the block size returned by `FP_GetFileBlockSizeFunc` function.

Parameters:

- `(in) fileRef` Reference to the opened data source returned by `FP_OpenFileFunc` function. Used for data source identification within File Provider interface implementation.
- `(out) pBuf` Pointer to the data buffer to read XML data to. The buffer is already allocated and your function should only read the data into it.
- `(in/out) readBytes` Pointer to the number of bytes your function should read to the buffer. If it was not possible to read the requested number of bytes (for example, the remaining data is shorter then the block size) then the function should update this value to the number of bytes actually read.

Result: returns `true` if at least one byte was read successively or `false` if not. Please make sure that you update `readBytes` with 0 when returning `false`.

FP_CloseFileFunc

Prototype:

```
typedef void FP_CloseFileFunc(UInt32 fileRef);
```

Function called after SAXLib finishes parsing to allow calling application to free memory and release resources used by File Provider interface.

Parameters:

- `(in) fileRef` Reference to the opened data source returned by `FP_OpenFileFunc` function. Used for data source identification within File Provider interface implementation.

FP_SeekFileFunc

Prototype:

```
typedef Boolean FP_SeekFileFunc(UInt32 fileRef, Int32 *offset);
```

Function called when SAXLib needs to change File Provider current source data position.

Parameters:

- `(in) fileRef` Reference to the opened data source returned by `FP_OpenFileFunc` function. Used for data source identification within File Provider interface implementation.
- `(in/out) offset` Pointer to the new data source offset to be the current position. If offset is not divisible by block size (returned by `FP_GetFileBlockSizeFunc` function) then you can set the current position to the nearest offset less or equal to the transferred value, but divisible by block size. Afterwards you have to update `offset` parameter with the new value. For example, the following formula can be used: `*offset = *offset / blockSize * blockSize`.

Result: returns `true` if the change was successful and `false` if not.

FP_GetFileSizeFunc

Prototype:

```
typedef UInt32 FP_GetFileSizeFunc(UInt32 fileRef);
```

Function returns the size in bytes of whole data which parser needs to process.

Parameters:

- `(in) fileRef` Reference to the opened data source returned by `FP_OpenFileFunc` function. Used for data source identification within File Provider interface implementation.

Result: the size of data in bytes.

Interface functions

Standard library functions

These are the standard functions which exist in any shared Palm library.

SAXLibOpen

Prototype:

```
Err SAXLibOpen(UInt16 refNum, UInt32 *pClientContext, UInt16 *pLibVersion)
```

This is the standard shared library open function. Call this function first, before any others.

Parameters:

- `(in) refNum` Library reference number. You can get this number by calling `SysLibFind` or `SysLibLoad` functions.
- `(out) pClientContext` Pointer to a variable to save client context to. Use the value returned for all subsequent library function calls.
- `(out) pLibVersion` Pointer to a variable to save library version. The version has a format `0xABBC`, where A is a major version number, BB is a minor version number and C is a bug fix number.

Result: returns `errNone` if there were no errors opening the library. Other possible error code is `memErrNotEnoughSpace`, when there was not enough free memory space to init the library structures.

SAXLibClose

Prototype:

```
Err SAXLibClose(UInt16 refNum, UInt32 clientContext)
```

This is the standard shared library close function. Call it when you have finished working with SAXLib library.

Parameters:

- `(in) refNum` Library reference number.
- `(in) clientContext` Client context received from `SAXLibOpen` function.

Result: returns `errNone` if there were no errors. Other possible error code is `SAXLibErrStillOpen`, when the library was unable to close.

Common functions

These are the common functions used to prepare library for parsing an XML file, start and resume parsing.

SAXLibSetContentHandler

Prototype:

```
void SAXLibSetContentHandler(UInt16 refNum,UInt32 clientContext,
ContentHandler *handler)
```

Use this function to provide SAXLib with pointers to Content Handler interface functions. Call this function before starting parsing. If Content Handler interface is not set you can still launch parsing, but in this case your application gets no information about XML file content.

Parameters:

- `(in) refNum` Library reference number.
- `(in) clientContext` Client context received from `SAXLibOpen` function.

- `(in) handler` Pointer to `ContentHandler` structure with pointers to Content Handler interface functions. If you do not want to receive certain events listed in Content Handler interface you need to set pointer to this function to `NULL` in `ContentHandler` structure. Pointers are copied from the location provided to SAXLib internal structures so you do not have to maintain `ContentHandler` structure after calling this function.

SAXLibSetFileProvider

Prototype:

```
void SAXLibSetFileProvider(UInt16 refNum, UInt32 clientContext,
FileProvider *provider)
```

Use this function to provide SAXLib with pointers to File Provider interface. Full implementation of this interface is required on your application side to parse XML files. Without this interface setting SAXLib will be unable to read XML file data.

Parameters:

- `(in) refNum` Library reference number.
- `(in) clientContext` Client context received from `SAXLibOpen` function.
- `(in) provider` Pointer to `FileProvider` structure with pointers to File Provider interface functions. You have to provide pointers to all functions from File Provider interface. Pointers are copied from the location provided to SAXLib internal structures so you do not have to maintain `FileProvider` structure after calling this function.

SAXLibParse

Prototype:

```
Err SAXLibParse(UInt16 refNum, UInt32 clientContext, Char* uri,
CharacterEncoding dstEncoding)
```

This function starts XML data parsing. Be sure you call it after setting Content Handler and File Provider interfaces with `SAXLibSetContentHandler` and `SAXLibSetFileProvider` functions.

Parameters:

- `(in) refNum` Library reference number.
- `(in) clientContext` Client context received from `SAXLibOpen` function.
- `(in) uri` Pointer to null-terminated character string that is a unique identifier of XML document. This string is used only to call File Provider interface functions to identify the document, so this string must be understandable to your File Provider implementation. There are no other requirements to this string.
- `(in) dstEncoding` If XML file contains Unicode characters, SAXLib maps them to encoding specified by this parameter. You can use any encoding from `CharacterEncoding` data type, but the most useful will be to specify codepage used currently by the Palm handheld your application is running on.

Result: returns `errNone` if there were no errors and parsing was finished successfully. Other possible error codes are:

- `SAXLibErrNoFileProvider`, when File Provider was not set before starting the parsing.

- SAXLibErrFileProviderError, when an error was returned by File Provider interface function.
- SAXLibErrEncodingNotSupported, when XML document encoding is not supported.
- SAXLibErrInvalidFile, when SAXLib was unable to parse XML file data.

SAXLibGetCurrentState

Prototype:

```
Err SAXLibGetCurrentState(UInt16 refNum, UInt32 currentState, MemHandle
*phState)
```

Call this function from CH_CharactersFunc function from Content Handler interface if you want to interrupt XML data parsing (by returning false from CH_CharactersFunc), but have to save current parsing state to resume parsing later.

Parameters:

- (in) refNum — Library reference number.
- (in) currentState — Current parsing state selector that your CH_CharactersFunc function receives as the 4th parameter.
- (in/out) phState — Pointer to the memory handle, that SAXLib will allocate to save current parsing state. It is a responsibility of your application to free this memory handle after using it.

Result: returns errNone if there were no errors. Other possible error code is SAXLibErrNotEnoughMemory, when there was not enough dynamic memory to save current parsing state.

SAXLibParseResume

Prototype:

```
Err SAXLibParseResume(UInt16 refNum, UInt32 clientContext, Char* uri,
CharacterEncoding dstEncoding, MemHandle hState)
```

This function resumes XML data parsing paused from CH_CharactersFunc function from Content Handler interface (by returning false from CH_CharactersFunc). It is similar to SAXLibParse and accepts one more parameter, the current parsing state returned by SAXLibGetCurrentState function.

Parameters:

- (in) refNum — Library reference number.
- (in) clientContext — Client context received from SAXLibOpen function.
- (in) uri — Pointer to null-terminated character string that is a unique identifier of XML document. This string is used only to call File Provider interface functions to identify the document, so this string must be understandable to your File Provider implementation. There are no other requirements to this string.
- (in) dstEncoding — If XML file contains Unicode characters, SAXLib maps them to encoding specified by this parameter. You can use any encoding from CharacterEncoding data type, but the most useful will be to specify codepage used currently by the Palm handheld your application is running on.
- (in) hState — Handle of the memory chunk that contains parsing state to resume from. This memory handle is allocated by SAXLibGetCurrentState function. Please note, that

SAXLibParseResume does not free this memory handle and your application is responsible for it.

Result: returns errNone if there were no errors and parsing was finished successfully. Other possible error codes are:

- SAXLibErrParam, when memory handle specified by hState parameter contains invalid parsing status data.
- SAXLibErrNoFileProvider, when File Provider was not set before starting the parsing.
- SAXLibErrFileProviderError, when an error was returned by File Provider interface function.
- SAXLibErrEncodingNotSupported, when XML document encoding is not supported.
- SAXLibErrInvalidFile, when SAXLib was unable to parse XML file data.

Document Locator interface functions

This group of functions can be used to identify current parsing position in XML data. All these functions receive as one of the parameters document locator identifier. You can obtain this identifier during XML data parsing by implementing CH_SetDocumentLocatorFunc function from Content Handler interface.

SAXLib_DL_GetSystemId

Prototype:

```
Char* SAXLib_DL_GetSystemId(UInt16 refNum, UInt32 locatorID)
```

This function returns a pointer to the XML document identifier that your application supplies to SAXLibParse and SAXLibParseResume functions (uri parameter).

Parameters:

- (in) refNum — Library reference number.
- (in) locatorID — Document locator interface ID provided by SAXLib during parsing to CH_SetDocumentLocatorFunc function from Content Handler interface.

Result: returns pointer to the null-terminated character string with XML document identifier or NULL in the case of any error. Please note, that SAXLib_DL_GetSystemId function does not allocate memory for this string and only returns pointer provided to SAXLibParse and SAXLibParseResume functions.

SAXLib_DL_GetPublicId

Prototype:

```
Char* SAXLib_DL_GetPublicId(UInt16 refNum, UInt32 locatorID)
```

Not used in the current library version, returns NULL.

SAXLib_DL_GetColumnNumber

Prototype:

```
UInt16 SAXLib_DL_GetColumnNumber(UInt16 refNum, UInt32 locatorID)
```

Use this function to get column number of the current parsing position in XML data file. Together with SAXLib_DL_GetLineNumber can be used to identify the current parsing position.

Parameters:

- (in) refNum — Library reference number.

- `(in) locatorID` Document locator interface ID provided by SAXLib during parsing to `CH_SetDocumentLocatorFunc` function from Content Handler interface.

Result: returns column number of the current parsing position or `0xFFFF` in the case of any error.

SAXLib_DL_GetLineNumber

Prototype:

```
UInt16 SAXLib_DL_GetLineNumber(UInt16 refNum, UInt32 locatorID)
```

Use this function to get line number of the current parsing position in XML data file. Together with `SAXLib_DL_GetColumnNumber` can be used to identify the current parsing position.

Parameters:

- `(in) refNum` Library reference number.
- `(in) locatorID` Document locator interface ID provided by SAXLib during parsing to `CH_SetDocumentLocatorFunc` function from Content Handler interface.

Result: returns line number of the current parsing position or `0xFFFF` in the case of any error.

Attribute interface functions

Attribute interface is a group of functions used to access attributes of XML data element. Each time SAXLib finds an XML element it calls user-defined `CH_StartElementFunc` function from Content Handler interface. One of the arguments SAXLib provides to this function is Attribute Identifier (`attrID` parameter). This identifier can be used to call Attribute Interface to get information about attributes of the element.

SAXLib_Attr_GetIndexByQName

Prototype:

```
UInt16 SAXLib_Attr_GetIndexByQName(UInt16 refNum, UInt32 attrID, Char*
qName)
```

Returns index in the attributes list for the attribute with specified qualified name (as appears in the XML parsed data file).

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) qName` Pointer to null-terminated character string with qualified attribute name to look in the list.

Result: returns index of the attribute or `0xFFFF` if attribute was not found or in the case of any error.

SAXLib_Attr_GetIndexByNamespace

Prototype:

```
UInt16 SAXLib_Attr_GetIndexByNamespace(UInt16 refNum, UInt32 attrID, Char*
uri, Char* localName)
```

Returns index in the attributes list for the attribute with specified namespace name (namespace URI and local name).

Parameters:

- `(in) refNum` Library reference number.

- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) uri` Pointer to null-terminated character string with namespace URI for the attribute to search.
- `(in) localName` Pointer to null-terminated character string with local name for the attribute to search.

Result: returns index of the attribute or `0xFFFF` if attribute was not found or in the case of any error.

SAXLib_Attr_GetLength

Prototype:

```
UInt16 SAXLib_Attr_GetLength(UInt16 refNum, UInt32 attrID)
```

Returns the number of attributes (attributes list length) for the currently parsed element.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.

Result: returns number of attributes or `0xFFFF` in the case of any error.

SAXLib_Attr_GetLocalName

Prototype:

```
Char* SAXLib_Attr_GetLocalName(UInt16 refNum, UInt32 attrID, UInt16 index)
```

Returns local name for the attribute with specified index.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) index` Attribute index to get local name. Attribute indexes are zero-based and must be less or equal to the number of attributes in the list (returned by `SAXLib_Attr_GetLength` function).

Result: pointer to null-terminated character string with local name of the attribute. This function allocates memory for the name and it is application's responsibility to free memory chunk returned by this function. In the case of any error the function returns `NULL`.

SAXLib_Attr_GetQName

Prototype:

```
Char* SAXLib_Attr_GetQName(UInt16 refNum, UInt32 attrID, UInt16 index)
```

Returns qualified name of the attribute with specified index.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) index` Attribute index to get qualified name. Attribute indexes are zero-based and must be less or equal to the number of attributes in the list (returned by `SAXLib_Attr_GetLength` function).

Result: pointer to null-terminated character string with qualified name of the attribute. This function allocates memory for the name and it is application's responsibility to free memory chunk returned by this function. In the case of any error the function returns `NULL`.

SAXLib_Attr_GetTypeByIndex

Prototype:

```
Char* SAXLib_Attr_GetTypeByIndex(UInt16 refNum, UInt32 attrID, UInt16
index)
```

Returns type of the attribute (if resolved) with specified index.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) index` Attribute index to get type name. Attribute indexes are zero-based and must be less or equal to the number of attributes in the list (returned by `SAXLib_Attr_GetLength` function).

Result: pointer to null-terminated character string with attribute type name. This function allocates memory for the type name and it is application's responsibility to free memory chunk returned by this function. If attribute type was not resolved the default type (`"CDATA"`) is returned. In the case of any error the function returns `NULL`.

SAXLib_Attr_GetTypeByQName

Prototype:

```
Char* SAXLib_Attr_GetTypeByQName(UInt16 refNum, UInt32 attrID, Char*
qName)
```

Returns type of the attribute (if resolved) with specified qualified name.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) qName` Pointer to null-terminated character string with qualified attribute name to look in the list.

Result: pointer to null-terminated character string with attribute type name. This function allocates memory for the type name and it is application's responsibility to free memory chunk returned by this function. If attribute type was not resolved the default type (`"CDATA"`) is returned. If attribute with specified name was not found or in the case of any error the function returns `NULL`.

SAXLib_Attr_GetTypeByNamespace

Prototype:

```
Char* SAXLib_Attr_GetTypeByNamespace(UInt16 refNum, UInt32 attrID, Char*
uri, Char* localName)
```

Returns type of the attribute (if resolved) with specified namespace name (namespace URI and local name).

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.

- `(in) uri` Pointer to null-terminated character string with namespace URI for the attribute to search.
- `(in) localName` Pointer to null-terminated character string with local name for the attribute to search.

Result: pointer to null-terminated character string with attribute type name. This function allocates memory for the type name and it is application's responsibility to free memory chunk returned by this function. If attribute type was not resolved the default type (`"CDATA"`) is returned. If attribute with specified name was not found or in the case of any error the function returns `NULL`.

SAXLib_Attr_GetUri

Prototype:

```
Char* SAXLib_Attr_GetUri(UInt16 refNum, UInt32 attrID, UInt16 index)
```

Returns namespace URI for the attribute with specified index.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) index` Attribute index to get namespace URI. Attribute indexes are zero-based and must be less or equal to the number of attributes in the list (returned by `SAXLib_Attr_GetLength` function).

Result: pointer to null-terminated character string with namespace URI of the attribute. This function allocates memory for the name and it is application's responsibility to free memory chunk returned by this function. In the case of any error the function returns `NULL`.

SAXLib_Attr_GetValueByIndex

Prototype:

```
Char* SAXLib_Attr_GetValueByIndex(UInt16 refNum, UInt32 attrID, UInt16
index)
```

Returns value of the attribute with specified index.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) index` Attribute index to get value. Attribute indexes are zero-based and must be less or equal to the number of attributes in the list (returned by `SAXLib_Attr_GetLength` function).

Result: pointer to null-terminated character string with value of the attribute. This function allocates memory for the value string and it is application's responsibility to free memory chunk returned by this function. In the case of any error the function returns `NULL`.

SAXLib_Attr_GetValueByQName

Prototype:

```
Char* SAXLib_Attr_GetValueByQName(UInt16 refNum, UInt32 attrID, Char*
qName)
```

Returns value of the attribute with specified qualified name.

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) qName` Pointer to null-terminated character string with qualified attribute name to look in the list.

Result: pointer to null-terminated character string with value of the attribute. This function allocates memory for the value string and it is application's responsibility to free memory chunk returned by this function. If attribute with specified name was not found or in the case of any error the function returns `NULL`.

SAXLib_Attr_GetValueByNamespace

Prototype:

```
Char* SAXLib_Attr_GetValueByNamespace(UInt16 refNum,UInt32 attrID,Char*
uri,Char* localName)
```

Returns value of the attribute with specified namespace name (namespace URI and local name).

Parameters:

- `(in) refNum` Library reference number.
- `(in) attrID` Attribute interface ID provided by SAXLib during parsing to `CH_StartElementFunc` function from Content Handler interface.
- `(in) uri` Pointer to null-terminated character string with namespace URI for the attribute to search.
- `(in) localName` Pointer to null-terminated character string with local name for the attribute to search.

Result: pointer to null-terminated character string with value of the attribute. This function allocates memory for the value string and it is application's responsibility to free memory chunk returned by this function. If attribute with specified name was not found or in the case of any error the function returns `NULL`.