# Lesson 6: Using XML Queries

In the previous lesson you learned how to issue commands and how to retrieve the results of selections using the `UniSelectList` class, whether for regular enquiry statements or from alternate key lookups.

Using a select list to pass a list of record keys and reading the set of records for display is a simple and well-tried technique, but one that is normally associated with server based operations. The problem is the number of round trips required to read the data, especially if it must come from more than one file. Better by far to build all the data on the server and send it back in a single call.

There are various techniques by which you can approach this, one of the simpler being to leverage the XML support that is now offered by the three enquiry languages supported on the UniVerse and UniData platforms.

RetrieVe, UniQuery and UniSQL all have the facility to produce output in XML format. XML is a comfortable fit to the MultiValue data model, and you can read about the range of support available and the features for generating and formatting XML output in the previous volumes in this series.

Support for XML has been a strong element of IBM's stewardship of the products. It is also a keystone of the .NET framework that offers a range of opportunities for both generating and consuming XML formatted data. XML may not be the most efficient way of structuring data – it is highly verbose – but the availability of tools on both ends of the client/server divide makes it a perfect choice for situations in which you need to pass quantities of data.

The various techniques for generating XML out of the database were covered in detail in the previous volume, *Self Paced Training - UniVerse Server Developer*. For this course, we will be concentrating on processing the results of those techniques and so we will restrict ourselves to using a simple and easy technique for generating XML from either UniVerse or UniData: using the enquiry languages and adding the keywords `TO XML` and appropriate modifiers to an enquiry statement[9], as in:

```
LIST BOOK_TITLES TOXML ELEMENTS
```

or

[9] You will need to be running at least version 10 of UniVerse and version 7 of UniData.

```
SELECT * FROM BOOK_TITLES TOXML ELEMENTS;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<BOOK_TITLES>
  <_ID>10</_ID>
  <SHORT_TITLE>Hancock a Comedy Genius (BBC Radio Collection)</SHORT_TITLE>
  <SHORT_AUTHOR>CAST</SHORT_AUTHOR>
</BOOK_TITLES>
<BOOK_TITLES>
  <_ID>11</_ID>
  <SHORT_TITLE>I'm Sorry I Haven't a Clue: Vol 8 (BBC Radio Collection)</SHORT_TITLE>
  <SHORT_AUTHOR>CAST</SHORT_AUTHOR>
</BOOK_TITLES>
<BOOK_TITLES>
  <_ID>12</_ID>
  <SHORT_TITLE>Friends, Lovers, Chocolate</SHORT_TITLE>
```

Building XML is a good way to generate data directly from the database, but of course you then need to do something with those results. In this lesson we will look at two ways in which you can do this: using the `UniXML` class or using `LINQ`.

## The UniXML class

The `UniXML` class is something of a utility class. It is designed for a single purpose only: to run an XML based enquiry statement and to capture the results as a `DataSet`.

**The DataSet**

The `DataSet` is the foundation of the current Microsoft data access model. Microsoft has adopted a range of strategies for holding and navigating tabular data over the years, of which the `DataSet` is probably the most fully developed to date.

A `DataSet` is an in-memory representation of a database, containing strongly-typed tables and relations and offering a (limited) ability to create views. `DataSets` are independent of any physical data source, and can be designed and populated entirely programmatically. They can be filled by ADO.Net providers, which in turn can interact with data sources using older technologies such as OleDB and ODBC. We will be looking at these in detail in the third Chapter of this course, when we turn our attention to the mainstream relationally-founded routes for accessing UniVerse and UniData.

`DataSets` were also the first target for data binding in .NET, though this has now extended into object binding, which will also be covered later in this course.

For regular UniVerse and UniData client/server programming, using a `Dataset` is not normally a necessity when working with data, especially for updates where the `UniFile` and the ability to call subroutines provide faster and more native routes. Whilst offering a depth of functionality, when it comes to real world solutions a `DataSet` can often be a cumbersome tool particularly when it comes to areas of conflict resolution and validation.

However, using a `DataSet` when derived from a `UniXML` instance is still a useful route for quickly and easily getting XML results gained from UniVerse or UniData enquiries into a meaningful and navigable format. And the best thing about it is that the `UniXML` class does this wholly automatically.

The `UniXML` class, you should not now be surprised to learn, is created by the `UniSession` through a `CreateUniXML` method:

```
MyUniXML = MyUniSession.CreateUniXML
```

The `UniXML` class runs a TCL or ECL command and handles results returned in an XML format. There is a major restriction: the only commands it will accept are pure enquiry commands using the regular enquiry verbs – LIST, SORT or (SQL) SELECT. You cannot, for example, give it the name of a paragraph or a program, which effectively restricts it to being used for single-stage enquiry statements.

To run the enquiry statement, you must call the `GenerateXML` method. This accepts two arguments: the command to run and any additional options to control the XML produced. For those who have followed the previous course, this is almost identical to the `XMLExecute()` function available in UniVerse and UniData Basic.

```
MyUniXml.GenerateXml(myCommand)
```

**Hint**

You can omit the `TOXML` clause on the enquiry command: the `UniXML` class will add that automatically.

Once the XML has been generated by the enquiry command it is available through the `XMLString` and `XSDString` properties that provide access to the document and schema respectively. More usefully however it can be parsed into a `DataSet` by calling the `GetDataSet` method.

```
System.Data.DataSet myDataSet = MyUniXml.GetDataSet();
```

## Exercise 2.6.1: Using the UniXML class for Simple Data

In this exercise, you will create a new lookup on the BOOK_TITLES file, this time using the `UniXML` class to do the work.

**Do This:**

Add a new form to your application named `SelectTitleXMLForm`.

This will present a text box named `txtTitle` through which the operator can enter part of a short title, and a Find button. Pass the `UniSession` instance across and add this to your main menus in the same manner as the other forms in your project.

Add a `DataGridView` control to the form, but do not add any columns to the grid.

The form will look similar to the `SelectTitleForm` you should have created in the last Lab exercise of the previous lesson.


Now code up the Find button to perform a partial string search on the BOOK_TITLES SHORT_TITLE using the text entered. This will use a `UniXML` instance to run an enquiry command and return a `DataSet`.

*Using VB.Net:*

```
Private Sub cmdFind_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles cmdFind.Click
   Dim unixml As UniXML
   Dim ds As System.Data.DataSet
   unixml = sess.CreateUniXML()
   Dim cmd As String = String.Format("SORT BOOK_TITLES WITH
→   SHORT_TITLE LIKE ""...{0}..."""", txtTitle.Text)
    Try
        unixml.GenerateXML(cmd)
    Catch ex As Exception
        MessageBox.Show(ex.Message)
        Return
    End Try
    Try
        ds = unixml.GetDataSet()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
        Return
    End Try
End Sub
```

*Using C#:*

```
private void cmdFind_Click(object sender, EventArgs e) {
   UniXML uniXML = sess.CreateUniXML();
   System.Data.DataSet ds;
   String cmd = String.Format("SORT BOOK_TITLES WITH
→   SHORT_TITLE LIKE \"...{0}...\"", txtTitle.Text);
    try {
        uniXML.GenerateXML(cmd);
    } catch (Exception ex) {
        MessageBox.Show(ex.Message);
        return;
    }
    try{
        ds = uniXML.GetDataSet();
    } catch (Exception ex) {
        MessageBox.Show(ex.Message);
        return;
    }
}
```

Now for the final step you can let the .NET framework do some of the work. The `DataGridView` control is designed primarily to act as a bound control, and if assigned a table from a `DataSet` as its `DataSource`, it will automatically generate the necessary columns and rows to display the table content.

So to view the results of your XML enquiry, you can simply assign the first and only table in the `DataSet` to the `DataSource` property of the `DataGridView` as below:

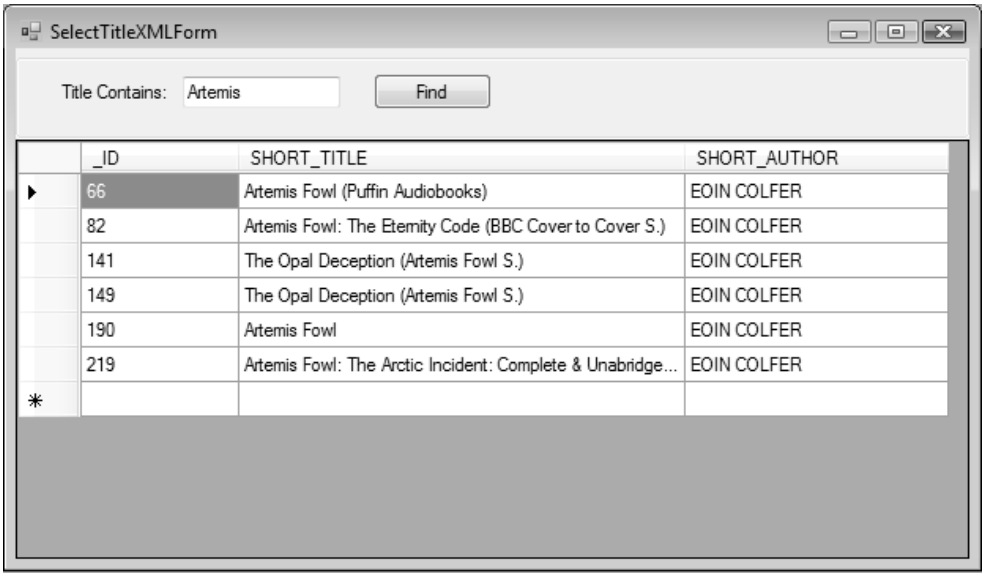
*Using VB.Net:*

```
dataGridView1.DataSource = ds.Tables(0)
```

*Using C#:*

```
dataGridView1.DataSource = ds.Tables[0];
```

Run the project and enter part of a title, e.g. Artemis. You should see a selection of books by Eoin Colfer.




As you can see, using the `UniXML` to present data makes for a very simple way to provide selections: I typically use `UniXML` to code up quick lookup forms for applications in situations where I am not too concerned with performance. But the `UniXML` class is also smarter than that.

In the example above, you referenced the results as `myDataSet.Tables[0]`. A `DataSet` can contain multiple tables following a standard relational schema, and can define relations between them that include parent/child relations. When listing multivalued data from the XML query, the `UniXML` class can set up header and detail relations between tables in the `DataSet`.

When you perform an XML query against multivalued data, the values will be returned in the XML document as nested entities. The fact that XML can handle nested entities, which is a key part of the MultiValue model, is one reason why UniVerse and UniData developers prefer XML over other data representations and why XML is such an important addition to the UniVerse and UniData products.

Where these fields form an association, the association name becomes the parent element of the nested row so preserving the natural association between the values:

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<BOOK_SALES>
  <_ID>13661*55800*1</_ID>
  <SALE_DATE>26 MAY 2005</SALE_DATE>
  <SURNAME>MAGGS</SURNAME>
  <FORENAME>CARL</FORENAME>
  <SALE_ITEMS-MV>
    <TITLE_ID>49</TITLE_ID>
    <QTY>1</QTY>
    <PRICE>9.99</PRICE>
    <GIFT>1</GIFT>
    <TITLE_NAME>Jingo</TITLE_NAME>
    <AUTHOR_NAME>Terry Pratchett</AUTHOR_NAME>
    <LINE_TOTAL>9.99</LINE_TOTAL>
  </SALE_ITEMS-MV>
  <SALE_ITEMS-MV>
    <TITLE_ID>47</TITLE_ID>
    <QTY>1</QTY>
    <PRICE>49.99</PRICE>
    <GIFT>0</GIFT>
    <TITLE_NAME>Harry Potter and the Goblet of Fire (Book 4)</TITLE_NAME>
    <AUTHOR_NAME>J.K. Rowling</AUTHOR_NAME>
    <LINE_TOTAL>49.99</LINE_TOTAL>
  </SALE_ITEMS-MV>
  <SALE_ITEMS-MV>
    <TITLE_ID>11</TITLE_ID>
    <QTY>1</QTY>
    <PRICE>12.99</PRICE>
    <GIFT>0</GIFT>
    <TITLE_NAME>I'm Sorry I Haven't a Clue: Vol 8 (BBC Radio Collection)</TITLE_NAME>
    <AUTHOR_NAME>Cast</AUTHOR_NAME>
    <LINE_TOTAL>12.99</LINE_TOTAL>
  </SALE_ITEMS-MV>
</BOOK_SALES>
</ROOT>
```

The `UniXML` class will preserve this relation when it comes to parsing XML data that contains nested entities, whether standalone or as part of an association, into a `DataSet`. As well as creating a single table containing the single-valued rows, it will create child tables for each association and for each of the non-associated fields and build the relationships between them.

## Exercise 2.6.2: UniXML with MultiValued Data

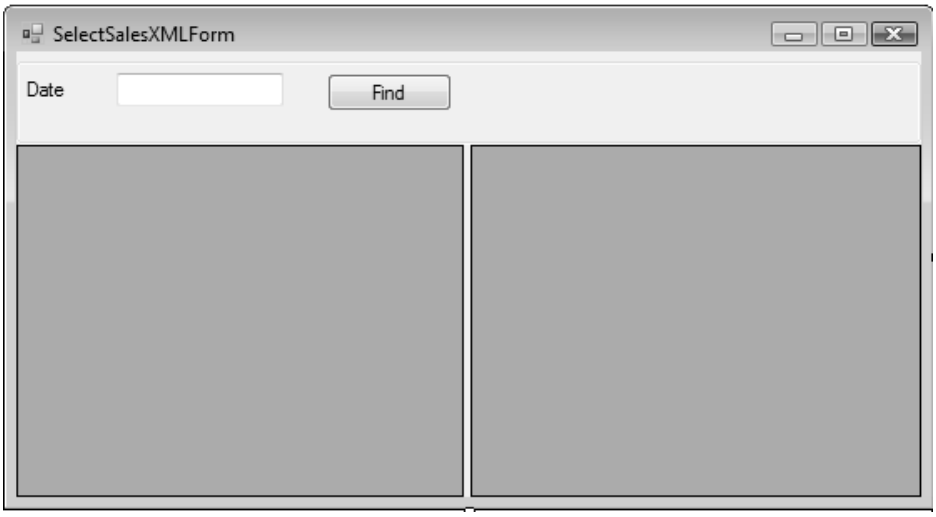
In this exercise, you will discover how `UniXML` can be used to generate a `DataSet` from multivalued data and how that can be presented easily on a form.

So far in this project we have largely ignored the main file of the demonstration database: the BOOK_SALES file. As the name suggests this file holds sales orders. The file breaks naturally into a header section recording the customer details and the order date, and a set of associated multivalued fields that make up the order detail lines.

You are going to build a form to display a list of sales orders for a specific date. In addition to the header details, as each order is highlighted the detail of that order will be displayed.

**Do This:**

- Add a new form to your project called `SelectSalesXMLForm`.
- Pass the UniSession instance and add this to your menus
- Add a text box named txtDate and a search button named cmdFind to the top. Then add two `DataGridView` controls side by side (you can add a `SplitContainer` to your form to make this easier), but once again do not add any columns. The resulting form should look like the one below:



In the Click event for the Find button, you will once again create a simple enquiry to return the sales order details for the selected date using a `UniXML` class:

*Using VB.net:*

```
Private Sub cmdFind_Click(ByVal sender As System.Object, ByVal
   e As System.EventArgs) Handles cmdFind.Click
   Dim unixml As UniXML = sess.CreateUniXML()
   Dim ds As System.Data.DataSet
   Dim cmd As String = String.Format("SORT BOOK_SALES WITH
     SALE_DATE = ""{0}""", txtDate.Text)
   Try
        unixml.GenerateXML(cmd)
    Catch ex As Exception
        MessageBox.Show(ex.Message)
        Return
    End Try
    Try
        ds = unixml.GetDataSet()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
        Return
    End Try
End Sub
```

*Using C#:*

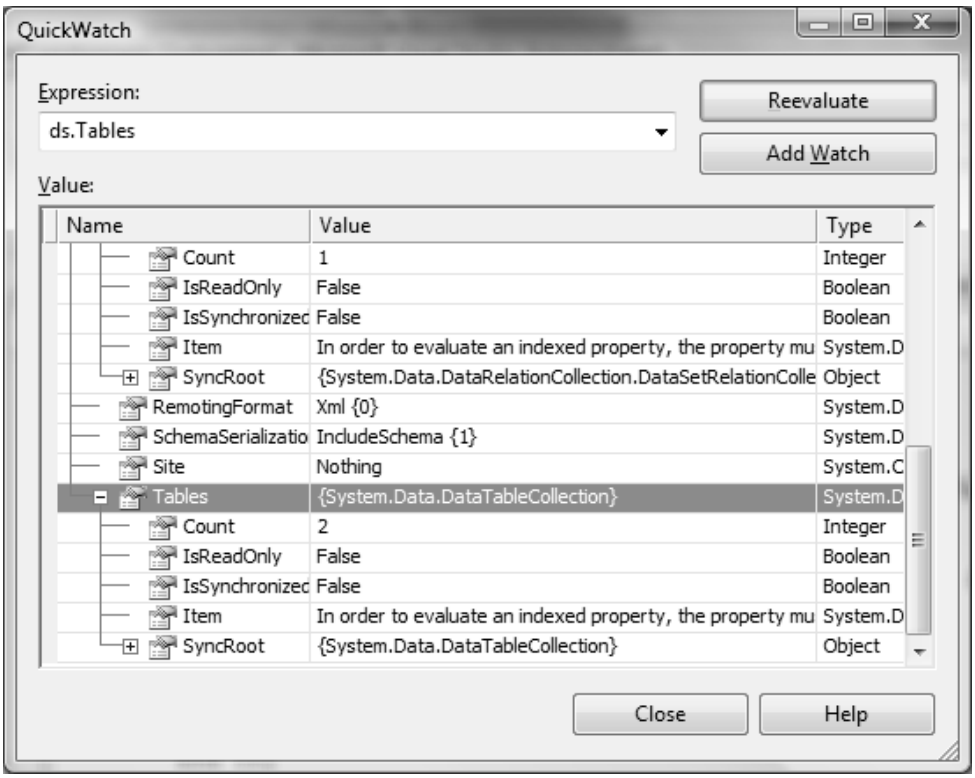
```
private void cmdFind_Click(object sender, EventArgs e) {
   UniXML unixml = sess.CreateUniXML();
   System.Data.DataSet ds;
   String cmd =
     String.Format("SORT BOOK_SALES WITH SALE_DATE = \"{0}\"",
         txtDate.Text);

    try {
        unixml.GenerateXML(cmd);
    } catch (Exception ex) {
        MessageBox.Show(ex.Message);
        return;
    }
    try {
        ds = unixml.GetDataSet();
    } catch (Exception ex) {
        MessageBox.Show(ex.Message);
        return;
    }
}
```

Now before you go any further it will be useful to look at the `DataSet` that will be produced by this enquiry. Set a break point on the line with the call to the `GetDataSet` method, and run the project.

Enter a date for the sales orders, for example `26 MAY 2005`, and click the Find button. This will run the enquiry and stop at the break point on the line fetching the resulting `DataSet`.

Step over that line and highlight the `DataSet` variable in your code window. From here you should be able to select `Quick Watch` from the context menu by clicking with the right mouse button:




You will notice that the `DataSet` now contains two tables.

In the Expression box at the top of the QuickWatch form, enter:

```
ds.Tables[0]
```

This will present the details for the main table, BOOK_SALES.

Next enter in the expression box:

```
ds.Tables[1]
```

This will present the details for the second table named `SALE_ITEMS-MV`. You may recognize this as the name of the association element in the XML listing above.

The population of the `DataSet` does not stop there. In addition to the two tables, the `UniXML` class has created a relationship between them. In the Expression window, enter:

```
ds.Tables[0].ChildRelations[0]
```

This defines the parent/child relationship, named `BOOK_SALES_SALE_ITEMS-MV`, between the main BOOK_SALES table and the SALES_ITEMS-MV child table.

Now you have seen the structure of the `DataSet` produced by the `UniXML` class from your enquiry, it is time to complete your form.

**Do This:**

Stop the project and return to the code for the Find button Click event. From here you can assign the `DataSource` property of the first `DataGridView` that will present the sales order header details using the first table generated:

Using VB.net:

```
dataGridView1.DataSource = ds.Tables(0)
```

Using C#:

```
dataGridView1.DataSource = ds.Tables[0];
```

Now the second `DataGridView` can be linked to the second table in the `DataSet`. However if you simply assign the `DataSource` property to the second table, this will display all of the detail lines but will not be coordinated with the first table. What we need is to link the two grids so that the detail lines are shown only for the sales order highlighted in the first grid.

They way you handle this may seem a little syntactically bizarre. For this you must also link the second grid to the *first* table in the `DataSet`: but then set the