kbmMW's DOM XML capabilities

Some capabilities require kbmMW v. 2.54 or newer.

All kbmMW Editions bundles support for manipulating XML documents very easily.

kbmMW contains two main XML parsing methods, SAX (which is very fast and lean) and DOM (which is based on SAX and thus still very fast, but require the document to reside in memory).

The SAX method is only good for reading and parsing XML files very fast and with very few ressources. However to easily manipulate XML documents, the DOM classes is the easiest and still very fast and efficient way.

The document 'Creating custom XML parsers' explain how to use and extend the SAX parser. This document will concentrate on the DOM classes.

Introduction

What is DOM?

DOM is an acronym for Document Object Model. Essentially it represents a way to reference the contents of an XML document.
When an XML document is read by a DOM parser, a tree of nodes is genereted. Each node represents an element in the XML document.

As it is possible directly to reference a single element in the XML document, it is very easy to extract relevant information from the XML document.

Similarly its possible to build an XML document easily using DOM, as one simply defines the nodes in the tree and saves it to a storage or converts the tree to a string which will then contain the full XML document.

Reading an XML document

The following is an example of how to read an XML document into a DOM instance.

```
uses
    …
    kbmMWXML;

var
   fs:TFileStream;
   dom:TkbmMWDOMXML;
   subnodecount:integer;
   n, xmlnode:TkbmMWDOMXMLNode;
begin
   dom:=TkbmMWDOMXML.Create;
   try
      fs:=TFileStream.Create('Somefile.xml', fmOpenRead);
      try
         dom.LoadFromStream(fs);

         // If the XML document is wellformed, and thus
         // no reading errors occurred, you now have
         // the XML document in memory as a tree of nodes where
         // the root can be accessed via the Root property.
         n:=dom.Root;

         // And subnodes (including the declarative node <?xml..?>)
         // can be access via the Nodes property.
         subnodecount:=n.Nodes.Count;
         if subnodecount>0 then
         begin
              xmlnode:=n.Nodes[0]; // Will get the <?zml..?> node on
                                   // a wellformed document.
         end;

         // Another way to get to the <?xml..?> node is using the
         // path support.
         xmlnode:=dom.GetNode('//xml');

         // Or getting some subnode.
         n:=dom.GetNode('//memo/from');

         …

      finally
         fs.free;
      end;
   finally
      dom.free;
   end;
end;
```

Its also possible to load the XML from a string using the LoadFromString method.

Saving an XML document as a file or stream

The following is an example of how to write the XML DOM tree out as an XML file.

```
uses
    …
    kbmMWXML;

var
   fs:TFileStream;
   dom:TkbmMWDOMXML;
begin
   dom:=TkbmMWDOMXML.Create;

   // Generate some nodes. Could forexample have been
   // generated by reading some XML document in as shown
   // in the previous example.
   …

   // If the XML tree has been manipulated by inserting new nodes or deleting
   // old ones, one should optionally call update to recalculate the
   // internal levels of the nodes in the tree. This is important if the
   // SaveToStream/SaveToString methods should autoindent subnodes.
   dom.Update;

   try
      fs:=TFileStream.Create('Somefile.xml', fmCreate);
      try
         dom.SaveToStream(nil,fs);

         // The first argument specifies that we want
         // to save from the root of the tree or in other
         // words save the complete DOM tree.
         …

      finally
         fs.free;
      end;
   finally
      dom.free;
   end;
end;
```

Its also possible to generate a string from the DOM tree using:the SaveToString method.

How to create a DOM tree frem scratch

The following is an example of how to create the following XML document from scratch:

```
<?xml version="1.0" ?>
<Memo distribution="internal">
    <from>Henry Stuff III</from>
    <to>All employees</to>
    <date>6 April 2001</date>
    <content>It is with great delight that I announce some results of
      ourquarterly cost cutting efforts. Over this past quarter, the
      customerservice segment of our business has managed to reduce
      customerreturns by over 50%. This has netted our business
      incredible savings and shows great ingenuity on thepart of our
      customer service staff. To quickly summarize two of thestrategies
      involved, the various customer service groups decided torelocate
      the service desk at the back of each storeinstead of the front. The
      width of aisles in that section of thestore was reduced, making it
      more difficult for customers to reachthe service desk. The hiring
      of people unable to speak the Englishlanguage also had a positive
      impact on the ability of the customerto return merchandise. A
      hearty congratulations and job well done toour teammates in
      customer service!</content>
</Memo>
```

One way to get to this XML is to do like this:

Declarative bits:

```
uses kbmMWXML;

procedure TForm1.Button1Click(Sender: TObject);
var
   fs:TFileStream;
   dom:TkbmMWDOMXML;
   nMemo:TkbmMWDOMXMLNode;
   node:TkbmMWDOMXMLNode;
begin
```

Create a DOM XML document. Specify that whitespaces should be preserved and that it should automatically add linefeeds where appropriate to make it human readable.

```
    dom:=TkbmMWDOMXML.Create;
    try
       dom.PreserveWhiteSpace:=true;
       dom.AutoLineFeed:=true;
```

Create the first declarative node of the XML document. Namely the <?xml…?> node. In addition a name/value type attribute is defined on the node.

```
        // Create XML node.
        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='xml';
        node.IsDeclaration:=true;
        node.AttribByName['version']:='1.0';
        dom.Root.Nodes.AddNode(node);
```

Create the memo container node:

```
        // Create Memo node.
        nMemo:=TkbmMWDOMXMLNode.Create(dom);
        nMemo.Name:='Memo';
        nMemo.AttribByName['distribution']:='internal';
        dom.Root.Nodes.AddNode(nMemo);
```

Create the from, to, date and content data nodes. The information is provided directly using the Data property of the node:

```
        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='from';
        node.Data:='Henry Stuff III';
        nMemo.Nodes.AddNode(node);

        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='to';
        node.Data:='All employees';
        nMemo.Nodes.AddNode(node);

        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='date';
        node.Data:='6 April 2001';
        nMemo.Nodes.AddNode(node);
        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='content';
        node.Data:='It is with great delight that I announce some results of our' +
                   'quarterly cost cutting efforts. Over this past quarter, the customer' +
                   'service segment of our business has managed to reduce customer' +
                   'returns by over 50%. This has netted our business ' +
                   'incredible savings and shows great ingenuity on the' +
                   'part of our customer service staff. To quickly summarize two of the' +
                   'strategies involved, the various customer service groups decided to' +
                   'relocate the service desk at the back of each store' +
                   'instead of the front. The width of aisles in that section of the' +
                   'store was reduced, making it more difficult for customers to reach' +
                   'the service desk. The hiring of people unable to speak the English' +
                   'language also had a positive impact on the ability of the customer' +
                   'to return merchandise. A hearty congratulations and job well done to' +
                   'our teammates in customer service!';
        nMemo.Nodes.AddNode(node);
```

Update the internal levels, search paths, referneces etc. of the DOM tree. It is optional unless you want to use the autoindenting facility or later on want to use GetNode or GetRelNode on the tree:

```
        dom.Update;
```

Store the XML in a file:

```
        fs:=TFileStream.Create('demo.xml',fmCreate);
        try
           dom.SaveToStream(nil,fs);
```

```
        finally
           fs.Free;
        end;
```

Clean up:

```
     finally
        dom.Free;
     end;
end;
```

The generated XML document do not contain any information about the types of the data values in the document. For that reason an XML schema (XSD) should be provided. The schema can be built manually using the TkbmMWDOMXML class the same way as a normal XML document as the XSD is itself an XML document.

However its also possible to include data type information along with the XML data in the same document.

The following is an example of how to do that:

Declarative bits:

```
uses kbmMWXML, kbmMWGlobal;

procedure TForm1.Button1Click(Sender: TObject);
var
   fs:TFileStream;
   dom:TkbmMWDOMXML;
   nMemo:TkbmMWDOMXMLNode;
   node:TkbmMWDOMXMLNode;
   fmt:TFormatSettings;
begin
```

Create a DOM XML document. Specify that whitespaces should be preserved and that it should automatically add linefeeds where appropriate to make it human readable.

```
     dom:=TkbmMWDOMXML.Create;
     try
        dom.PreserveWhiteSpace:=true;
        dom.AutoLineFeed:=true;
```

Create the first declarative node of the XML document. Namely the <?xml…?> node. In addition a name/value type attribute is defined on the node.

```
        // Create XML node.
        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='xml';
        node.IsDeclaration:=true;
        node.AttribByName['version']:='1.0';
        dom.Root.Nodes.AddNode(node);
```

Create the memo container node. As the elements of this container will reference XML datatypes, some namespace references should be added to make the document valid. The namespaces should

always be named xsd and xsi (which is the defacto standard for those two namespace definitions) and should usually always use the URI's at www.w3.org as shown:

```
        // Create Memo node.
        nMemo:=TkbmMWDOMXMLNode.Create(dom);
        nMemo.Name:='Memo';
        nMemo.AttribByName['distribution']:='internal';
        nMemo.NameSpaces.CreateAddNameSpace('xsd',
                                            'http://www.w3.org/2001/XMLSchema',nMemo);
        nMemo.NameSpaces.CreateAddNameSpace('xsi',
                                            'http://www.w3.org/2001/XMLSchema-instance',nMemo);
        dom.Root.Nodes.AddNode(nMemo);
```

Create the from, to, date and content data nodes as type defined nodes:

```
        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='from';
        node.SetAsString('Henry Stuff III');
        nMemo.Nodes.AddNode(node);

        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='to';
        node.SetAsString('All employees');
        nMemo.Nodes.AddNode(node);

        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='date';
        node.SetAsDate(mwscFixed,fmt,EncodeDate(2001,4,6));
        nMemo.Nodes.AddNode(node);

        node:=TkbmMWDOMXMLNode.Create(dom);
        node.Name:='content';
        node.SetAsString('It is with great delight that I announce some results of our' +
                   'quarterly cost cutting efforts. Over this past quarter, the customer' +
                   'service segment of our business has managed to reduce customer' +
                   'returns by over 50%. This has netted our business ' +
                   'incredible savings and shows great ingenuity on the' +
                   'part of our customer service staff. To quickly summarize two of the' +
                   'strategies involved, the various customer service groups decided to' +
                   'relocate the service desk at the back of each store' +
                   'instead of the front. The width of aisles in that section of the' +
                   'store was reduced, making it more difficult for customers to reach' +
                   'the service desk. The hiring of people unable to speak the English' +
                   'language also had a positive impact on the ability of the customer' +
                   'to return merchandise. A hearty congratulations and job well done to' +
                   'our teammates in customer service!');
        nMemo.Nodes.AddNode(node);
```

Update the internal levels, search paths, referneces etc. of the DOM tree. It is optional unless you want to use the autoindenting facility or later on want to use GetNode or GetRelNode on the tree:

```
        dom.Update;
```

Store the XML in a file:

```
        fs:=TFileStream.Create('demo2.xml',fmCreate);
        try
           dom.SaveToStream(nil,fs);
        finally
           fs.Free;
        end;
```

Clean up:

```
     finally
        dom.Free;
     end;
end;
```

The output is now slightly different, but still valid XML, just now containing XML datatypes for all elements:

```
<?xml version="1.0" ?>
<Memo distribution="internal" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <from xsi:type="xsd:string">Henry Stuff III</from>
    <to xsi:type="xsd:string">All employees</to>
    <date xsi:type="xsd:date">2001-04-06</date>
    <content xsi:type="xsd:string">It is with great delight that I
      announce some results of ourquarterly cost cutting efforts. Over
      this past quarter, the customerservice segment of our business has
      managed to reduce customerreturns by over 50%. This has netted our
      business incredible savings and shows great ingenuity on thepart of
      our customer service staff. To quickly summarize two of
      thestrategies involved, the various customer service groups decided
      torelocate the service desk at the back of each storeinstead of the
      front. The width of aisles in that section of thestore was reduced,
      making it more difficult for customers to reachthe service desk.
      The hiring of people unable to speak the Englishlanguage also had a
      positive impact on the ability of the customerto return
      merchandise. A hearty congratulations and job well done toour
      teammates in customer service!</content>
</Memo>
```

How to get to a specific element in an XML document

There are several ways to access the elements in an already existing DOM XML tree.

Iterating thru the nodes

Each element is called a node in the tree.

The DOM tree has a top node which actually isnt part of the generated XML document, but is used as the root of the tree. It isnt stored in an XML file and is not part of the XML standard, but is there while manipulating a DOM tree in kbmMW's XML implementation. The top node is accessed via the Root property.

```
var
```

```
   i:integer;
begin
   for i:=0 to dom.Root.Nodes.Count-1 do
         ShowMessage('This element is in the root of the DOM XML tree: ' +
                     dom.Root.Nodes[i].Name;
end;
```

Subnodes can be accessed via the Nodes list as in this recursive example:

```
procedure ShowNodes(ANode:TkbmMWDOMXMLNode);
var
   i:integer;
begin
   for i:=0 to ANode.Nodes.Count-1 do
   begin
        ShowMessage(ANode.Nodes[i].Name);
        ShowNodes(ANode.Nodes[i]);
   end;
end;

procedure ShowAll;
begin
   ShowNodes(dom.Root);
end;
```

Accessing the nodes this way, require good knowledge about how the XML document is built. If for example XML references are used, the document may actually have sub documents on a higher level, which is then references from a main document.
That makes it complex to always know exactly where the specific node one is searching for exists in the DOM tree as the tree is a 1 to 1 representation of the actually loaded XML document.

An example of an XML document with references could be:

```
<?xml….?>
<node1>
   <node2 ref=”node3”/>
   <node3 id=”node3”>
      <node4 …/>
      <node5 …/>
   </node3>
</node1>
```

which essentially represent the same logical structure as:

```
<?xml….?>
<node1>
   <node2>
```

```
      <node4 …/>
      <node5 …/>
   </node3>
</node1>
```

It shows that although the logical storage is the same (node4 and node5 is part of node2 which is in turn part of node1.
In the first example, node4 and node5 can also be seen as being part of node3 which is in turn part of node1.

This example is quite simple, but much more complex XML documents are using references.

For that reason, kbmMW implements a different way to access nodes by something called paths. kbmMW will automatically detect where nodes reference other subtrees/nodes and take that into account when searching a node via a path.

Searching a node via path

A path is a string of element names seperated by forward slash (/).
Eg. node 2 in the previous examples could be references with the path: //node1/node2
and node 4 by: //node1/node2/node4

Notice that //node1/node2/node4 will work regardless of which of the 2 XML documents where used as kbmMW detects the references and use that information seemlessly.

The extra / in front of the path represents the Root node of the DOM tree. In other words, we search nodes relative to the root node.

Eg.

```
var
   n:TkbmMWDOMXMLNode;
begin
   ..
   n:=dom.GetNode('//node1/node2/node4');
   if n<>nil then
     ShowMessage('Found node4');
   ..
end;
```

Its also possible to search a node relative to another node. Eg.

```
var
   topnode,n:TkbmMWDOMXMLNode;
begin
   ..
   topnode:=dom.GetNode('//node1');
   if topnode<>nil then
   begin
      n:=dom.GetRelNode(topnode,'node2/node4');
      if n<>nil then
         ShowMessage('Found node4');
   end;
   ..
end;
```

Attributes

Attributes are widely used in XML. Each element/node can have any number of attributes which often are used for values that can be expressed as strings or integers, instead of embedding those values as a data element of its own.

In our 'memo' XML sample we have at least two attributes (in the typed value sample we have more), namely in the xml declaration (the version attribute) and in the Memo section where the distribution attribute is.

```
<?xml version="1.0" ?>
<Memo distribution="internal">
```

Its very easy to create and access attributes on a node. To access all attributes on a node as a list of strings, use the Attribs property of the DOM object. Eg.

```
ShowMessage('The attributes of the node is:'+dom.Root.Nodes[0].Attribs.Text);
```

Will show the xml declaration nodes attributes.

To get the (string) value of a named attribute use:

```
var
   s:string;
begin
   ..
   s:=dom.Root.Nodes[0].AttribByName['version'];
```

```
   ShowMessage(s);
end;
```

In case our 'memo' XML sample is loaded into the DOM tree, we should see the string 1.0 as result.

If the attribute doesn't exist, an empty string is returned.

To create or set an attribute use:

```
dom.Root.Nodes[0].AttribByName['version']:='1.1';
```

This sets the version attribute of the xml declaration node to 1.1 instead of 1.0.
If the version attribute doesn't already exist, it will be automatically created.
Although attributes are case sensitive in XML, AttribByName search case insensitive to simplify the life of the developer. If case sensitivity is an issue, use the Attribs string list instead.

Its possible to delete an attribute by:

```
s:=dom.Root.Nodes[0].DeleteAttribByName('version');
```

If the version attribute exists (and it does in our sample), it will be removed.

All attributes of a node can be set to the value of a stringlist, which could for example be the attributes of another node. Eg.

```
s:=dom.Root.Nodes[0].AssignAttribs(dom.Root.Nodes[1].Attribs);
```

Now the xml declaration node will have a copy of the attributes of the <memo> node.
This doesn't make much sense in this sample as it would make the generated XML invalid as the xml declaration node always should contain a version attribute.

Namespaces and schemas

Another important topic in XML, is namespaces.
A namespace is a space which contain names. That also means that there can exist multiple spaces which contains names that may be the same names as the names in other spaces.

Hence the combination of a namespace and a name is a unique identifier.

A namespace is in XML declared like this:
```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

which declares the namespace xsd and xsi. The namespaces points to a web document which is called an XML schema (xsd) which contains the declarations of the data types available in that name space. The documents can be seen in a webbrowser by opening these URL:

http://www.w3.org/2001/XMLSchema.xsd
http://www.w3.org/2001/XMLSchema-instance.xsd

These documents are essentially also only XML documents and could thus be loaded into a kbmMW XML DOM object and examined there if needed.

Later in the sample 'memo' XML document, we refer to the namespace when specifying some attributes. Eg.

```
<from xsi:type="xsd:string">Henry Stuff III</from>
```

This node refers to two namespaces, xsi and xsd which we have already declared. The declarations for the xsi namespace comes from the XMLSchema-instance.xsd document and the declarations for the xsd namespace comes from the XMLSchema.xsd document.

xsi:type indicates that we specify a value in this attribute which must contain the values allowed in the schema defined by the xsi namespace. xsd:string refers to the definition of string in the namespace xsd.

In the schema, string is defined to be:

```
<xs:simpleType name="string" id="string">
  <xs:annotation>
    <xs:appinfo>
        <hfp:hasFacet name="length" />
        <hfp:hasFacet name="minLength" />
        <hfp:hasFacet name="maxLength" />
        <hfp:hasFacet name="pattern" />
        <hfp:hasFacet name="enumeration" />
        <hfp:hasFacet name="whiteSpace" />
        <hfp:hasProperty name="ordered" value="false" />
        <hfp:hasProperty name="bounded" value="false" />
        <hfp:hasProperty name="cardinality" value="countably infinite" />
        <hfp:hasProperty name="numeric" value="false" />
    </xs:appinfo>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-2/#string" />
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="preserve" id="string.preserve" />
  </xs:restriction>
</xs:simpleType>
```

This describes what a string element can contain of meta information which again typically are attributes.
For more information about how schemas work, please consult
http://www.w3.org/2001/XMLSchema

kbmMW knows about namespaces and new ones can be declared if needed.
This happens in via a nodes NameSpace property. Each node/element can belong to zero or more namespaces. All subnodes automatically inherits namespaces from parent (embedding) nodes.

To declare a namespace reference on a node do:

```
somenode.NameSpaces.CreateAddNameSpace('namespacename','http://yourplace.com/yourxsd',somenode);
```

Now this node declares and owns a namespace of its own which subnodes can also refer to.

Its possible to get the URI (the web link) of a defined namespace via the GetURI function of the namespaces property. Eg.

```
ShowMessage('The URI is:'+node.NameSpaces.GetURI('xsd'));
```

which will show: **http://www.w3.org/2001/XMLSchema**
even though somenode hasn't implicitely defined the schema on it, but one of its parents has.

XML support reference

kbmMW's XML DOM engine supports the following XML element tags:

XML syntax	Explanation	How to test for it or set it?
<!-- … -->	Comment	node.IsComment
<![CDATA[…]]>	Undefined non interpreted data	node.IsCData
<? … ?>	Declaration	node.IsDeclaration
<! … >	Markup declaration	node.IsMarkupDeclaration
< … />	Standard ended element	none of the above is true
< … >	Standard open element with sub elements	none of the above is true
</ … >	Standard close element	none of the above is true

The following special characters are always converted forth and back if they are part of attribute values or element data (except for in a CDATA element):

&	&amp;
'	&apos;
"	&quot;
<	&lt;
>	&gt;

Nodes can be type set with values to any of the following types

Datatype	XML type	Explanation
String	xsi:type=xsd:string xsi:type=base64	If the string do not need BASE 64 conversion. If the string require BASE 64 conversion. It is automatically detected if a string needs conversion or not. If it is not base 64 converted, special XML character conversion is still applied.
WideString	xsi:type=xsd:string xsi:type=base64	The string is automatically converted to UTF 8. After the UTF 8 conversion, the same operation applies as with the String type.
Int	xsi:type=xsd:int	The value is a signed integer value.
Float	xsi:type=xsd:double	Depending on the conversion rules given in the call to setting or getting the value, the value will either be formatted according to a locale, or as a standard XML formatted floating point value which a dot as decimal comma, and no thousands seperators.
DateTime	xsi:type=xsd:datetime	Depending on the conversion rules given in the call to setting or getting the value, the value will either be formatted according to a locale, or as a standard XML Date/Time formatted string: YYYY-MM-DDTHH:mm:SS.NNN where NNN is optional.
Date	xsi:type=xsd:date	Depending on the conversion rules given in the call to setting or getting the value, the value will either be formatted according to a locale, or as a standard XML Date formatted string: YYYY-MM-DD
Time	xsi:type=xsd:time	Depending on the conversion rules given in the call to setting or getting the value, the value will either be formatted according to a locale, or as a standard XML Time formatted string: HH:mm:SS.NNN where NNN is optional.

Variant		Sets the value according to the variant type. varOleStr -> WideString varDate -> DateTime varBoolean -> Boolean varDouble, varSingle and varCurrency -> Float varInt64, varShortInt, varWord, varLongWord, varSmallInt, varByte, varInteger -> Int varNull and varEmpty -> empty element. varString -> String variant byte array -> VariantByteArray other variant array -> VariantArray
VariantArray	xsd:type=arrayType low=lowindex high=highindex dim=1	The node will contain a number of subnodes that are the values of the array.
VariantByteArray	xsd:type=xsd.base64 low=lowindex high=highindex dim=1	The node will contain Mime encoded data.
Boolean	xsi:type=xsd:boolean	true or false
Stream	xsi:type=xsd:string	The node will contain a subnode named ‘base64data’ which in turn contains mime encoded data.

This concludes the whitepaper on the kbmMW XML DOM class.

Kim Madsen
Components4Developers