



AltioLive Developer Training

Version 5.4

3. Referencing Data in AltioLive Applications using XPath

Summary

This follows on from the first steps module by explaining how to use XPath to reference data in AltioLive applications.

Integra SP

88 Wood Street London
EC2V 7RS
United Kingdom

www.altio.com

tel: +44 (0) 20 8528 1045

Contents

Data: XML and XPath	2
Introduction to the XML Path Language (XPath)	2
Data Referencing in AltioLive	2
Location Paths	3
Location Steps	3
Predicates	4
String functions	5
Substitution syntax in AltioLive	5
Summary of XPath notation	7
Example of XML Data and XPath	8
For more information about Xpath	10
Exercises on XPath	11
First Exercise	11
Second Exercise	12
Third Exercise: Using the Data Builder to test the XPath statements	12
Launching the Data Builder	13
Introduction to the Data Builder	13
Testing XPath statements	14
Fourth Exercise: Using Controls with filtered data in the Stocks application	14
Adding two select controls to the Main window	14
Creating values for the Sector select controls	15
Using the Syntax Builder to link the Sector Select control to the Stock List	17
Creating values for the Market select control and link it to the Stock List control	19
Fifth Exercise: associating controls options with XML data	21
Associating data attributes to a select control	21
Appendices	23
Appendix A	23
Appendix B	24

Data: XML and XPath

Introduction to the XML Path Language (XPath)

In the previous session (see: 2. The First Steps to Build your Application) we looked at Datatypes: the representation of data used in the application. Datatypes allow us to easily associate particular types of data with controls, and give us a general idea of the shape of the data.

Datatypes alone do not allow us to be very specific about the data we require in a control. We can populate lists easily but it is difficult to show data in one control that is contextual to data in another control. For this we need to understand XPath statements and how they can be used to reference very specific sections of data.

We have structured our information using XML; now we need to address specific items of data within our XML file. Locating information in an XML document is critical for associating or relating it to other information.

In order to address components you have to know the addressing scheme with which the components are arranged. The basis of addressing XML documents is to think of nodes arranged hierarchically, similar to a tree-shape of nested elements.

XPath defines common semantics and syntax for addressing XML, and bases these primarily on the hierarchical position of components in the tree. This ordering is referred to as document order in XPath. We convey XPath addresses in non-XML syntax.

Data Referencing in AltioLive

The data referencing syntax used in AltioLive is a subset of the XPath notation. It enables us to select one or more XML data elements in a hierarchical structure, relative to a particular element (the context node) or the top element. In AltioLive, the top element refers to each element in the first level below the **<DATA>** element.

There are differences between the AltioLive implementation of XPath and the complete set of XPath functions. With AltioLive, for example, the data referencing syntax returns the value of nodes only.

With AltioLive, a control may have Data Source and Data Field properties. XPath notation is used within the Data Source property of a control, with particular data attributes specified directly at the Data Field property. The elegant way AltioLive specifies data attributes means that a full XPath implementation allowing attributes to be accessed directly is not required and so is not supported.

AltioLive also has extended the XPath syntax to support dynamic substitution as well as adding some additional functions.

The subset of the XPath syntax supported in AltioLive is described below:

Using an extract from the stocks.xml file:

```
<STOCKS>
  <STOCK NAME="Abbey National" MARKET="LONDON" LOW="1170"
HIGH="1192" SECTOR="Banks" STOCK_ID="S1">
    <HISTORY DATE="20020531" OPEN="70.7" CLOSE="77.7"
HIGH="76.5" LOW="69.7" VOL="3122000.0" STOCK_ID="S1"/>
    <HISTORY DATE="20020601" OPEN="77.7" CLOSE="78.8"
HIGH="91.2" LOW="75.0" VOL="3147091.4" STOCK_ID="S1"/>
  </STOCK>
  <STOCK NAME="Alliance & Leic" MARKET="NASDAQ" LOW="763.5"
HIGH="793.5" SECTOR="Banks" STOCK_ID="S2">
    <HISTORY DATE="20020531" OPEN="90.1" CLOSE="93.2"
HIGH="95.8" LOW="73.5" VOL="3122000.0" STOCK_ID="S2"/>
    <HISTORY DATE="20020601" OPEN="93.2" CLOSE="101.7"
HIGH="94.4" LOW="82.4" VOL="2666057.7" STOCK_ID="S2"/>
  </STOCK>
  <STOCK NAME="...>
    <HISTORY ...>
    <HISTORY ...>
  </STOCK>
</STOCKS>
```

Location Paths

- The **/** notation represents a path from the root node of the XML data.
For example: **/STOCKS/STOCK** selects all **STOCK** elements, which are the children of the root element **STOCKS**
- The **//** notation will select all elements in the document which fulfill the following criteria.
For example: **//STOCK** selects all **STOCK** elements in the entire XML data.
- You can use both notations in the same path:
For example: **/STOCKS//HISTORY** selects all history elements under the node **STOCKS**

Please note: Be careful using this as this searches the entire XML data. You should always give the complete path if possible!

Location Steps

Axis specifiers

These XPath terms can be used to specify a particular range of element nodes around a particular context node

- **The ancestor axis:** it contains the ancestors of the context node; these ancestors consist of the parent of the context node and the parent's parent and so on. The ancestor axis will always include the root node, unless the context node is the root node.
For example: **ancestor::STOCK** where a particular **HISTORY** is the context, this references the **STOCK** element that is the parent (or grandparent, great-grandparent, etc.) of the **HISTORY** (but not any other **STOCK** at the same level in the tree).

- **The parent axis:** it contains the parent of the context node, if there is one.
For example: **parent::STOCK** where a particular **HISTORY** is the context, this references the **STOCK** element that is the parent of the **HISTORY** (but not any other **STOCK** ancestors: grandparent, great-grandparent, etc)
- **The ancestor-or-self axis:** contains the context node and the ancestors of the context node.
For example: **ancestor-or-self::STOCK** where a particular **HISTORY** is the context, this references any **STOCK** element that is above the **HISTORY** in the data hierarchy. If the context node were a **STOCK** element, it would also have been referenced.
- **The self axis:** it contains the context node.
For example: **self::HISTORY** where a particular **HISTORY** is the context, this references the **HISTORY**.

There is also support for **preceding-sibling::** and **following-sibling::**.

Node Tests

- The asterisk ***** selects all elements located by preceding path
- Function **name()** returns the name of the element
For example: **//*[name()='HISTORY']** returns every element in the data hierarchy which has the name **HISTORY**
- Function **text()** returns the text within the specified element
For example: **/STOCKS/STOCK/HISTORY[text()='Very cheap']** returns all the **HISTORY** elements where the text within the element is **Very cheap**.

Within an XPath statement they are referred to as **text()** or **name()**. While any case will do, the documentation should be all lower case. Effectively, they are functions that return the text or name respectively.

Predicates

Expressions in square brackets can further specify an element.

- **[n]** A number in the brackets gives the position of the element in the selected set.
For example: **/STOCKS/STOCK/HISTORY[5]** will return the fifth **HISTORY** element found under **STOCKS** and **STOCK** parent elements in the XML data
- **[@ATT='xxxxnn']** Attributes are specified by **@** prefix and values of attributes can be used as selection criteria.
For example: **/STOCKS/STOCK/HISTORY[@DATE='20020601']** will return all the **HISTORY** elements where the value of the **DATE** attribute is "20020601"
- **[@ATT>'n']** The attribute values can be evaluated with Boolean operators.
For example: **/STOCKS/STOCK/HISTORY[@OPEN>'80']** will return all **HISTORY** elements where the value of the **OPEN** attribute is more than **80**.
- **[@ATT_A=/ELEMENT_A[1]/@ATT_B]** Attributes from different elements can be compared.
For example: **/STOCKS/STOCK/HISTORY[@HIGH = /STOCKS/STOCK[1]/@LOW]** will return all the **HISTORY** elements where its **HIGH** attribute matches the **LOW** attribute of the first **STOCK** element.
- **/ELEMENT_A[@ATT_A=ELEMENT_B[@ATT_B='xxxx']/@ATT_C]** AltioLive allows embedded sets of square brackets [].

For example:

`/STOCKS/STOCK/HISTORY[@STOCK_ID=/STOCKS/STOCK[@MARKET='LONDON']/@STOCK_ID]` will return all the **HISTORY** elements with a value of **STOCK_ID** attribute that matches the value of the **STOCK_ID** attribute of the **STOCK** element where the **MARKET** attribute value is **LONDON**.

- You can specify **and** or **or** between conditions. You can also use parentheses to control the evaluation order.

For example: `/STOCKS/STOCK/HISTORY[(@LOW>'70.0' and @CLOSE>'70.0') and @STOCK_ID='S1']` will return the **HISTORY** elements where the **LOW** and **CLOSE** attributes are greater than **70** and the **STOCK_ID** attribute is **S1**.

String functions

- **contains(string, string)** returns TRUE if the first string argument contains the second string argument.

For example: `/STOCKS/STOCK[contains(@NAME,'Abbey')]` will return all the **STOCK** elements where the **NAME** attribute value includes the string **Abbey**.

- **starts-with(string, string)** returns TRUE if the first string argument starts with the second string argument.

For example: `/STOCKS/STOCK[starts-with(@NAME,'A')]` returns all the **HISTORY** elements where the **NAME** attribute value begins with **A**.

- **ucase(string)** converts a string to uppercase. This can make search functions more affective by removing case sensitivity.

For example: `/STOCKS/STOCK[contains(ucase(@NAME),ucase('Abbey'))]` will return **STOCK** elements where the **NAME** includes **Abbey**, irrespective of case.

- **concat(string, string)** joins two strings together to create one string.
- **number(string)** converts a string into a number.
- **date(string)** converts a string into a date.

Substitution syntax in AltioLive

In AltioLive, a dollar-brace substitution syntax is used when referencing the data of a control.

For example: `/STOCKS/STOCK/HISTORY[@STOCK_ID=${HISTORY_SELECT.value}]` returns all the **HISTORY** elements where the **STOCK_ID** matches the value selected in the **HISTORY_SELECT** control.

This is useful for filtering a list by the selections made in one or more select boxes.

Filtering data by the contents of a particular part of a control

Data can also be filtered by the contents of a particular part of a control. Generally this is a selected row in a list.

For example: `/STOCKS/STOCK[@STOCK_ID=${STOCK_LIST}/@STOCK_ID]/HISTORY` will return all the child **HISTORY** elements of the **STOCK** element where the **STOCK_ID** attribute matches the **STOCK_ID** of the **STOCK** element selected in the list **STOCK_LIST**.

This can also be written as:

`${STOCK_LIST}/HISTORY`

Where the data statement for the **STOCK_LIST** control is **/STOCKS/STOCK**

The braces are causing a substitution where the data statement from the referenced control replaces the braces and their contents. In the case of a list or a select box the data is substituted from the selected row or item.

*Please note: The **\${}** format is efficient since the **STOCK** element does not need to be found.*

Suppose a row selected in the **STOCK_LIST** control has a value of **S3** for the **STOCK_ID**. The XPath is written:

```
/STOCKS/STOCK[@STOCK_ID=${STOCK_LIST}/@STOCK_ID]/HISTORY
```

And the substitution is:

```
/STOCKS/STOCK[@STOCK_ID='S3']/HISTORY
```

The **\${windowcontext}**

This reference allows opening a new window and passing contextual data to the new window. The data statement for the new window acknowledges that the data reference is from elsewhere

For example: **\${windowcontext}/ancestor-or-self::STOCK**

The data reference from the previous window specifies the particular **STOCK** element that will provide the data for this window.

The **\${windowelement}**

Controls within a window use the **\${windowelement}** notation to reference the data statement of the window.

For example: **\${windowelement}/HISTORY**

The **\${control.propertyname}.**

A control's properties are referenced as **\${control.propertyname}**.

For example: **\${MYCONTROL.value}** would return the current value associated with a control. In particular you could use this to get the text in an editable text box, the caption of a tab control, the selection value of a checkbox or radio button.

Summary of XPath notation

The following sections provides an overview of xpath notation and is not a concise list. You should use the AltioLive online help or Designer wizards to see a full list.

LOCATION PATHS	
Optional / or //, zero or more location steps	
LOCATION STEPS	
Axis specifiers	ancestors::
	ancestors::-or-self
	parent::
	self::
Node Test	name()
	text()
	*
STRING FUNCTIONS	
Contains(string, string)	Returns TRUE if the first string argument starts with the second string argument.
starts-with(string, string)	Returns TRUE if the first string argument starts with the same string as the second string arguments
concat(string, string)	Joins two strings together to create one string
number(string)	Converts a string into a number.
date(string)	Converts a string into a date.
SUBSTITUTION SYNTAX	
\${CTRL}	References the data source of the specified control CTRL
\${CTRL}/@ATTRIB	References the value of the attribute ATTRIB within the control CTRL
\${CTRL.value}	References the value displayed in the control CTRL
\${windowcontext}	Returns an additional element to give the window extra context. This cannot be set at runtime as it is usually set when the user opens the window with the Data context parameter in the Open window action.
\${windowelement}	Returns the data element of the window (set with the Data source property of the window).
BOOLEAN OPERATORS	
=	equal
!=	If equal

<	inferior
>	superior
<=	Inferior or equal
>=	Superior or equal
LOGICAL OPERATORS	
and	Delimits conditions inside predicates
or	Delimits conditions inside predicates
not (predicate)	Returns opposite of predicate statement
ABBREVIATED SYNTAX FOR LOCATION PATHS	
(nothing)	child::
@attrib	/descendant-or-self::node()/
//	Node tree root
Element/@attrib	Value of attribute of particular element

Example of XML Data and XPath

The following examples and exercise use this sample XML data:

```
[1] <DATA>
[2]   <AAA>
[3]     <BBB>
[4]       <CCC />
[5]       <CCC NAME='john'/>
[6]       <CCC NAME='ben'/>
[7]     </BBB>
[8]   <CCC/>
[9] <BBB/>
[10] <DDD USER='john'>
[11]     <EEE/>
[12]     <CCC>
[13]       <FFF/>
[14]     </CCC>
[15]     <EEE NAME='dan'/>
[16] </DDD>
[17] <GGG>john</GGG>
[18] <EEE NAME='dan' ID='ID01'/>
[19] </AAA>
[20]</DATA>
```

(Note: this XML is available in the Sample XML Data document)

AAA	Selects the AAA element children of the context node.
*	Selects all children of the context node.
*/AAA	Selects all AAA grandchildren of the context node
AAA//BBB	Selects the BBB descendants of the AAA children of the context node.
//BBB	Selects the BBB descendants of the document root and thus selects all BBB elements in the document.
//BBB/CCC	Selects all CCC elements in the same document as the context node that have a BBB parent.
/AAA/BBB	Selects the BBB children of the top element AAA.
self::CCC	Selects the context node if it is a CCC element otherwise selects nothing.
parent::AAA	Selects the parent of the context node if it is an AAA element otherwise selects nothing.
ancestor::BBB	Selects all ancestors of the context node that are BBB elements.
ancestor-or-self::CCC	Selects all ancestors of the context node that are CCC elements and also the context node itself if it is a CCC element.
BBB[@NAME]	Selects all BBB children of the context node that has a NAME attribute.
BBB[@NAME][1]	Selects the first BBB element that is a child of the context node and has a NAME attribute.
BBB[1][@NAME]	Selects the first BBB element that is a child of the context node if it has a NAME attribute.
BBB[@NAME='john']	Selects all BBB children of the context node that has a NAME attribute with value 'john'.
BBB[@NAME='john' and @ID='5']	Selects all BBB children of the context node that has a NAME attribute with value 'john' and an ID attribute with value '5'. Up to ten conditions can be specified when creating an element match construct.
BBB[text()='john']	Selects all BBB children of the context node that has 'john' as text content.
ancestor-or-self::CCC[1]	Selects the CCC ancestor that is closest to the context node in the hierarchy or the context node itself if it is a CCC element.
//CCC[@NAME=@USER]	Selects all CCC elements that have a NAME attribute with value equal to the value of the @USER attribute of the context node.
//CCC[@NAME=text()]	Selects all CCC elements that have a NAME attribute with value equal to the text content of the context node.
//BBB[CCC/@NAME='ben']	Selects any BBB elements where one of the child CCC nodes has an attribute NAME equal to 'ben'

<code>//CCC[@NAME=parent::AAA/@USER]</code>	Selects all CCC elements that have a NAME attribute with value equal to the value of the @USER attribute of the AAA element parent of the context node.
<code>//CCC[@NAME=parent::AAA/text()]</code>	Selects all CCC elements that have a NAME attribute with value equal to the value of the text content of the AAA element parent of the context node.

For more information about Xpath

A tutorial on XPath is available at:

<http://www.zvon.org/xxl/XPathTutorial/General/examples.html>

The specification for XPath is available at: <http://www.w3.org/TR/xpath>

Exercises on XPath

First Exercise

[Appendix A](#) (see at the end of the document) shows some example of XML data that could be used in an AltioLive application (we have numbered the elements for reference).

Please note: We recommend printing **Appendix A** to work on this exercise.

Work out the elements referenced by the following XPath statements. Where a context node is included, this is the data source passed from a previous control.

Example:

Construct	Context Node	Elements Selected
FFF	[12]	Answer = line 13

Where the data referenced from the previous control is line 12 (/AAA/DDD/CCC) an additional data reference of FFF takes us to line 13.

Construct	Context node	Elements Selected
/AAA	-	
/AAA/BBB	-	
//CCC	-	
/AAA/BBB//CCC	-	
//CCC[@NAME]		
//CCC[@NAME='ben']		
//CCC[@NAME=@USER]	[10]	
//CCC[@NAME=parent::DDD/@USER]	[11]	
CCC	[10]	
*	[10]	
ancestor::DDD	[13]	
EEE[@NAME][1]	[10]	
EEE[1][@NAME]	[10]	
//GGG[text()='john']	-	
//GGG[text()=@USER]	[10]	
/AAA/BBB/CCC[@NAME=text()]	[17]	
//EEE[@NAME='dan' and @ID='ID01']	-	
*[name()='CCC']	[10]	

Second Exercise

[Appendix B](#) (see at the end of the document) shows some example of XML data that could be used in an AltioLive application (we have numbered the elements for reference).

Please note: We recommend printing **Appendix B** to work on this exercise.

Using XPath write statements to reference the following elements in the data shown **Appendix B**). Consider that new elements could be added to the data; therefore it is important to ensure that only the required element will be referenced.

Construct	XPath statement
[2]	
[10]	
[7] & [8]	
All the HISTORY elements	
All the HISTORY elements where the STOCK_ID is 'S3'	
All the HISTORY elements where the date is '31st May 2002'	
[11]	
All the stock items beginning with 'A'	
All HISTORY elements where the STOCK_ID is not 'S1'	
All HISTORY elements where the volume traded is more than 3,000,000	
All the HISTORY elements for the London market on the 1st June 2002*	
All STOCK items where the lowest HISTORY value was lower than 75 **	

* Hint: it might help to construct a query to find all the London stock elements first

** Hint: it is possible to continue specifying the location path within the predicate clause


There are various ways to reference the elements listed above.

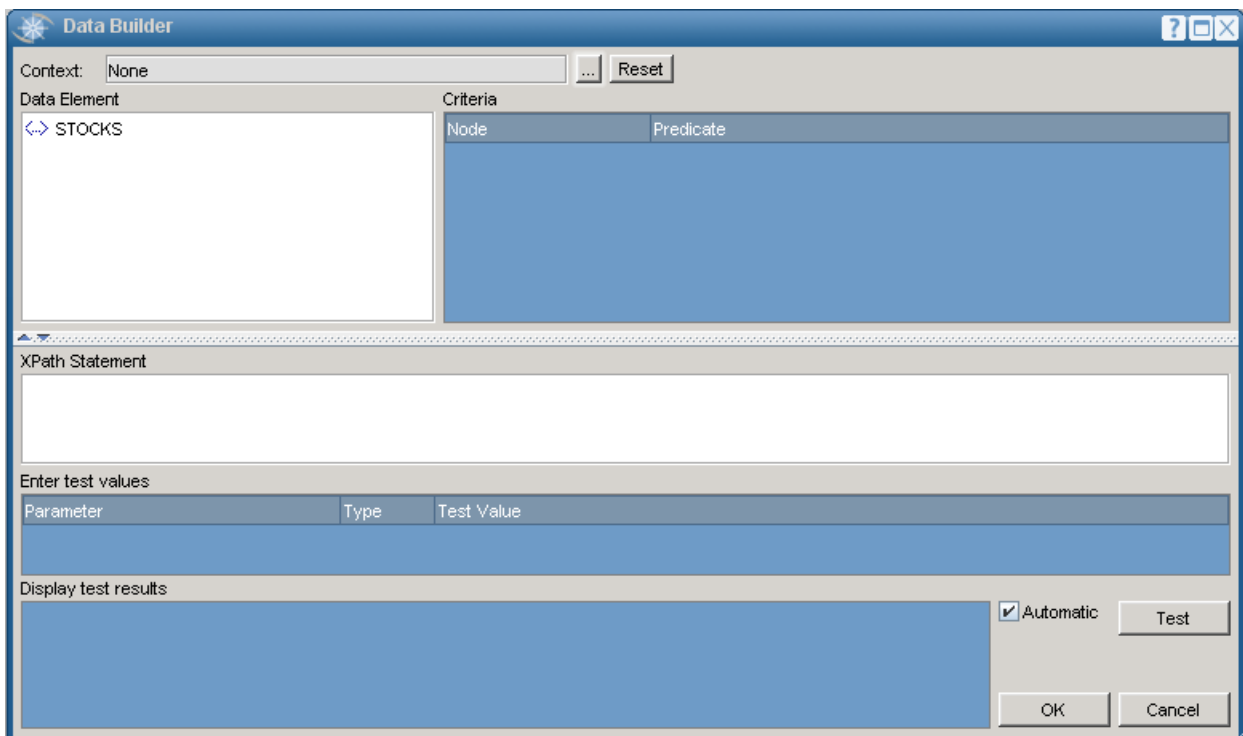
We can test XPath statements in the data builder, which is part of the Designer.

Third Exercise: Using the Data Builder to test the XPath statements

The Data Builder is a tool to allow us to manipulate XPath. Later, we will explore the Data Builder in more detail; in this exercise we will use it just to test our XPath statements.

Launching the Data Builder

1. Launch Studio Live.
2. In the **Studio explorer** window, expand the **My Projects** node.
3. Right-click on **STOCKS** project and select **Designer** in the context menu.
4. In the **View Explorer** window, expand the **MAIN_WIN** window
5. Select the **STOCK_LIST** and display the **Properties** window as explained in the previous training (See: **The First steps to build your own application**.)
6. Expand the **General** properties.
7. Click on the **Data source** property and click the ellipsis button  to launch the Data builder:



Introduction to the Data Builder

1. Expand all the nodes in the **Data Element** tree:
2. Click on each node in turn (**STOCKS**, **STOCK**, **HISTORY**) and notice how the different parts of the dialog change:
 - a. **Criteria:** shows each node involved in reaching the selected node and facilitates the entry of predicates at each step and a position. For example if you select **HISTORY** in the **Data Element tree**, the criteria shows its ancestors (**STOCKS** and **STOCK**) and the Xpath statement used to reach the **HISTORY** element.
 - b. **XPath Statement:** shows current XPath statement, composed of the nodes and predicates. You can build it by selection in the Data Element and Criteria or by typing the statement directly. This is the XPath that will appear in the **Data Source** property when **OK** is pressed.
 - c. **Enter test values:** will only contain entries if there are **#{CONTROL}** substitutions in the predicates so that sample values for controls may be entered.

- d. **Display test results:** shows the results of the current XPath statement. If Automatic is checked, then it will be updated as the XPath Statement changes. Only uncheck it if there is a very large data set and updates are slow. Then you will use the **Test** button to update the results.

The **Context** field located at the top of the window indicates the root from which the XPath starts, which may be the selected element in another control or a window. If it is set to **None** then the context is the **DATA** node.


Testing XPath statements

- Select **STOCK** from the **Data Element** tree and look at the **Display test results** list.
- In the **Criteria** box, delete the predicate for **STOCK**.
- In the **Criteria** box, enter a **STOCK** predicate of **@MARKET="LONDON"**.

The test results have been reduced to those matching the predicate (here only the markets in London).

- In the **Criteria** box, add another **STOCK** predicate: enter **and** followed by **@SECTOR="Banks"**.

The test results have been reduced further to those matching the predicate (here: only the markets in London in the Bank sector).

***Please note:** There is a **Syntax Builder** available from the ellipsis button  in the **Criteria** List which helps with building predicates. We will examine it a little later.*



- Press **Cancel** when you have finished.

Fourth Exercise: Using Controls with filtered data in the Stocks application

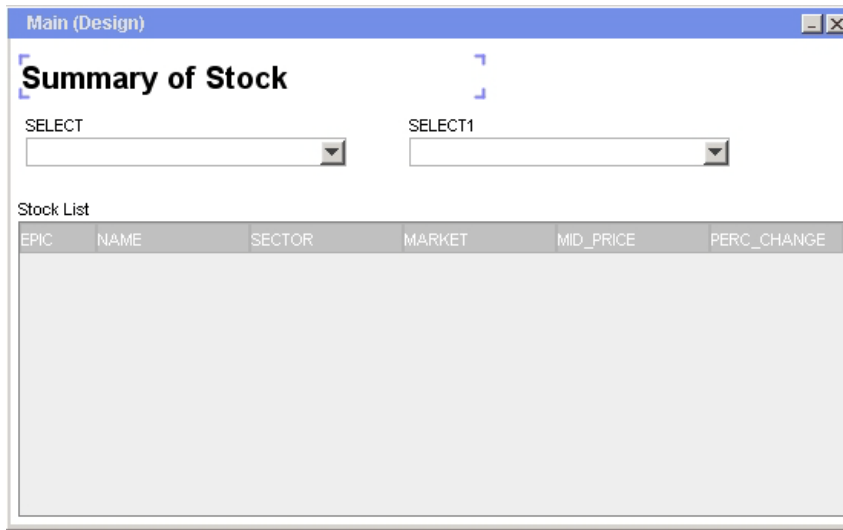
Knowing how to use XPath to filter data displayed on the screen allows providing a more manageable set of information for an end user.

In an earlier session we created a list control, called **STOCK_LIST**, to display a selection of **STOCK** attributes. Lists such as this one can become very long; this practical exercise covers how we can filter our data to particular attribute values through the use of XPath statements.

Adding two select controls to the Main window

1. From the **View Explorer** window, double-click on the **Windows** folder.
2. Double-click on **MAIN_WIN**. A preview of the **MAIN_WIN** window displays.
3. If the **Control Palette** is not displayed, click on the **Control Palette** icon .
4. Drag two select controls  from the **Control Palette** onto the window.

5. Reposition controls so that the window looks like the window shown below.



6. For the first Select control, called **SELECT**, set the properties to:

Property	Value	Comments
General		
Name	SECTOR_SELECT	
Caption	Sector	

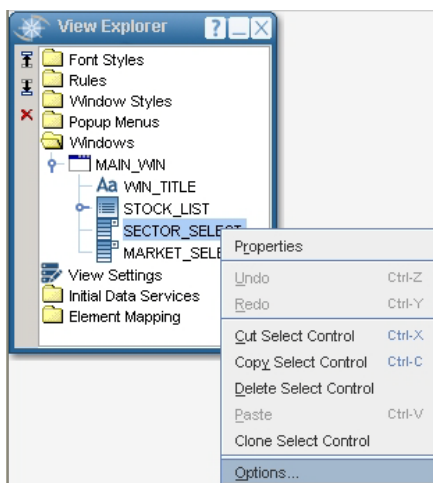
7. For the second Select control, called **SELECT1**, set the properties to:

Property	Value	Comments
General		
Name	MARKET_SELECT	
Caption	Market	


Creating values for the Sector select controls

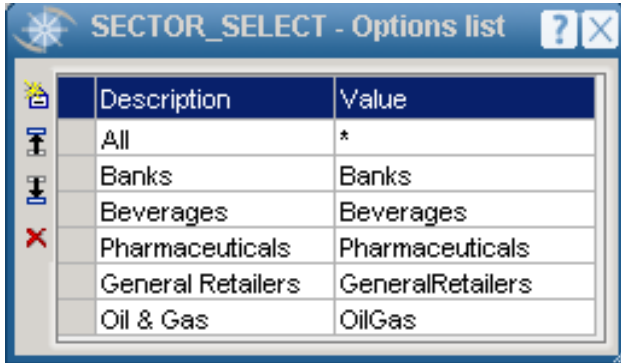
For the next part of this exercise, we are going to create the select box values by typing in the row options. Later, we will discuss how to display data-driven controls.

1. Right-click on the **SECTOR_SELECT** control (either in the **View Explorer** window or in the preview window) and select **Options** in the context menu.




To configure our own list, we use the **Options** window. We will see that each option has two attributes: a description and a value. The description is displayed in the select box and the value is used to compare against a particular data attribute.

- Use the New button  to add the options as shown below:



The **SECTOR** attribute provides a good filter for the List. The sectors within the data file are: **Banks**, **Beverages**, **Pharmaceuticals**, **General Retailers**, **Oil/Gas** and **All** (to allow all the sectors together to be shown).

Please note: Be careful that the Values are correct, as they must exactly match the **@SECTOR** attributes in the data file.


- Save the View.
- Go to the **Test** mode by clicking on the Run button  and see the option descriptions listed in the **Sector** select control:




The options will not change the list content until we amend the **Data Source** property of the **Stocks** List.

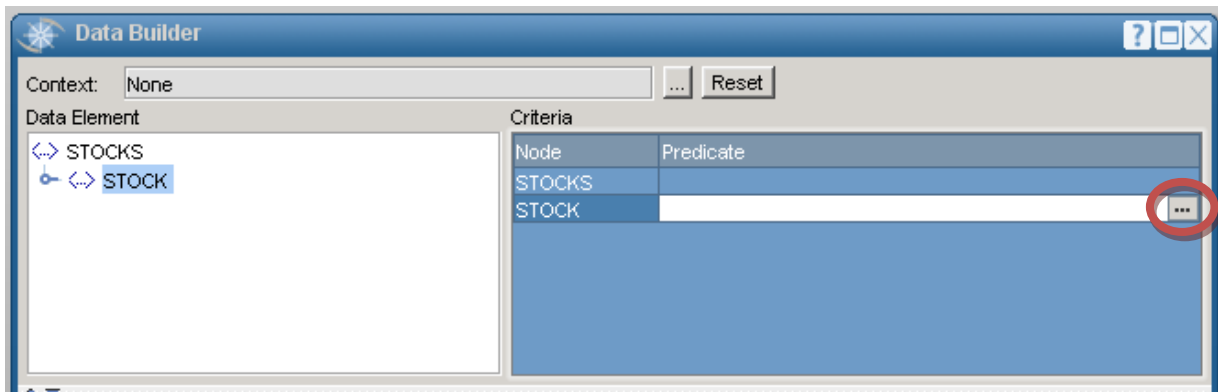
Using the Syntax Builder to link the Sector Select control to the Stock List

We will use the Data Builder and Syntax Builder to modify the data viewed in the **Stock** List control so that the Value selected in the **Sector** select control filters the data.

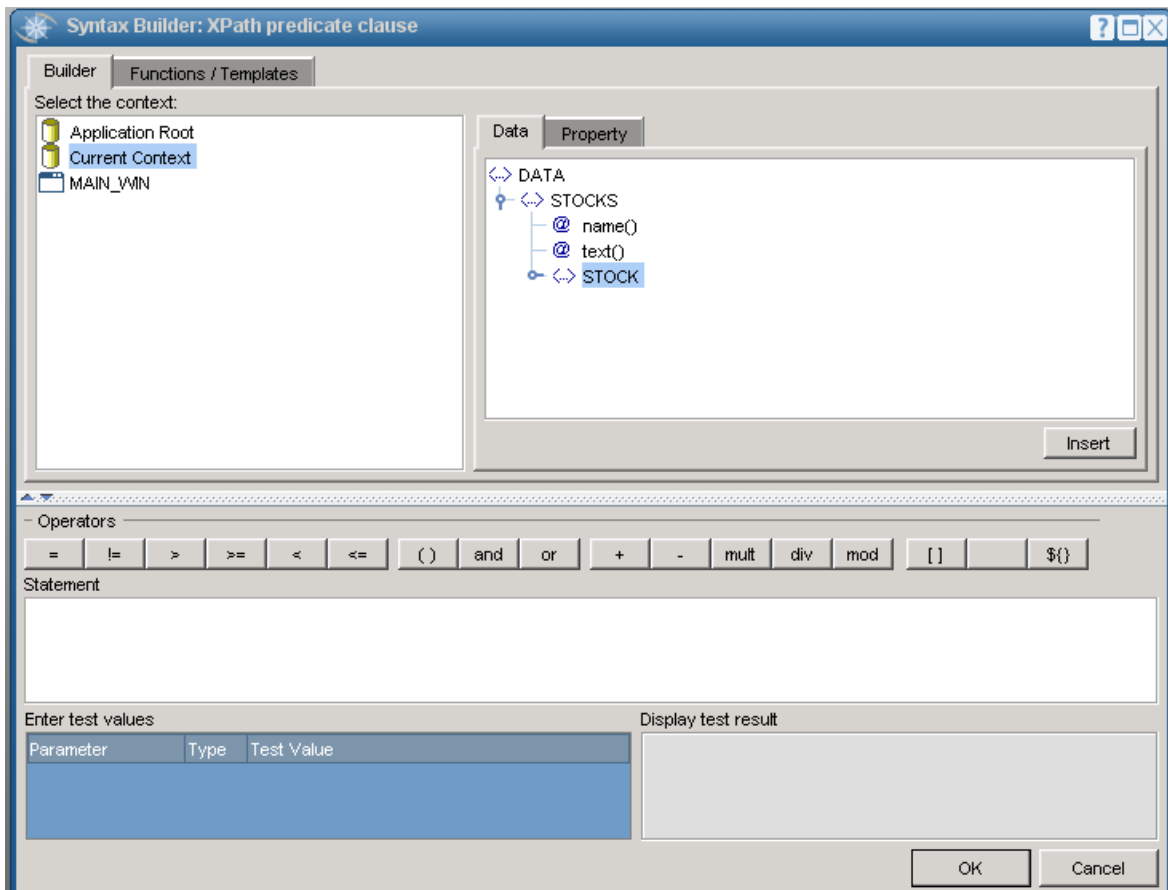
1. Open the **Properties** window for the Stocks List.
2. Expand the **General** properties and click on the **Data source** property.
3. Click the ellipsis button  to launch the Data builder.

To filter the data, we will create a Predicate clause. This will extend this data source to reference the contents of the select box

4. Click in the **STOCK Predicate** part of the **Criteria** list and click the ellipsis button  :

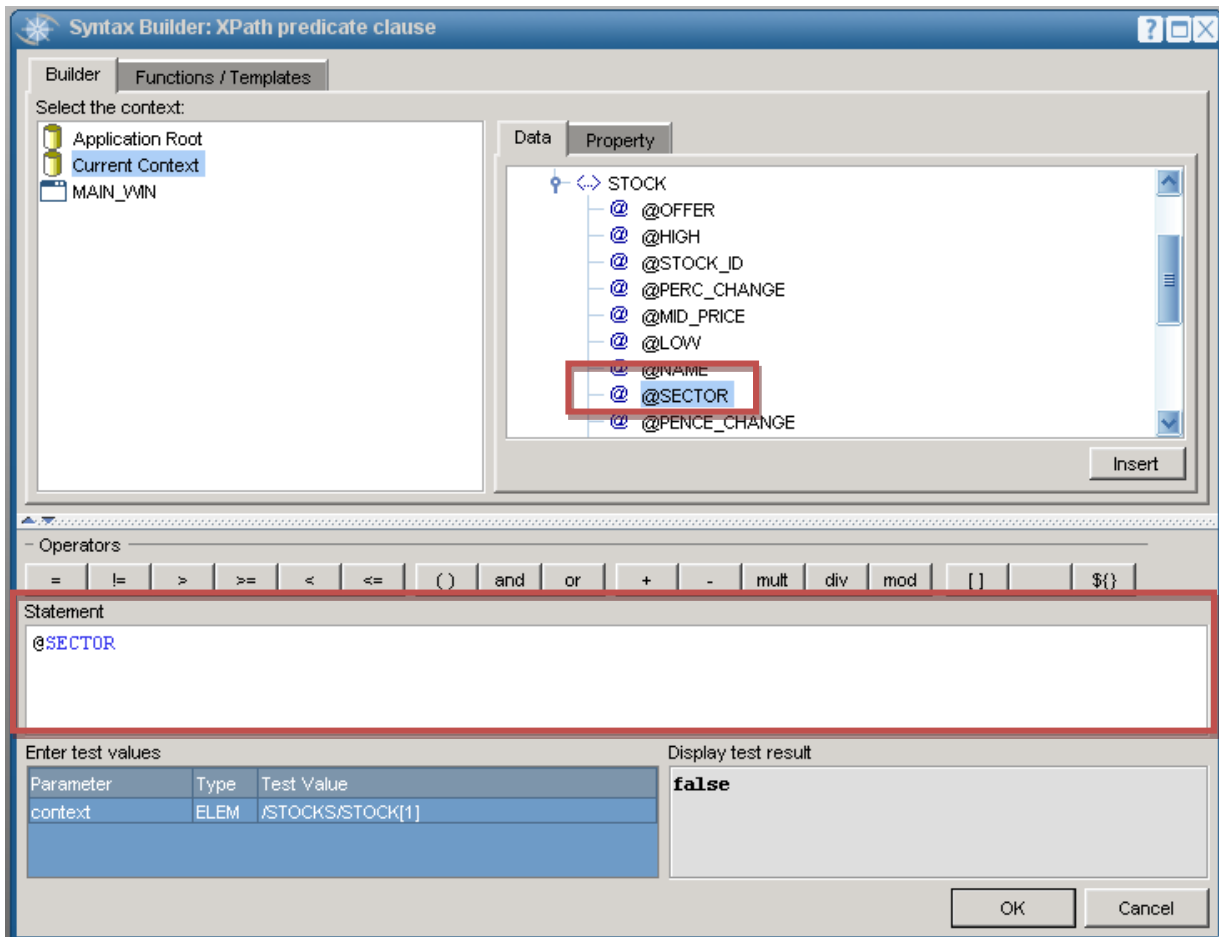


The Syntax Builder is displayed:



The Current Context is selected in the tree on the left, and this will be `/STOCKS/STOCK`. We can insert an attribute into the predicate by pressing Insert or double-clicking in the tree on the right. The inserted attribute will be specified relative to the context. For example, if we double-click on `@OFFER`, we insert `@OFFER`. But if we double-click on `name()` under the STOCKS element we insert `parent::STOCKS/name()`.

- From the **Data** tab, expand the **STOCK** node and double-click the `@SECTOR` attribute. The attribute appears in the **Statement** area




- Click the = button.
- In the **Select the context** area, expand the **MAIN_WIN** window and select the **SECTOR_SELECT** control.
- Go to the **Property** tab, scroll through the list and double-click on the **value** property.

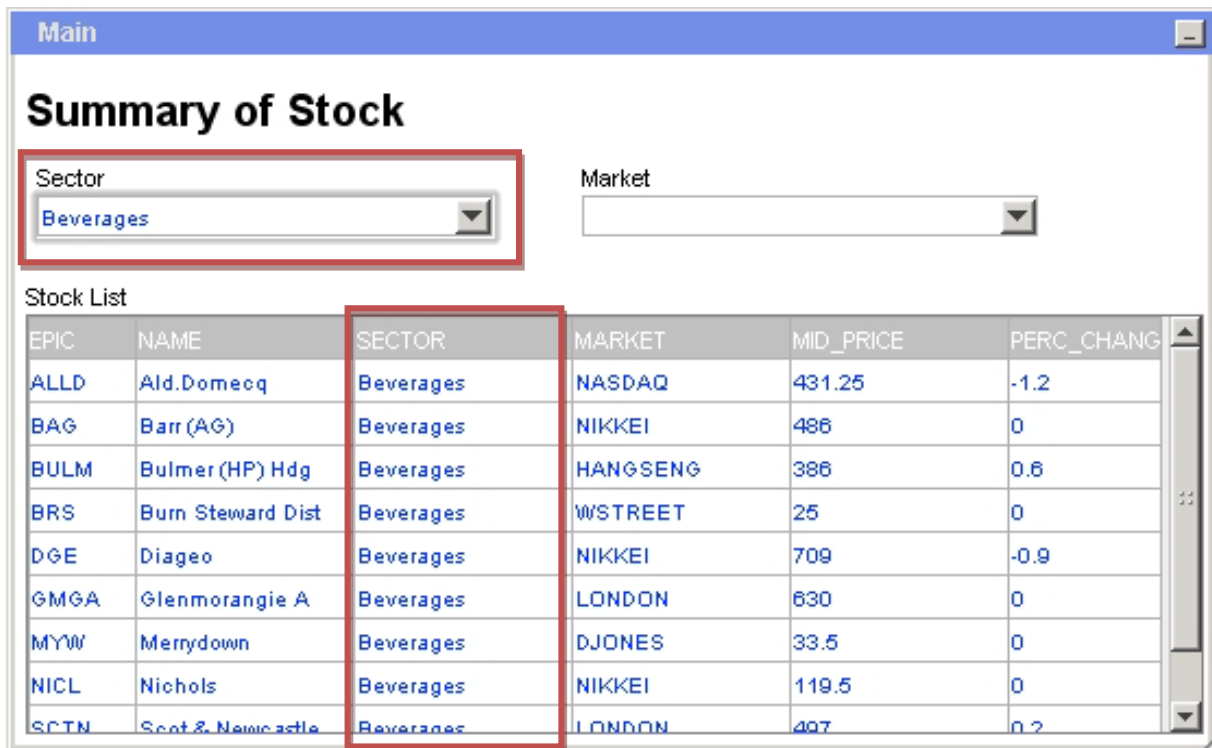
You should now see a Statement of `@SECTOR=${SECTOR_SELECT.value}`.



Please note: the **Enter test values** list now contains test values for the context passed in and **SECTOR_SELECT.value**. The **Display test result** box contains the result of evaluating the predicate with the test values provided.

- Click **OK** to close the **Syntax Builder** and accept the predicate changes.

You can now see that the Xpath statement has changed to `/STOCKS/STOCK[@SECTOR = ${SECTOR_SELECT.value}]`. The **Enter test values** list allows us to test the statement by entering values that would be selected at the control. The default of `*` gives us all elements, but another value (for example: **Banks**) gives us a smaller results set.

10. Press **OK** to close the **Data Builder** and accept the changes.
11. Test the Stocks application by clicking on the Run button  to see that the **Stocks** list is now filtered on the selected sector:

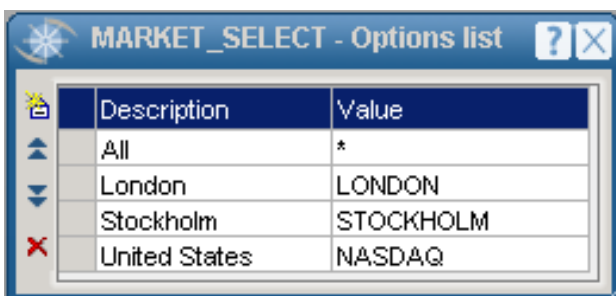



12. Stop the application by clicking on the **Stop** button .
13. Save your View by clicking on the **Save** icon .

Creating values for the Market select control and link it to the Stock List control


We will go through the same process again with the **Market** select control so that it also filters our list:

1. Right-click on the **Market** select control and select **Options** in the context menu.
2. Add the following options to the **Market** select control:



3. Open the **Properties** window for the **Stocks** list.
4. Expand the **General** properties and click on the **Data source** property.
5. Click the ellipsis button  to launch the **Data builder**.

To filter the data so that both the **Sector** and the **Market** select controls act as filters, we will change the Predicate clause. This will extend this data source to reference only the contents that match the values selected in the two select controls

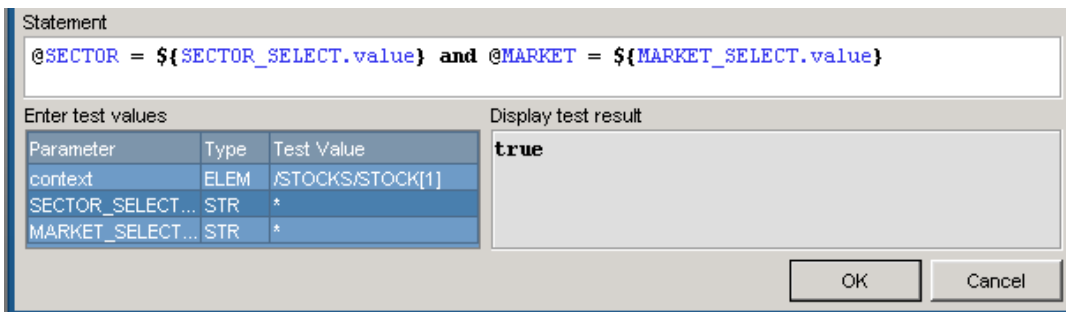
6. Click in the **STOCK Predicate** part of the **Criteria** list.
7. Click the ellipsis button  to launch the Syntax builder.

Please note: For the moment the predicate displays `@SECTOR = ${SECTOR_SELECT.value}` which allows the **Sector** list to act as a filter. We do not want to delete this predicate but change it so that both **Sector** and **Market** List act as filters.

8. Click on the **and** button and delete the two brackets.
9. Expand the **STOCK** node in tree view on the right and double-click on **@MARKET**.
10. Click on the = button.
11. In the **Select the context** area, expand the **MAIN_WIN** window and select the **MARKET_SELECT** control.
12. Go to the **Property** tab, scroll through the list and double-click on the **value** property.

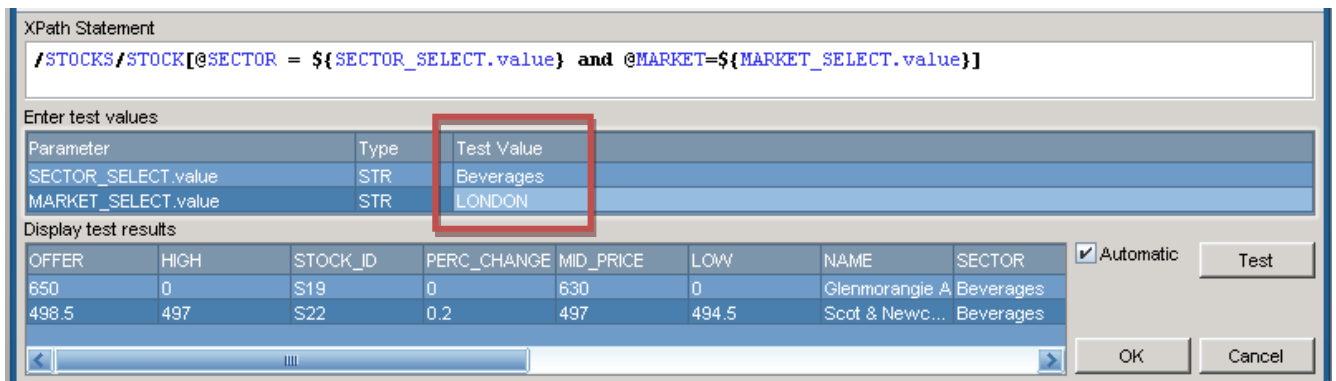
You should now see a Statement of `@SECTOR=${SECTOR_SELECT.value}` and `@MARKET=${MARKET_SELECT.value}`. The **Display test result** should display **true**.


Please note: the **Enter test values** list now contains test values for the context passed in, for the `SECTOR_SELECT.value` and for the `MARKET_SELECT.value`. The **Display test result** box contains the result of evaluating the predicate with the test values provided.

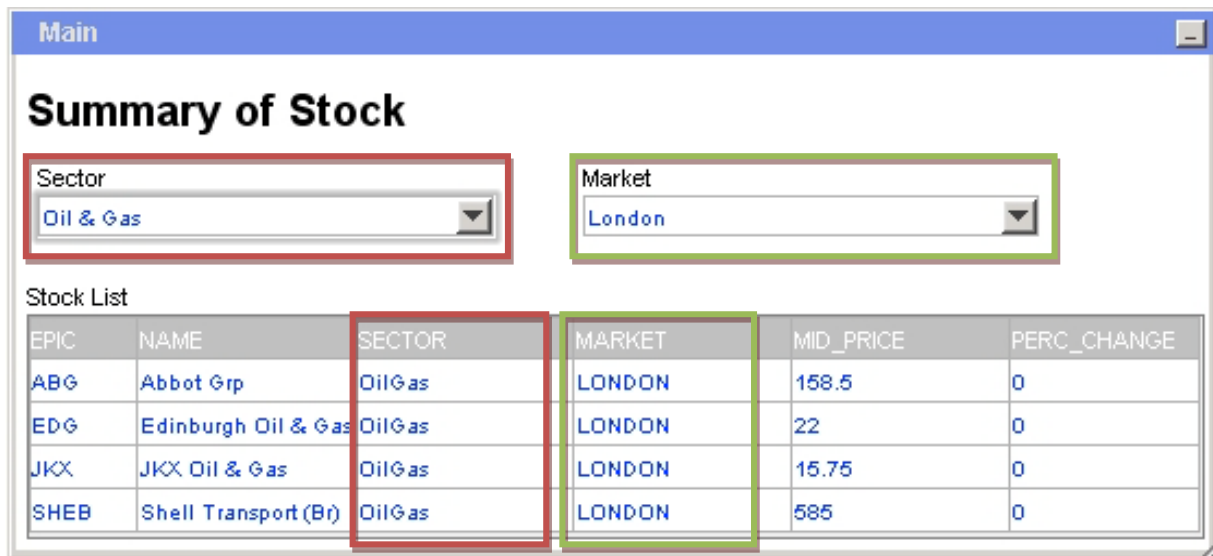




13. Click **OK** to close the Syntax Builder and accept the predicate changes.

You can now see that the Xpath statement has changed to `/STOCKS/STOCK[@SECTOR = ${SECTOR_SELECT.value} and @MARKET=${MARKET_SELECT.value}]`. The **Enter test values** list allows us to test the statement by entering values that would be selected at the control. The default of * gives us all elements, but another values (for example: **Beverages** for `SECTOR_SELECT.value` and **LONDON** for `MARKET_SELECT.value`) gives us a smaller results set:



14. Press **OK** to close the **Data Builder** and accept the changes.
15. Test the **Stocks** application by clicking on the Run button  to see that the **Stocks** list is now filtered on the selected sector and market:



16. Stop the application by clicking on the **Stop** button .
17. Save the view by clicking on the **Save** icon .

Now we have the list filtered by two different items of data.

This arrangement of controls is very useful for a main summary window where a variety of different users will want to see different sections of the data. The filtered list allows each user to tailor what they see.

Fifth Exercise: associating controls options with XML data

The values for the **Sector** and **Market** select controls were typed directly into the select options for the controls. Where controls are copied and used multiple times, the management of the data in these controls becomes harder to keep up to date. Also, the application configuration file becomes larger with repeating sections.



It is possible to create look-up tables for data repeatedly used in an application. Data used in controls, such as select controls, can be accessed from a **Look-up** table rather than adding it via the **Options** window.

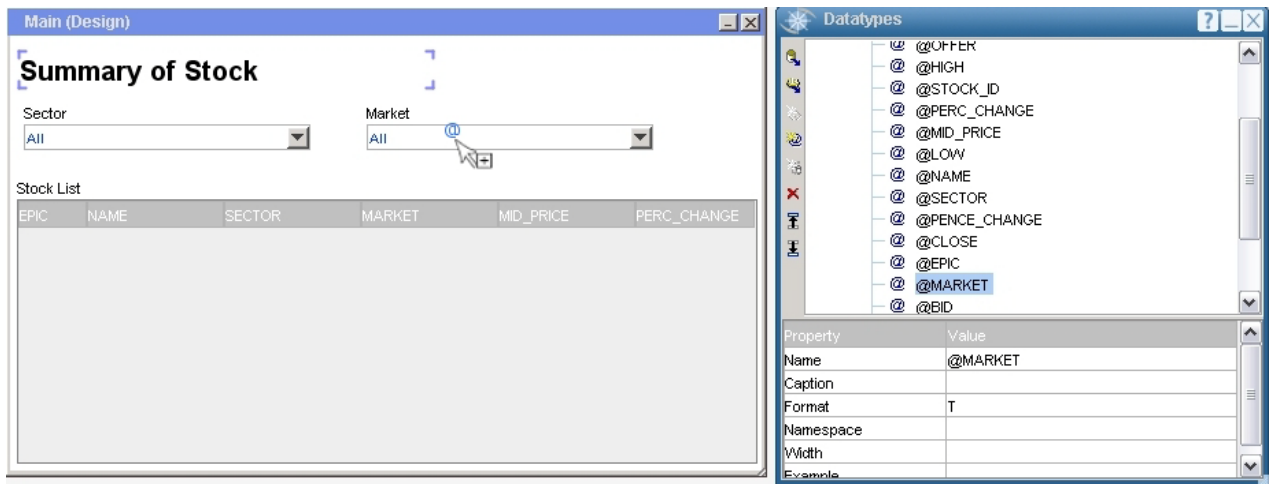
The practical exercises that follow look at how controls can be associated with XML data:




- the first exercise looks at driving a select control from existing data attributes,
- the second looks at the creation of a static look-up data source.

Associating data attributes to a select control

The select controls can have the columns populated from an XML data file. As with most controls, we can drag and drop attributes from the datatypes onto the control. Try this with the Market select control:

1. Open the **Markets Options** window by right-clicking on the **Market** select control and selecting **Options** in the context menu.
2. From the **Options** window, remove all the options except **ALL *** (**ALL** is not a value contained in the data so it must be typed-in). To do so, select the option you want to delete and click on the **Delete** button  .
3. Close the **Options** window.
4. From the **View Explorer** window, double-click on the **MAIN_WIN** window to display the preview of the window.
5. Open the **Datatypes** Window by clicking on the **Datatypes** icon  .
6. Expand the **STOCKS** and **STOCK** elements and drag the **@MARKET** attribute from the **Datatypes** window onto the **Market** select control located on the **Main** window preview.



7. Test the application by clicking on the **Run** button .
8. You will see that each market is included multiple times. In order to prevent this, select the **MARKET_SELECT** control from the **View Explorer** and expand the **General** properties in the **Properties** window.
9. Set the **Unique** property of the select control to **Y**.
10. You may also want to sort the select control alphabetically. From the **General** properties, set the **Sort by** property to **@MARKET**.
11. Test again by clicking on the **Run** button .
12. Save the view by clicking on the **Save** icon .

Next Step: Using Actions and Events in an application

AltioLive Developer Training

Appendices

Appendix A

```
[1] <DATA>
[2]   <AAA>
[3]     <BBB>
[4]       <CCC />
[5]       <CCC NAME='john'/>
[6]       <CCC NAME='ben'/>
[7]     </BBB>
[8]   <CCC/>
[9] <BBB/>
[10] <DDD USER='john'>
[11]   <EEE/>
[12]   <CCC>
[13]     <FFF/>
[14]   </CCC>
[15]   <EEE NAME='dan'/>
[16] </DDD>
[17] <GGG>john</GGG>
[18] <EEE NAME='dan' ID='ID01'/>
[19] </AAA>
[20]</DATA>
```

(Note: this XML is available in the Sample XML Data document)

Appendix B

```

[1]<STOCKS>
[2] <STOCK NAME="Abbey National" MARKET="LONDON" LOW="1170" HIGH="1192" SECTOR="Banks" STOCK_ID="S1">
[3] <HISTORY DATE="20020531" OPEN="70.7" CLOSE="77.7" HIGH="76.5" LOW="69.7" VOL="3122000.0" STOCK_ID="S1"/>
[4] <HISTORY DATE="20020601" OPEN="77.7" CLOSE="78.8" HIGH="91.2" LOW="75.0" VOL="3147091.4" STOCK_ID="S1"/>
[5] </STOCK>
[6] <STOCK NAME="Alliance & Leic" MARKET="NASDAQ" LOW="763.5" HIGH="793.5" SECTOR="Banks" STOCK_ID="S2">
[7] <HISTORY DATE="20020531" OPEN="90.1" CLOSE="93.2" HIGH="95.8" LOW="73.5" VOL="3122000.0" STOCK_ID="S2"/>
[8] <HISTORY DATE="20020601" OPEN="93.2" CLOSE="101.7" HIGH="94.4" LOW="82.4" VOL="2666057.7" STOCK_ID="S2"/>
[9] </STOCK>
[10]<STOCK NAME="Bank of Scotland" MARKET="STOCKHOLM" LOW="751" HIGH="790" SECTOR="Banks" STOCK_ID="S3">
[11] <HISTORY DATE="20020531" OPEN="51.1" CLOSE="55.1" HIGH="51.3" LOW="49.1" VOL="3122000.0" STOCK_ID="S3"/>
[12] <HISTORY DATE="20020601" OPEN="55.1" CLOSE="56.8" HIGH="66.9" LOW="45.7" VOL="2627401.2" STOCK_ID="S3"/>
[12] <HISTORY DATE="20020602" OPEN="56.8" CLOSE="61.6" HIGH="75.5" LOW="52.2" VOL="1903086.4" STOCK_ID="S3"/>
[13]</STOCK>
[14]</STOCKS>

```

Please note: this XML is also available in the Sample XML Data document)

Document Information

KEYWORDS: XML, XPATH, DATA BUILDER, SYNTAX BUILDER

Integra SP – Altio

Telephone: +44 (0) 20 8528 1045

Internet: www.altio.com

Copyright © 2010 Integra SP

Copyright in this document is vested in Integra SP. The contents of the document (wholly or in part) must not be reproduced, distributed, used or disclosed without the prior written permission of Integra SP.

Integra recognizes the trademarks or registered trademarks of any third party product or company name referenced in this document at the time of its publication.