Creating a custom XML parser

for kbmMW v. 2.00+

kbmMW is bundled with a transport stream format which is able to understand XML. For this purpose a special high performance XML parser has been designed from ground up and utilized in the transport.

Basing the custom XML parser on the TkbmMWCustomXMLParser, you will be given tokens which can contain XML tags, endtags, attributes and namespace information.

The XML parser supports reading XML documents with tags and symbols. The tags can include attributes and namespace names, be closing or end tags.

A couple of examples of tags:

Tag with symbol and closing tag:

```
<SOMETAG>
somesymbol
</SOMETAG>
```

Ending tag:

```
<SOMETAG/>
```

Tag with attributes, multiline symbol and closing tag:

```
<SOMETAG name='xyz' value='aaa'>
somesymbol
somesymbol more
</SOMETAG>
```

Ending tag with attributes:

```
<SOMETAG name='xyz' value='aaa' someattrib='somevalue'/>
```

Ending tag with namespace:

```
<SOMENAMESPACE:SOMETAG/>
```

Remember that you should declare the namespace in your document opening tag. (The kbmMW XML parser doesnt care if you do, but the XML standard require it). Eg.

```
<YOURDOC xmlns='mynamespace:someURLofyourown'>
  <mynamespace:TAG1>
  ... somesymbol...
  </mynamespace:TAG1>
</YOURDOC>
```

mynamespace can be any word/name you would like.

someURLofyourown should be any globally unique string. If you have a website registered with your own domain name, its usually included, as that is a kind of guarantee that the value is unique. Further its common to add additional path which indicates the version of the namespace which your document is using etc. Again the URL is purely seen as a globally unique string value, nothing else.

Eg.

```
<YOURDOC xmlns='X:http://www.yourdomainname.com/1.00/.../'>
```

A simple example of an XML document reader which reads a document of the following kind containing two values as attributes:

```
<YOURXML>
  <DATA name='MyValue1' Value='1'/>
  <DATA name='MyValue2' Value='2'/>
</YOURXML>
```

```
TYourXMLDocument = class
protected
    FMyValue1:integer;
    FMyValue2:integer;
public
    procedure Parse(XMLStream:TMemoryStream);
end;

procedure TYourXMLDocument.Parse(XMLStream:TMemoryStream);
var
   reading:boolean;
begin
   // Setup default values.
   FMyValue1:=-1;
   FMyValue2:=-1;

   with TkbmMWCustomXMLParser.Create(XMLStream) do
      try
         // Look for next token. Dont look for a symbol.
         NextToken(false);

         // If tokentype is mwxml_tEnd, the end of the XML document has been reached.
         reading:=false;
         while TokenType<>mwxml_tEnd do
         begin

            // Check what type of token we've got.
            case TokenType of

               mwxml_tXMLTag:
                  // Check if got start or end of your document tag.
                  if TokenXMLTagIs('YOURXML') then
                  begin
                       // The reading flag will be set to true when the start tag of
                       // your document has been read, provided it is'nt also an endtag
                       reading:=not (IsEndTag or IsClosingTag);
                  end

                  // Else if reading values within your document
                  // (everything between <YOURXML> and </YOURXML>)
```

```
                else if reading then
                begin
                     if TokenXMLTagIs('DATA') then
                     begin

                          // Check the attributes.
                          if AttribValue['NAME']='MyValue1' then
                             FMyValue1:=strtoint(AttribValue['VALUE'])
                          else if AttribValue['NAME']='MyValue2' then
                             FMyValue2:=strtoint(AttribValue['VALUE']);
                     end;
                end;

                // Ignore other token types.
            end;
        end;
    finally
        Free;
    end;
end;
```

To read your XML document, create an instance of your TYourXMLDocument and call its Parse procedure giving a memory stream containing the XML document as argument.

It may look a bit complicated to read two values from an XML document, but the beauty is that its very fast and flexible. Values can be there or not, and the order in which they appear is not important.

The shown sample utilize several methods and properties of the TkbmMWCustomXMLParser class. But even more are available.

Constructors

constructor Create(const ABuffer:PChar)

Prepares for parsing an XML document from a zero terminated buffer.

constructor Create(const AStream:TCustomMemoryStream)

Prepares for parsing an XML document from a memory stream.

Navigators

procedure NextToken(const ASymbol:boolean)

Scans the buffer for next XML token/symbol.. The ASymbol argument determines if it should scan for a symbol or an XML token.

function TokenXMLTagIs(s:string):boolean

Returns true of the current token is named s. If a namespace is given as part of the name s, it will automatically be taken into account.

function GetBookmark:PChar

Returns a bookmark which one can return to if to reparse from a specific position in the XML document.

procedure GotoBookmark(const ABookmark:PChar)
Repositions to a specific place in the XML document.

procedure SetPosition(const APos:int64)
Place the cursor to a specific offset into the XML document.

function GetPosition:int64
Gets the current position as an offset from start in the XML document.

Attributevalues

function GetStringAttribValue(aAttrib:string; const ADef:string):string
Return the string value of the attribute with the given name aAttrib. If its not found, return the default string ADef.

function GetIntAttribValue(aAttrib:string; const ADef:integer):integer
Return the integer value of the attribute with the given name aAttrib. If its not found, return the default integer value ADef.

Properties

property TokenName:string
Return the name of the latest token after calling NextToken.

property TokenString:string
Return the complete token string obtained from the latest call to NextToken.

property TokenType:TkbmMWXMLToken
Return the type of token obtained from the latest call to NextToken. It can be any of:

mwxml_tEnd	End of XML document reached.
mwxml_tSymbol	Found a symbol (eg. data value between tags <start>.... symbol... </start>)
mwxml_tXMLTag	Found an XML tag. (eg <start>) with optional attribute values.

property NameSpace:string
Return the namespace of the latest found tag.

property TokenStartOfs:integer
Return the offset of the start of the found token in the XML document from the latest call to NextToken.

property TokenEndOfs:integer
Return the offset of the end of the found token in the XML document from the latest call to NextToken.

property Attribs:TStringList
Contain the list of attributes found for the current tag.

property IndexOfAttrib[aAttrib:string]:integer
Get the index into the Attribs string list of the attribute named aAttrib.

property AttribValue[aAttrib:string]:string
Get the string value of the attribute named aAttrib. If none with that name is found, an empty string is returned.

property IsEndTag:boolean
Returns true if the token from the latest call to NextToken is an end tag. Eg. <SOMETAG/> which is a tag with both and opening and a closing part in the same tag.

property IsClosingTag:boolean
Returns true of the token from the latest call to NextToken is a closing tag. Eg. </SOMETAG> which is the closing tag from the opening tag <SOMETAG>. Eg. <SOMETAG>....</SOMETAG>

This concludes the whitepaper about creating custom XML parsers.

Kim Madsen
Components4Developers