CSIS 4135

Object Data Source
More Data Controls
Improving Data Base Access
LINQ to SQL
ADO .NET and XML

Databinding

Functionality is built into user interface controls that allows them to display data returned from a database

- `GridView`, for example

Data controls were built especially for databinding

- `SqlDataSource`, for example

Data Controls

- We've seen `GridView` for displaying returned results of a SELECT statement
- `DetailsView` will show the details of a single record
  - Enable paging to allow moving from one record to the next
- `FormView` will allow editing, inserting, and deleting individual records

Data Controls

Connect a `GridView` to a `DetailsView` by implementing its `SelectedIndexChanged` event handler to modify the `DetailsView`'s `PageIndex` to match the selected index in the grid

Data Controls

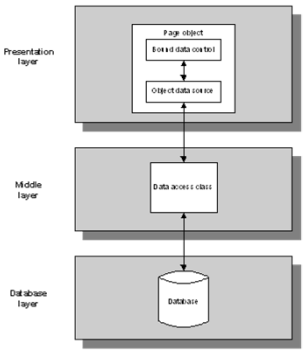
Use a `FormView` by implementing its `SelectedIndexChanged` event handler to modify the `DetailsView`'s `PageIndex` to match the selected index in the grid.

The 3-layer Architecture

- **Presentation layer:** Pages that define the user interface of the application
- **Middle layer:** Manages data access and business rules
- **Database layer:** The database

The `SQLDataSource` puts too much dependency on SQL code in the presentation layer.

The 3-layer architecture in ASP.NET



Data Access in the Middle Layer

- One approach is to use a `DataSet`
- Add a `DataSet` item to the web site
  - Drag any tables needed to the `DataSet`
  - Then configure as needed
  - This creates an XML Schema Definition (XSD) file

ObjectDataSource

- `ObjectDataSource` objects can be configured to use the `DataSet` to provide the connection to the UI controls
- Configure the methods to be used to get tables needed by the UI controls

Using Stored Procedures

Generally better than coding SQL statements in C#

- More efficient
- More secure
- Can be written by DBMS experts and just called from the code-behind

Stored Procedures

- One or more SQL commands stored in a database as an executable object
- Can be called from within client application code
- Advantages
  - Better control of access to tables
  - Reduced network traffic
  - Faster execution

Example

```
create procedure quote_authors as
  select author from quotes
return
```

`quote_authors`

Create the procedure

Query

Stored Procedure with Parameter

```
create procedure quotes_by_author
( @Auth NVarchar(50) )
as
 select quotation, creation_date
 from quotes
  where author = @Auth
return
```

Procedure to Add a Quote

```
create procedure add_quote
   @author nvarchar(50),
   @cat_ID int,
   @quotation nvarchar(250)
as
  insert into quotes
   (Quotation, Author, Category_ID, Creation_date)
  values
   (@quotation, @author, @cat_ID, getdate())
```

Code Behind

```
SqlConnection Conn = new
    SqlConnection(ConfigurationSettings.AppSettings[
                                        "connectString"]);
SqlCommand c = new SqlCommand("add_quote", Conn);
c.CommandType = CommandType.StoredProcedure;
c.Parameters.Add("@author", TextBox1.Text);
c.Parameters.Add("@category", Convert.ToInt32(TextBox2.Text));
c.Parameters.Add("@quotation", TextBox3.Text);
Conn.Open();
c.ExecuteNonQuery();
Conn.Close();
```

LINQ

Language Integrated Query

- Enables performing complex query operations on any enumerable object
- In particular, can be used to query a database

Some C# keywords for working with LINQ

Keyword	Description
from	Identifies the source of data for the query.
where	Provides a condition that specifies which elements are retrieved from the data source.
orderby	Indicates how the elements that are returned by the query are sorted.
select	Specifies the content of the returned elements.
group	Groups the elements by one or more data items and lets you perform aggregate functions on the groups.
join	Combines data from two data sources.

Using LINQ to query a database

1. Create an object model that maps to the data model of the database.
2. Define the query.
3. Execute the query.

Using LINQ to query the Products table in the Halloween database

```
HalloweenDataContext db =
          new HalloweenDataContext();
var products = from product in db.Products
               where product.OnHand > 0
               orderby product.ProductID
               select new
               {
                   product.ProductID,
                   product.Name,
                   product.UnitPrice,
                   product.OnHand
};
GridView1.DataSource = products;
GridView1.DataBind();
```

Binding to a LinqDataSource control


```
<asp:DropDownList ID="DropDownList1" runat="server"
    DataSourceID="LinqDataSource1"
DataTextField="LongName"
    DataValueField="CategoryID" Width="130px"
AutoPostBack="True">
</asp:DropDownList>
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
    ContextTypeName="HalloweenDataContext"
    OrderBy="LongName"
    Select="new (CategoryID, LongName)"
    TableName="Categories">
</asp:LinqDataSource>
```

XML

- XML = eXtensible Markup Language
- Like HTML, it uses tags to markup data, but
  - Was designed for representing structured data
  - Represents data hierarchically (in a tree)

XML and Structured Data

- Pre-XML representation of data:

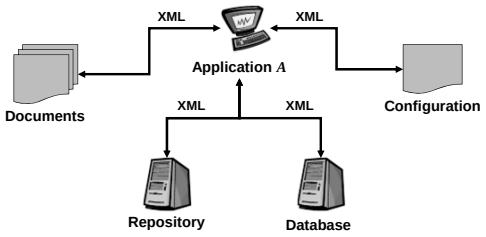
```
"PO-1234","CUST001","X9876","5","14.98"
```

- XML representation of the same data:

```
<PURCHASE_ORDER>
      <PO_NUM> PO-1234 </PO_NUM>
      <CUST_ID> CUST001 </CUST_ID>
      <ITEM_NUM> X9876 </ITEM_NUM>
      <QUANTITY> 5 </QUANTITY>
      <PRICE> 14.98 </PRICE>
</PURCHASE_ORDER>
```

XML Overview

- XML is a "use everywhere" data specification



XML in .NET

- XML use is ubiquitous throughout .NET
- ADO.NET
  - Provides conversion between DataSets and XML
  - DataSets are serialized as XML
- ASP.NET
  - Application information stored in XML-based configuration files
- Web Services
  - Based on XML standards: SOAP, WSDL, UDDI, …

XML Overview

- XML itself is fairly simple
- Most of the learning curve is for all of the related technologies

XML Schemas: Overview

- DTD (Document Type Definitions)
  - Not written in XML
  - No support for data types, value constraints, etc.
- XSD (XML Schema Definition)
  - Written in XML
  - Supports data types
  - Current standard recommended by W3C

XML Schemas: Purpose

- Define the "rules" (grammar) of a document
  - Structure
  - Data types, value bounds, etc.
- A XML document that conforms to a schema is said to be valid
- Define which elements are present and in what order
- Define the structural relationships of elements

XML Schemas: DTD Example

- XML document:

```
<BOOK>
   <TITLE>All About XML</TITLE>
   <AUTHOR>X. Emil Developer</AUTHOR>
</BOOK>
```

- DTD schema:

```
<!DOCTYPE BOOK    [
<!ELEMENT BOOK    (TITLE, AUTHOR+) >
<!ELEMENT TITLE   (#PCDATA) >
<!ELEMENT AUTHOR  (#PCDATA) >
]>
```

Schemas: XSD Example

```
<xsd:schema id="NewDataSet" targetNamespace="http://tempuri.org/schema1.xsd"
   xmlns="http://tempuri.org/schema1.xsd"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="book">
    <xsd:complexType content="elementOnly">
      <xsd:all>
        <xsd:element name="title" minOccurs="1" type="xsd:string"/>
        <xsd:element name="author" minOccurs="1" type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Catalog" msdata:IsDataSet="True">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element ref="book"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

DataSet to XML

Use the `WriteXml` method to send a `DataSet` to an output stream as XML

```
// Assume ds is a DataSet that has been filled
// First clear any output already sent to the browser
 Response.Clear();

 // Then send header that says the data to follow is xml
 Response.ContentType = "application/xml";

 // Finally get the dataset to write out the XML
 ds.WriteXml(Response.Output);

 // Tell the server we are done
 Response.End();
```

Reading and Processing XML

```
WebRequest req = WebRequest.Create( <some URL> );
WebResponse resp = req.GetResponse();
// get the XML into an XmlDataDocument
XmlTextReader xml = new
        XmlTextReader(resp.GetResponseStream());
XmlDataDocument xmlDoc = new XmlDataDocument();
xmlDoc.Load(xml);
```

Processing an XmlDocument

```
XmlNodeList rows =
            xmlDoc.GetElementsByTagName("Row");
for (int i = 0; i < rows.Count; i++)
{
  XmlNodeList nodes = rows.Item(i).ChildNodes;
  foreach (XmlNode node in nodes)
  {
      Response.Write(node.Name + " = " +
                     node.InnerText + "<BR>");
  }
}
```

Processing an XmlDocument

- In some cases, it is more convenient to use:

```
xmlDoc.DataSet.ReadXml(xml);
```

- Now the `xmlDoc.DataSet` can be used like any other `DataSet` object

XML Support in SQL Server

- Ability to query SQL Server over HTTP
- Retrieve relational data as XML
  - `SELECT … FOR XML`
- Query XML data
- Join XML data with existing database tables
- Update the database via XML Updategrams