

Document	:	XmlInformation
Author	:	M. Majoor
Subject	:	Information on the XML processing of the application

Revision/Date	Author	Description
20070311	M.Majoor	Initial draft
20081005	M.Majoor	Minor correction in example missing a separator (~ ~). Added commands to set a save path and the HTML file to show after a transformation.

Introduction

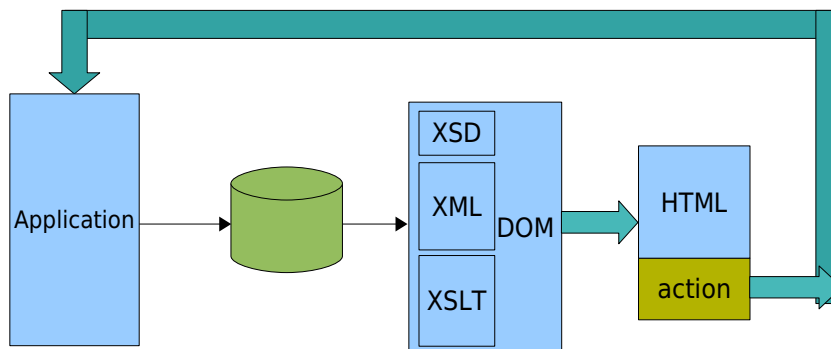
This document describes the fundamentals behind the XML processing taken part in the application. XML stands for Extensible Markup Language. Together with the related DOM (Document Object Model) and XSL (Extensible Stylesheet Language), it allows the application to process information using a 'human readable' language.

XML is basically nothing more than an ASCII file, using some specific rules. The application uses XML files as if they are databases. Using the DOM, which is used internally only, selective data can be extracted from the XML file. Using the XSL, the (selective) data can be converted into a presentable format (HTML in this case). This document will not go into detail on XML itself, except where it is needed. It assumes that the reader is already familiar with XML. Since this document is mainly documentation on how a user can create its own presentation of the XML data, it is essential that the user has some basic knowledge on XSLT too. XSLT is part of XSL, where the 'T' stands for 'Transformation'. The application uses a XSLT file to convert the XML data into HTML data. Because HTML data is used, knowledge about HTML is also assumed. Furthermore, a XSD file (**X**ML **S**chema **D**efinition) is used, which describes the structure of the XML document.

Thus, a basic knowledge of XML, XSD, XSL (XSLT) and HTML is assumed. If someone is not familiar with these, then there is plenty (!) information available on the 'net.

Processing

The processing taken place in the application, related to XML, is shown in the next figure:



The application selects the XML and the XSLT file to process. These files are passed through the DOM. This generates a HTML file, which is being displayed. The HTML form allows for certain actions to be generated. These actions are fed back to the application. These actions are not based on any standard, because there is none. There is no standard on XML binding in HTML, although it is possible using some specific browsers and/or Java scripts. Because of all the problems concerning having the correct browser or Java interpreter a proprietary method has been implemented.

Some things to remember is that XML data is based on 'nodes'. A node itself is an element which can contain text, attributes, and other elements.

With the transformation of XML data into HTML data, elements (and text and attributes) can be displayed. If the HTML data is only used to display elements, then there are no special considerations involved. However, if an element is being changed, or the HTML data wants to select another element, then there is one important consideration:

- ➔ There must be a reference to the active element. This is something the XSLT transformation file must generate.
Further in the document an example is shown on how such a reference can be generated automatically.

- ➔ The data being fed back to the application is by means of a text string. This text includes the command to perform, and any additional parameters. Both the command and the parameters are made up of a *name* and a *value*, separated by ':'. Commands and parameters are separated by '~|~' (note the leading and trailing space!). Any leading/trailing spaces for the *name* are removed. Spaces in the *value* are left intact (as if the value is a string). When numbers are used for a value, then these can either be in decimal, or in a hexadecimal format. Only integer numbers are supported. For a hexadecimal number use the prefix '\$'. The *name* is not case sensitive. The value might be case sensitive and is passed on as it is.

Examples of a command string are:

COMMAND:comment ~|~ PARAMETER1:100 ~|~ PARAMETER2:\$1234 ~|~ PARAMETER3:A string
~|~ PARAMETER4:Another string (etc)

Note: Instead of 'COMMAND:comment' it is also allowed only to use the command (e.g. 'COMMAND')

What the application does, is that it searches first for a ':' in the string. All text until this first ':' is the parameter name (except any spaces). Then the search is for '~|~' (or the end of the line, and note the leading space!). All text until this '~|~' is the parameter value.

Note for application programmers: The string is passed on to the application as a null terminated string. It is the responsibility of the application to release the memory allocated for this string.

Commands

Command	Parameters [==optional]	Type	Description
FORM			Set form parameters
	[caption]	String	Set caption of form
	[top]	Number	Set top position
	[left]	Number	Set left position
	[width]	Number	Set width
	[height]	Number	Set height
MESSAGE			<p>Send a Windows message. Note: Messages are send using 'SendMessage'. The windows handle is retrieved using 'FindWindow(WindowClass, WindowName)'. Example: MESSAGE ~ ~ windowclass:application ~ ~ message:\$1234 ~ ~ wparam:0 ~ ~ param:0</p> <p>To close the application: MESSAGE:close application ~ ~ windowclass:application ~ ~ message:\$0010</p>
	WindowClass	String	<p>The windows handle is retrieved using 'FindWindow(WindowClass, WindowName)'. Special, reserved, names, are: APPLICATION: Sends it to the application itself (default) BROADCAST: Broadcasts the message SELF: Sends it to the active form itself</p>
	WindowName	String	Window name of the window to send the message to.
	Message	Number (32-bit)	Message identifier
	wParam	Number (32-bit)	Additional message-specific information
	lParam	Number (32-bit)	Additional message-specific information
XML			<p>XML parameters are instructions which are processed in the order that they are received. Most instructions don't generate an action, but are only used to define a setting, which is then later used by an action. Note that some instructions can be duplicated. For instance, multiple attributes can be defined, since an element can have multiple attributes. Some of the instructions need a transformation action to be of any use. Some actions only have an impact on a single element, or have an effect on multiple elements. The element selection is an XPath expression and can be thought of as a directory pointer. By default all elements are selected, meaning that the directory pointer points to the 'root'.</p> <p>General syntax of an XML tag to which the instructions refer to:</p> <pre><elementname attributename:attributevalue attributename:attributevalue> elementvalue </elementname></pre>

[action]	String	Action to execute. Uses the instructions accumulated so far. After the action has executed all accumulated instructions are removed. An invalid action also removed all accumulated instructions.		
		<table><tr><td>action:transform</td><td>Do a (re-) transformation with the selected element(s). By default all elements are 'selected' (the root: '/'). XML ~ ~ select:/ ~ ~ action:transform XML ~ ~ select:satellite=Astra 19.2E ~ ~ action:transform Note that a transformation generates a new file. This file is saved using the savepath:path parameter using the XSL filename, but with a HTML extension. The current HTML file is then reloaded after the transform. Using the parameter htmlfile:filename.html the HTML file name can be specifically set. This must be set before the action parameter: XML ~ ~ savepath:html ~ ~ htmlfile:html\frames.html ~ ~ select:satellite=Astra 19.2E ~ ~ action:transform</td></tr></table>	action:transform	Do a (re-) transformation with the selected element(s). By default all elements are 'selected' (the root: '/'). XML ~ ~ select:/ ~ ~ action:transform XML ~ ~ select:satellite=Astra 19.2E ~ ~ action:transform Note that a transformation generates a new file. This file is saved using the savepath:path parameter using the XSL filename, but with a HTML extension. The current HTML file is then reloaded after the transform. Using the parameter htmlfile:filename.html the HTML file name can be specifically set. This must be set before the action parameter: XML ~ ~ savepath:html ~ ~ htmlfile:html\frames.html ~ ~ select:satellite=Astra 19.2E ~ ~ action:transform
		action:transform	Do a (re-) transformation with the selected element(s). By default all elements are 'selected' (the root: '/'). XML ~ ~ select:/ ~ ~ action:transform XML ~ ~ select:satellite=Astra 19.2E ~ ~ action:transform Note that a transformation generates a new file. This file is saved using the savepath:path parameter using the XSL filename, but with a HTML extension. The current HTML file is then reloaded after the transform. Using the parameter htmlfile:filename.html the HTML file name can be specifically set. This must be set before the action parameter: XML ~ ~ savepath:html ~ ~ htmlfile:html\frames.html ~ ~ select:satellite=Astra 19.2E ~ ~ action:transform	
		<table><tr><td>action:update</td><td>Update first of the selected element(s). XML ~ ~ select:satellite[1] ~ ~ value:Astra 19.2East ~ ~ action:update</td></tr></table>	action:update	Update first of the selected element(s). XML ~ ~ select:satellite[1] ~ ~ value:Astra 19.2East ~ ~ action:update
		action:update	Update first of the selected element(s). XML ~ ~ select:satellite[1] ~ ~ value:Astra 19.2East ~ ~ action:update	
		<table><tr><td>action:insert</td><td>Insert after the selected element. The data is inserted <i>before</i> the selected element.</td></tr></table>	action:insert	Insert after the selected element. The data is inserted <i>before</i> the selected element.
action:insert	Insert after the selected element. The data is inserted <i>before</i> the selected element.			
<table><tr><td>action:remove</td><td>Remove the selected element.</td></tr></table>	action:remove	Remove the selected element.		
action:remove	Remove the selected element.			
<table><tr><td>action:clone</td><td>Create an identical copy of the selected element (the whole structure and contents).</td></tr></table>	action:clone	Create an identical copy of the selected element (the whole structure and contents).		
action:clone	Create an identical copy of the selected element (the whole structure and contents).			

[name]	String	name:elementname Set an element name. Only used for new elements (e.g. insert).		
[value]	String	value:elementvalue Set the element value. If multiple value definitions are used then they are all concatenated together, separated by a CR/LF.		
[attribute]	String	attribute:attributename=attributevalue Set an attribute.		
The following parameters are actions which are executed immediately, without removing any accumulated instructions:				
[select]	String	select:XPath Select one element or multiple elements. XPath expressions are used to select these.		
[element]	String	element:XPath=elementvalue element:elementvalue When a whole form is 'submitted' then this format is internally generated to update all controls on a form. Short form for the following individual instructions, except that accumulated instructions are not removed: select:.... ~	~ value:.... ~	~ action:update Without the XPath (which is the element name) the active element is used.
[xmlfile]	String	xmlfile:xmlfilename Set a new XML file.		

	[xsltfile]	String	Xsltfile:xsltfilename Set a new XSLT file.
	[htmlfile]	String	htmlfile:htmlfilename Set a new HTML file.
	[savepath]	String	savepath:path Sets the path for saving of a transformed XSL file.
	[save]		save=yes save=no save=0 save=false Define if the XML file should be saved when it is closed.
	[savenow]		savenow: Save the XML file.

The HTML file being generated is being displayed by an internal HTML browser. Because of the interaction with the application, some of the attributes associated with certain HTML objects, are used in a specific way. The browser specifics, and the implementation of HTML objects is shown in the next table. Please note that this list might not be complete. Some parameters might be optional even if this is not indicated in the table. Have a look at the help files and samples of the HTMLViewer component.

HTML objects

Object	Attributes	Type	Description
<body ...>			Color and background information of the form. Example: <code><body text="00FF00 link="FF0000" background="background.gif"></code>
	text	String	Normal text color.
	link	String	Link (hotspot) color.
	vlink	String	Color for visited link.
	olink	String	Color when mouse moved over a link.
	bgcolor	String	Background color.
	background	String	Background image file.
<form ...>			The form tag, typically in the header of the HTML document, has the attributes which are passed on when a submit event is generated (by means of a submit button). Example: <code><form action="" method="" target="" enctype=""></code>
	action	String	Action to perform. Examples: MESSAGE ~ ~ "XML ~ ~ action:variables" "XML ~ ~ action:select" "XML ~ ~ action:insert-before"
	method	String	Not used
	target	String	Element to select. Generates the 'target=...' parameter if not empty.
	enctype	String	Not used
	When the submit button is pressed above parameters are passed on, with all the control elements of the HTML form. With a few exceptions, from a control element typically the contents of the <i>name</i> attribute and the <i>value</i> attributes are passed on as a list of <i>name/value</i> strings. These are to be seen as the individual parameters which are used as parameters for the command. The following string is internally generated (with the bold items being added): command ~ ~ target:target ~ ~ name:value ~ ~ name:value ~ ~ name:value ...		

<meta ...>			With the meta tag startup commands can be defined. Two of the attributes of the meta tag can be used for two separate commands. Note that these are handled in the order in how they appear. Note: The XML part also generates an internal meta tag. Because of this it is best to place our own meta tag after the <head> tag in the HTML document. Example: <meta name="caption of the form">
	[http_eq]	String	Not used (text is restricted)
	[name]	String	First command
	[content]	String	Second command
<input type="edit' ...>			Text input field. Note: The font used is the <pre> or <code> font. Example: <input type="edit" name="name edit" value="value edit">
	name	String	Name identifier used when a form is submitted.
	value	String	The initial contents of the input field.
	[size]	Number	Size of input field.
<input type="password' ...>			Text input field (input characters are echoed as '*'). Example: <input type="password" name="name password" value="value password">
	name	String	Name identifier used when a form is submitted.
	value	String	The initial contents of the input field.
<input type="submit' ...>			The form submit button (uses the <form> tag settings). This generates multiple events (onclick/form submit). Example: <input type="submit" value="submit button" onclick="onclick button">
	value	String	Caption of the button.
	[onclick]	String	The text passed on when the button is clicked. This event is generated before the submit event is generated.
<input type="button" ...>			Button. Example: <input type="button" value="value button" onclick="onclick button"> Close application: <input type="button" value="Close application" onclick="MESSAGE::;windowclass:application;; message:\$0010"
	value	String	Caption of the button.
	[onclick]	String	The text passed on when the button is clicked.

<code><input type="radio" ...></code>			A radio button. A radio button itself belongs to a radio button group. Of this group only one of the radio buttons will be checked. Example: <code><input type="radio" name="group" checked="" onclick="onclick radio"></code>
	name	String	The group name. Radio buttons belonging to the same group should have the same group name. Of such a group only a single radio button is active.
	value	String	Name identifier used when a form is submitted.
	[checked]	String	Indicates that the radio button is checked. Only one radio button in a group can be checked at one time. For an unchecked radio button omit the 'checked=""' attribute. Typical use is 'checked="checked"'. Note: If two radio buttons of the same group have this 'checked=' attribute then both will initially be checked. Only when a radiobutton of the group is altered, only one radio button will be checked.
	[onclick]	String	The text passed on when the button is clicked.
<code><input type="checkbox" ... ></code>			A checkbox. Example: <code><input type="checkbox" checked="checked" onclick="onclick checkbox"></code>
	name	String	Name identifier used when a form is submitted.
	[checked]	String	Indicates that the checkbox is checked. For an unchecked checkbox omit the 'checked=""' attribute. Typical use is 'checked="checked"'. Note: If two checkboxes of the same group have this 'checked=' attribute then both will initially be checked. Only when a checkbox of the group is altered, only one checkbox will be checked.
	[onclick]	String	The text passed on when the button is clicked.
<code><select ... <option ...></select></code>			A list box. When a form is submitted only the selected items are passed on when [multiple] is used. Without [multiple] there is always one item selected, and this one is passed on. Example: <code><select name="listbox" size=1 multiple onclick="onclick listbox"><option>1<option selected>2</select></code>
	name	String	Name identifier used when a form is submitted.
	size	Number	Number of items being displayed.
	[multiple]		Indicates that multiple selections are possible (including <i>no</i> selection). If multiple selections are possible, then one must select an item from the list box. The one being shown is not necessarily a selected item!
	[onclick]	String	The text passed on when something in the list box is clicked. Typically not used, since this only indicates that the list box has been clicked and no feedback on the item is passed on.
	<code><option [selected]>text</code>		Defines a line in the list box. The optional 'selected' parameter indicates if the item is to be a selected item.

<input type="image" ...>			An image. This generates multiple events (alt/onclick). Example: <code><input type="image" src="src image" alt="alt image" onclick="onclick image"></code>
	src	String	Name of image file.
	alt	String	The text passed on when the button is clicked. This is processed before the 'onclick' is processed.
	[onclick]	String	The text passed on when the button is clicked.
			An image. Note: Although clicking on this image is possible, the mouse cursor does not change when hovering over the image. Example: <code></code>
	src	String	Name of image file.
	alt	String	The text passed on when the button is clicked.
<a ... 			Image with hot spot. This generates multiple events (alt/href). Note: Because of the hot spot the mouse cursor does change its appearance. Also note that the hot spot makes the image appear slightly different. Example: <code></code>
	src	String	Name of image file.
	alt	String	The text passed on when the button is clicked. This is processed before the 'href' is processed.
	href	String	The text passed on when the hot spot is clicked.
<map ...>			Image with multiple (polygon) hot spots. This generates multiple events (alt/href). Example: <code> <map name="imagemap"> <area shape="polygon" href="hotspot text" coords="XX,XX,XX,XX,XX,XX,XX,XX"> <area shape="polygon" href="hotspot text" coords="XX,XX,XX,XX,XX,XX,XX,XX"> </map></code>
	src	String	Name of image file.
	alt	String	The text passed on when the button is clicked. This is processed before the 'href' is processed.
	href	String	The text passed on when the hot spot is clicked.

<textarea ...>			Text area. Example: <textarea name="name textarea" style="width:100%" rows="3" wrap="soft" onclick="onclick textarea"> ... </textarea>
	name	String	Name identifier used when a form is submitted.
	onclick	String	The text passed on when the button is clicked. Note: Typically not used because it only re-acts when the mouse is clicked inside the text area, and not when the text changes or the text area is no longer selected.
	[rows]	Number	Number of rows.
	[cols]	Number	Number of columns.
	[wrap]	String	Method for wrapping the text. SOFT HARD

Example of a HTML file with some the controls

```
<html>
<font face="arial" size="2">
<meta name="form" -!- caption:Caption of the form">
<form action="action form" method="get" target="target form" enctype="enctype form">
  <input type="edit"      name="name edit"      value="value edit"      ><BR>
  <input type="password"  name="name password" value="value password"><BR>
  <input type="button"    value="value button"  onclick="9 10 11 12"><BR>

  <input type="radio"     name="group1" value="value radio button 1, of group 1" checked="checked" onclick="onclick radio 1">Radio button 1, of group 1<BR>
  <input type="radio"     name="group1" value="value radio button 2, of group 1"                   onclick="onclick radio 2">Radio button 2, of group 1<BR>
  <input type="radio"     name="group1" value="value radio button 3, of group 1"                   onclick="onclick radio 3">Radio button 3, of group 1<BR>
  <input type="radio"     name="group2" value="value radio button 1, of group 1" checked="checked" onclick="onclick radio 1">Radio button 1, of group 2<BR>
  <input type="radio"     name="group2" value="value radio button 2, of group 2"                   onclick="onclick radio 2">Radio button 2, of group 2<BR>
  <input type="radio"     name="group2" value="value radio button 3, of group 3"                   onclick="onclick radio 3">Radio button 3, of group 2<BR>

  <input type="checkbox"     name="name checkbox" checked="" onclick="onclick checkbox">A checkbox<BR>

  Item lists:
  <select name="listbox1" size=1 onclick="onclick list box 1">
    <option>Item 1
    <option>Item 2
    <option selected>Item 3
    <option>Item 4
  </select>
  <select name="listbox2" multiple size=4 onclick="onclick list box 2">
    <option selected>Item 1
    <option>Item 2
    <option selected>Item 3
    <option>Item 4
  </select>

  <input type="image" src="note.gif" onclick="onclick image">An image<BR>
  An image 2<BR>
  <a href="hotspot text">An image 3</a><BR>

  <a name="usemap">
    <BR>
  </a>
  <map name="imagemap">
    <area shape="polygon" href="hotspot text 1" coords="06,40,21,03,36,39,21,36">
    <area shape="polygon" href="hotspot text 2" coords="47,05,62,07,78,05,62,41">
  </map>

  <textarea name="name textarea" style="width:100%" rows="3" wrap="soft" onclick="onclick textarea">Text area contents</textarea><BR><BR>

  <input type="submit" value="submit button"><BR>

</form>
</font>
</html>
```

XSL templates

When a template is applied to generate a HTML document, sometimes the data needs to be 'editable'. This also implies that the (new) data needs to be fed back to the the source (so it is updated). For this to work, one needs to know which element is being active. For this, two 'functions' can be used. Note that these functions need to be included in the template.

Here are the support 'functions' for this. Note that these 'functions' are called using
 <xsl:call-template name="..."/> (were ... is the name of the 'function')

<!-- Here follow some 'functions' which can be applied to position
 which need to include the reference to a node

Usage:

<xsl:call-template name="..."/>

-->

```
<!-- Get node path (for element) -->
<xsl:template name="elementPath">
  <xsl:for-each select="(ancestor-or-self::*)">/*[<xsl:value-of select="count(preceding-
sibling::*)" />]</xsl:for-each>
</xsl:template>
```

```
<!-- Get node path (for attribute) -->
<xsl:template name="attributePath">
  <xsl:for-each select="parent::*">
    <xsl:call-template name="elementPath" />
  </xsl:for-each>
  <xsl:text>/@</xsl:text>
  <xsl:value-of select="name(.)" />
</xsl:template>
```