C++, VC6, Windows, VS, MFC, Dev

HTML Reader C++ Class Library

By gUrM33T

Posted : 30 Mar 2004
Updated : 30 Mar 2004
Views : 82,357

A lightweight, fast, simple, and low-overhead C++ class library based on push model parsing.

- Download Library Source (Fully Documented) - 24 Kb

Contents

Introduction

After failing to search for a class library that allows to read HTML text from either in-memory string buffers or physical disk files, I decided that there is a severe need to have a library like that. There are many parsers available for XML (eXtensible Markup Language), for instance, Simple API for XML (SAX), that allow you to parse XML simply by handling events that the reader generates as it parses specific symbols from the given XML document.

Inspired by the SAX parser for XML, I decided to develop an HTML Reader C++ Class Library myself from scratch that offers a simple, lightweight, fast, and the most important, a low-overhead solution to process an HTML document. Like SAX, I decided to develop an events-based parser, which raises events as it encounters various elements in the document. The advantage of an events-based parser is that the reader reads a section of an HTML document, generates an event, and then moves on to the next section. It uses less memory and is better for processing large documents.

Events-Based Parser

An events-based parser uses the callback mechanism to report parsing events. These callbacks turn out to be protected virtual member functions that you will override. Events, such as the detection of an opening tag or the closing tag of an element, will trigger a call to the corresponding member function of your class. The application implements and registers an event handler with the reader. It is upto the application to put some code in the event handlers designed to achieve the objective of the application. Events-based parsers provide a simple, fast, and a lower-level access to the document being parsed.

Events-based parsers do not create an in-memory representation of the source document.

They simply parse the document and notify client applications about various elements they find along the way. What happens next is the responsibility of the client application. Events-based parsers don't cache information and have an enviably small memory footprint.

Files

To use the HTML Reader Class Library in your MFC application project, you will need to add a number of files in your project:

Header File	Source File	Class
LiteHTMLReader.h	LiteHTMLReader.cpp	`CLiteHTMLReader`
LiteHTMLTag.h	-	`CLiteHTMLTag`
LiteHTMLAttributes.h	-	`CLiteHTMLAttributes`
LiteHTMLAttributes.h	LiteHTMLAttributes.cpp	`CLiteHTMLElemAttr`
LiteHTMLEntityResolver.h	LiteHTMLEntityResolver.cpp	`CLiteHTMLEntityResolver`

NOTE: `LiteHTMLCommon.h` must also be included in your project.

Brief Description of Classes

- `CLiteHTMLReader` is the main class of our library that works in conjunction with other `CLiteHTML*` classes to parse the given HTML document. It contains methods (`Read` and `ReadFile`) to initiate the parsing process that can operate on either in-memory string buffers or a physical disk file. `CLiteHTMLReader` allows you to trap events that the reader generates as it finds various elements in the document such as the starting of a tag, ending of a tag, an HTML comment, etc. But to handles these events, your application must define a class that implements an interface `ILiteHTMLReaderEvents` declared in the `LiteHTMLReader.h` file.

- `CLiteHTMLTag` class, as its name implies, is related to the HTML tags. It deals with the parsing and storage of tag information from the given string such as the name of the tag and the attributes/properties of a tag. It provides a method (actually, all the above-specified classes provide a method named `parseFromStr`) that is called by the `CLiteHTMLReader` class' `Read` and `ReadFile` method as the document is being parsed. Typically, `CLiteHTMLTag` is not used directly by your application. As specified above, it works in conjunction with the reader helping in the parsing of HTML tags.

- The `CLiteHTMLElemAttr` and `CLiteHTMLAttributes` classes are inter-related as `CLiteHTMLAttributes` provides a collection-based mechanism to hold an array of `CLiteHTMLElemAttr` objects that are accessible either by the name of the attribute or a zero-based index value. As was the case with the `CLiteHTMLTag` class, these classes are also not typically used by your application directly.

- The last is the `CLiteHTMLEntityResolver` class that helps in resolving the entity references. Entity references are numeric or symbolic names for characters that may be included in an HTML document. They are useful for referring to rarely used characters, or those that authoring tools make it difficult or impossible to enter. Entity references begin with a "&" sign and end with a semi-colon (;). Some common examples are: `<` representing the < sign, `>` representing the > sign, etc.

From the above discussion, one thing is clear that the `CLiteHTMLTag`, `CLiteHTMLAttributes`, and, `CLiteHTMLElemAttr` class provide a method named `parseFromStr` that is used by the `CLiteHTMLReader` to further delegate the parsing process

while reading an HTML document.

Usage

OK, now let's come to the part of learning how to use this library in an MFC project:

1. The **first** step is pretty simple. All you have to do is to add all of the files (given in the FILES section above) in your project.

2. The **second** step, although optional, is to create a class that implements `ILiteHTMLReaderEvents` interface. `ILiteHTMLReaderEvents` is, in actual, an abstract class that acts as an interface which must be implemented by all those classes that need to handle events raised by the `CLiteHTMLReader` class. For example,

```
#include "stdafx.h"

#include "LiteHTMLReader.h"


class CEventHandler : public ILiteHTMLReaderEvents
{
private:
    void BeginParse(DWORD dwAppData, bool &bAbort);
    void StartTag(CLiteHTMLTag *pTag, DWORD dwAppData, bool &bAbort);
    void EndTag(CLiteHTMLTag *pTag, DWORD dwAppData, bool &bAbort);
    void Characters(const CString &rText, DWORD dwAppData, bool &bAbort);
    void Comment(const CString &rComment, DWORD dwAppData, bool &bAbort);
    void EndParse(DWORD dwAppData, bool bIsAborted);
};
```

   You must have noticed that "optional" word I used above. The reason behind this is that if you do not provide your own implementation of the event handler(s), the `ILiteHTMLReaderEvents` class provides a default implementation that does nothing. To learn more about the `ILiteHTMLReaderEvents` interface, jump to the ILiteHTMLReaderEvents Described section of this article.

3. The **third** step is to create an instance of the `CLiteHTMLReader` class like this:

```
    CLiteHTMLReader theReader;
```

4. Now we should call either `Read` or `ReadFile` method of the `CLiteHTMLReader` class???

   NO! Our event handler implementation will not start receiving notifications until we register it with the reader by calling the `setEventHandler` method of the `CLiteHTMLReader` class. So, supposing that the name of our class that is implementing the `ILiteHTMLReaderEvents` interface is `CEventHandler`, the **fourth** step is to create an instance of the `CEventHandler`, and call `setEventHandler` by passing it the address of this instance variable.

```
    CEventHandler theEventHandler;
    theReader.setEventHandler(&theEventHandler);
```

   Now, for all of you, who are thinking if it is possible to pass a `NULL` pointer to the `setEventHandler` method, the answer is YES and that too at any time you want. And not to mention, you can also change the event handler at any time by calling `setEventHandler` and passing the address of some other instance.

5. Now, the **fifth** and the final step is to call either `Read` or `ReadFile` method on the `CLiteHTMLReader` instance variable we created in step 3 by passing it the appropriate parameter i.e. if you decide to parse an in-memory string buffer, call the `Read` method and pass the address of the string you want to parse. In case, you need to parse an HTML document from a disk file, you can call another method `ReadFile` that is similar to `Read` but accepts a file handle (`HANDLE`) instead of a pointer to an array of characters. Take a look at the example:

```
TCHAR   strToParse[] = _T("<HTML>"
     "<HEAD>"
     "<TITLE>"
     "<!-- title goes here -->"
     "</TITLE>"
     "</HEAD>"
     "<BODY LEFTMARGIN="15px">This is a sample HTML document.</BODY>"
     "</HTML>");
theReader.Read(strToParse);
```

OR

```
CFile   fileToParse;
if (fileToParse.Open(_T("test.html"), CFile::modeRead))
{
    theReader.ReadFile(fileToParse.m_hFile);
    fileToParse.Close();
}
```

More About Event Handling

The `ILiteHTMLReaderEvents` class presents an interface that must be implemented by all those classes that want to handle the notifications sent by the `CLiteHTMLReader` while parsing an HTML document. The order of events handled by the `ILiteHTMLReaderEvents` handler is determined by the order of information within the document being parsed. It's important to note that the interface includes a series of methods that the `CLiteHTMLReader` invokes during the parsing operation. The reader passes the appropriate information to the method's parameters. To perform some type of processing for a method, you simply add code to the method in your own `ILiteHTMLReaderEvents` implementation.

The common parameters received by all of the methods defined in `ILiteHTMLReaderEvents` class, except `EndParse` include:

- `dwAppData`: A 32-bit application-specific data.
- `bAbort`: You can set this parameter to either `true` or `false` according to your application's needs to specify whether the reader should continue parsing rest of the data in the buffer or aborts immediately after the current event handler completes processing.

The `EndParse` method receives `bIsAborted` parameter instead of the `bAbort` that signifies if `EndParse` has occured because of the normal parsing termination.

Along with the above-specified parameters, all of the methods except `BeginParse` and `EndParse`, receive some extra information (specific to the event) which is retrieved by the reader while parsing the HTML document. For instance, when an HTML tag (either opening or closing) is parsed, the `StartTag` or `EndTag` methods receive a pointer to a `CLiteHTMLTag` that contains the name of the tag and the attributes (if any) of the tag. Attribute information is retrieved only if the tag parsed is an opening tag as closing tags cannot contain any

attribute/value pairs. If there is no attribute information associated with a CLiteHTMLTag, the pointer variable contains NULL. So it is obvious that EndTag method always receives a NULL pointer. It is the responsibility of an application (and a good programming practice) to check for NULL pointer before using it.

Similarly, the Comment and Characters method of the class receives a reference to a CString containing the extracted text. The Comment method receives rComment parameter containing the comment text excluding the delimeters i.e. without <!-- and -->. The Characters method receives a rText parameter that signifies either the contents of an element or some text that could not be parsed by the reader.

Class View

CLiteHTMLReader Class Members

Member	Description
`CLiteHTMLReader()`	Constructs a `CLiteHTMLReader` object.
`EventMaskEnum setEventMask(DWORD);`	Sets a new event mask.
`EventMaskEnum setEventMask(DWORD, DWORD);`	Changes the current event mask by adding and/or removing flags.
`EventMaskEnum getEventMask(void) const;`	Returns the event mask previously set by a call to `setEventMask`.
`DWORD setAppData(DWORD);`	Sets application-specific data to be passed to event handlers.
`DWORD getAppData(void) const;`	Returns app-specific data previously set by a call to `setAppData`.
`ILiteHTMLReaderEvents* setEventHandler(ILiteHTMLReaderEvents*);`	Registers an event handler with the reader.
`ILiteHTMLReaderEvents* getEventHandler(void) const;`	Returns the currently associated event handler.
`UINT Read(LPCTSTR);`	Parses an HTML document from the specified string.
`UINT Read(HANDLE);`	Parses an HTML document from a file given its HANDLE.

CLiteHTMLTag Class Members

Member	Description
`CLiteHTMLTag()`	Constructs a `CLiteHTMLTag` object.
`CLiteHTMLTag(CLiteHTMLTag&, bool)`	Constructs a `CLiteHTMLTag` object from an existing instance. The first parameter is the reference to a source `CLiteHTMLTag`, and the second

	parameter determines whether to make a copy or to take ownership of the encapsulated CLiteHTMLAttributes pointer.
~CLiteHTMLTag()	Destroys a CLiteHTMLTag object.
CString getTagName(void) const;	Returns the name of the tag.
const CLiteHTMLAttributes* getAttributes(void) const;	Returns a pointer to an attribute collection associated with this CLiteHTMLTag.
UINT parseFromStr(LPCTSTR, bool&, bool&, bool);	Parses an HTML tag from the string specified by the first parameter. The second and third parameter receive a boolean true/false indicating that the tag parsed is an opening and/or closing tag, respectively. The fourth parameter specifies whether to parse tag's attributes also.

CLiteHTMLAttributes Class Members

Member	Description
CLiteHTMLAttributes()	Constructs a CLiteHTMLAttributes object.
CLiteHTMLAttributes(CLiteHTMLAttributes&, bool)	Constructs a CLiteHTMLAttributes object from an existing instance. The first parameter is the reference to a source CLiteHTMLAttributes, and the second parameter determines whether to make a copy or to take ownership of the encapsulated pointer.
~CLiteHTMLAttributes()	Destroys a CLiteHTMLAttributes object.
int getCount() const;	Returns the count of CLiteHTMLElemAttr items.
int getIndexFromName(LPCTSTR) const;	Looks up the index of an attribute given its name.
CLiteHTMLElemAttr operator[](int) const;	Returns a CLiteHTMLElemAttr object given an attribute's index.
CLiteHTMLElemAttr getAttribute(int) const;	Returns a CLiteHTMLElemAttr object given an attribute's index.

`CLiteHTMLElemAttr operator[](LPCTSTR) const;`	Returns a `CLiteHTMLElemAttr` object given an attribute name.
`CLiteHTMLElemAttr getAttribute(LPCTSTR) const;`	Returns a `CLiteHTMLElemAttr` object given an attribute name.
`CString getName(int) const;`	Returns the name of an attribute given its index.
`CString getValue(int) const;`	Returns the value of an attribute given its index.
`CString getValueFromName(LPCTSTR) const;`	Returns the value of an attribute given its name.
`CLiteHTMLElemAttr* addAttribute(LPCTSTR, LPCTSTR);`	Adds a new `CLiteHTMLElemAttr` item to the collection.
`bool removeAttribute(int);`	Removes an `CLiteHTMLElemAttr` item from the collection.
`bool removeAll(void);`	Removes all `CLiteHTMLElemAttr` items from the collection.
`UINT parseFromStr(LPCTSTR);`	Parses attribute/value pairs from the given string.

CLiteHTMLElemAttr Class Members

Member	Description
`CString getName(void) const;`	Returns the name of an `CLiteHTMLElemAttr`.
`CString getValue(void) const;`	Returns the value of an `CLiteHTMLElemAttr`.
`bool isColorValue(void) const;`	Determines if the attribute value contains a color reference.
`bool isNamedColorValue(void) const;`	Determines if the attribute value is a named color value.
`bool isSysColorValue(void) const;`	Determines if the attribute value is a named system color value.
`bool isHexColorValue(void) const;`	Determines if the attribute value is a color value in hexadecimal format.
`bool isPercentValue(void) const;`	Checks to see if the attribute contains a percent value.
`COLORREF getColorValue(void) const;`	Returns the color value of the attribute.
`CString getColorHexValue(void) const;`	Returns the RGB value of the attribute in hexadecimal format.
`unsigned short getPercentValue() const;`	Returns a percent value of the attribute.

`short getLengthValue(LengthUnitsEnum&) const;`	Returns a length value of the attribute.
`operator bool() const;`	Converts attribute value to `bool`.
`operator BYTE() const;`	Converts attribute value to `BYTE` (`unsigned char`).
`operator double() const;`	Converts attribute value to `double`.
`operator short() const;`	Converts attribute value to `signed short int`.
`operator LPCTSTR() const;`	Returns the value of the attribute.
`UINT parseFromStr(LPCTSTR);`	Parses an attribute/value pair from the given string.

License

About the Author

gUrM33T

Occupation: Web Developer
Location: Canada

Discussions and Feedback

62 messages have been posted for this article. Visit **http://www.codeproject.com/KB/library/GomzyHTMLReader.aspx** to post and view comments on this article, or click **here** to get a print view with messages.