Unit-II Advanced ADO.NET

Introduction

ADO.NET provides consistent access to data sources such as Microsoft SQL Server and XML, as well as to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update the data that they contain.

ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results.

ADO.NET is a data-access technology that enables applications to connect to data stores and manipulate data contained in them in various ways. It is based on the .NET Framework and it is highly integrated with the rest of the Framework class library. The ADO.NET API is designed so it can be used from all programming languages that target the .NET Framework, such as Visual Basic, C#, J# and Visual C++.

ADO uses a small set of Automation objects to provide a simple and efficient interface to OLE DB. This interface makes ADO a good choice for developers in higher level languages, such as Visual Basic and VBScript, who want to access data without having to learn the DETAILS of COM and OLE DB.

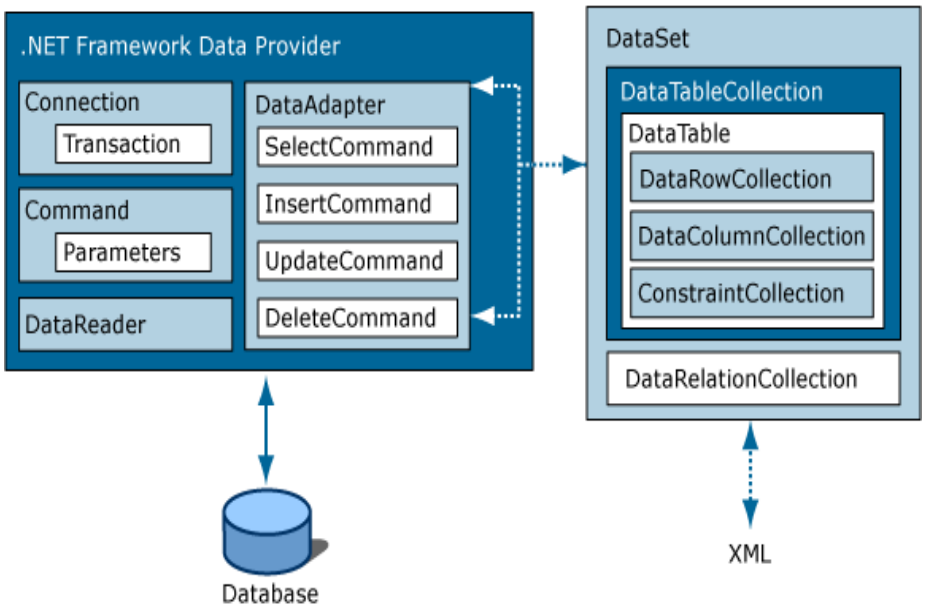
ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO)

Disconnected Data access

ADO.NET Components

There are two components of ADO.NET that you can use to access and manipulate data:

- .NET Framework data providers
- The DataSet



1. NET Framework Data Providers

The NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.

The **Connection** object provides connectivity to a data source.

The **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.

The **DataReader** provides a high-performance stream of data from the data source. Finally, the DataAdapter provides the bridge between the **DataSet** object and the data source.

The **DataAdapter** uses **Command** objects to execute SQL commands at the data source to both load the **DataSet** with data, and reconcile changes made to the data in the **DataSet** back to the data source.

i) The Connection object

Listed below are the common connection object methods we could work with:

- **Open** - Opens the connection to our database
- **Close** - Closes the database connection

- **Dispose** - Releases the resources on the connection object. Used to force garbage collecting, ensuring no resources are being held after our connection is used. Incidentally, by using the Dispose method you automatically call the Close method as well.
- **State** - Tells you what type of connection state your object is in, often used to check whether your connection is still using any resources. Ex. if (ConnectionObject.State == ConnectionState.Open)

ii) The Command Object

- **ExecuteReader** - Simply executes the SQL query against the database, using the Read() method to traverse through data.
- **ExecuteNonQuery** – Used whenever you work with SQL stored procedures with parameters.
- **ExecuteScalar** - Returns a lightning fast single value as an object from your database Ex. object val = Command.ExecuteScalar(); Then check if != null.
- **ExecuteXmlReader** - Executes the SQL query against SQL Server only, while returning an XmlReader object.
- **Prepare** – Equivalent to ADO's Command.Prepared = True property. Useful in caching the SQL command so it runs faster when called more than once. Ex. Command.Prepare();
- **Dispose** – Releases the resources on the Command object. Used to force garbage collecting, ensuring no resources are being held after our connection is used. Incidentally, by using the Dispose method you automatically call the Connection object's Close method as well.

iii) The DataReader Object

- **Read** – Moves the record pointer to the first row, which allows the data to be read by column name or index position.
- **HasRows** - HasRows checks if any data exists, and is used instead of the Read method. Ex. if (DataReader.HasRows).
- **IsClosed** - A method that can determine if the DataReader is closed.
- **Next Result** - Equivalent to ADO's NextRecordset Method, where a batch of SQL statements are executed with this method before advancing to the next set of data results.
- **Close** – Closes the DataReader

iv) The DataAdapter

Using an adapter, you can read, add, update, and delete records in a data source. To allow you to specify how each of these operations should occur, an adapter supports the following four properties:

- **SelectCommand** – reference to a command (SQL statement or stored procedure name) that retrieves rows from the data store.
- **InsertCommand** – reference to a command for inserting rows into the data store.
- **UpdateCommand** – reference to a command for modifying rows in the data store.
- **DeleteCommand** – reference to a command for deleting rows from the data store.

2. The DataSet

The ADO.NET DataSet is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.

The ADO.NET DataSet contains DataTableCollection and their DataRelationCollection . It represents a collection of data retrieved from the Data Source.

The **DataSet** contains a collection of one or more DataTable objects made up of rows and columns of data, as well as primary key, foreign key, constraint, and relation information about the data in the **DataTable** objects.

We can use Dataset in combination with DataAdapter class. The DataSet object offers a disconnected data source architecture. The Dataset can work with the data it contain, without knowing the source of the data coming from. That is, the Dataset can work with a disconnected mode from its Data Source . It gives a better advantage over DataReader , because the DataReader is working only with the connection oriented Data Sources.

In any .NET data access page, before you connect to a database, you first have to import all the necessary namespaces that will allow you to work with the objects required. As we're going to work with SQL Server, we'll first import the namespaces we need. Namespaces in .NET are simply a neat and orderly way of organizing objects, so that nothing becomes ambiguous.

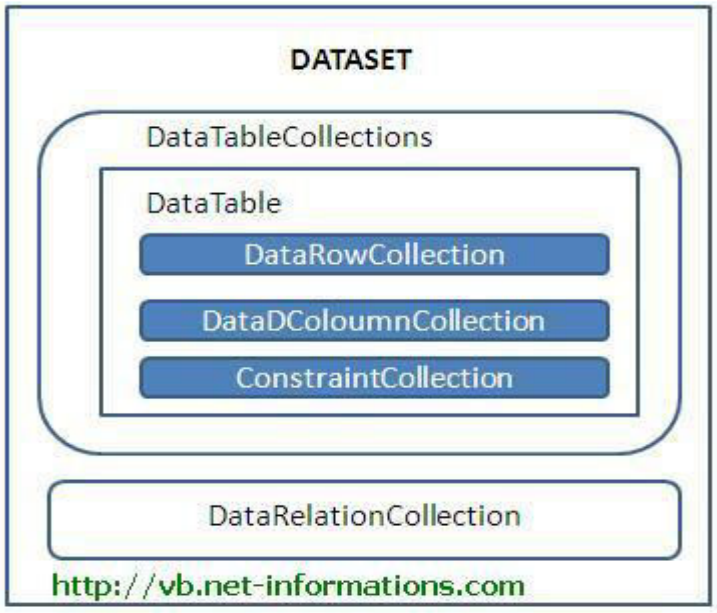
Note

(Namespaces: All the classes are defined in single name called namespaces in ASP.NET.)

Example:

1. <%@ Import Namespace="System" %>
2. <%@ Import Namespace="System.Data" %>
3. <%@ Import Namespace="System.Data.SqlClient" %>)

The Dataset contains the copy of the data we requested. The Dataset contains more than one Table at a time. We can set up Data Relations between these tables within the DataSet. The data set may comprise data for one or more members, corresponding to the number of rows.



GridView Control

GridView control is a successor to the ASP.NET 1.X DataGrid control. It provides more flexibility in displaying and working with data from your database in comparison with any other controls. The GridView control enables you to connect to a datasource and display data is tabular format

Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc.

ProductID	ProductName	UnitPrice	UnitsInStock
1	Dell Studio XPS 8100	999	6
2	Dell Studio XPS 9100	1399	7
3	Dell Inspiron 1545/9176 Laptop	399	6
4	Dell Inspiron M301Z Laptop	599	8
5	HP Envy 13-1100EA Laptop	799	9
6	Motorola A1010	399	4
7	Samsung i710	699	10
8	Nokia N95	499	4
9	HP - Deskjet Printer	599	2
10	Epson - WorkForce 30 Printer	499	9
11	InFocus IN1100	799	9
12	InFocus IN102 Projector	599	9
		8288	83

Average Price: 690.67

GridViews support:

- Automatic sorting (click on a column heading to sort by that column)
- Automatic paging (sort of - true paging is only possible if you use more complicated data sources)
- Editing and deleting of data
- Selection of rows

Important properties

Behavior Properties of the GridView Control	
AllowPaging	true/false. Indicate whether the control should support paging.
AllowSorting	true/false. Indicate whether the control should support sorting.
SortExpression	Gets the current sort expression (field name) that determines the order of the row.
SortDirection	Gets the sorting direction of the column sorted currently (Ascending/Descending).
DataSource	Gets or sets the data source object that contains the data to populate the control.
DataSourceID	Indicate the bound data source control to use (Generally used when we are using SqlDataSource or AccessDataSource to bind the data, See 1st Grid example).

AutoGenerateEditButton	true/false. Indicates whether a separate column should be added to edit the record.
AutoGenerateDeleteButton	true/false. Indicates whether a separate column should be added to delete the record.
AutoGenerateSelectButton	true/false. Indicate whether a separate column should be added to select a particular record.
AutoGenerateColumns	true/false. Indicate whether columns are automatically created for each field of the data source. The default is true.
Style Properties of the GridView Control	
AlternatingRowStyle	Defines the style properties for every alternate row in the GridView.
EditRowStyle	Defines the style properties for the row in EditView (When you click Edit button for a row, the row will appear in this style).
RowStyle	Defines the style properties of the rows of the GridView.
PagerStyle	Defines the style properties of Pager of the GridView. (If AllowPaging=true, the page number row appears in this style)
EmptyDataRowStyle	Defines the style properties of the empty row, which appears if there is no records in the data source.
HeaderStyle	Defines the style properties of the header of the GridView. (The column header appears in this style.)
FooterStyle	Defines the style properties of the footer of GridView.
Appearance Properties of the GridView Control	
CellPadding	Indicates the space in pixel between the cells and the border of the GridView.
CellSpacing	Indicates the space in pixel between cells.
GridLines	Both/Horizontal/Vertical/None. Indicates whether GrdiLines should appear or not, if yes Horizontal, Vertical or Both.
HorizontalAlign	Indicates the horizontal align of the GridView.
EmptyDataText	Indicates the text to appear when there is no record in the data source.

ShowFooter	Indicates whether the footer should appear or not.
ShowHeader	Indicates whether the header should appear or not. (The column name of the GridView)
BackImageUrl	Indicates the location of the image that should display as a background of the GridView.
Caption	Gets or sets the caption of the GridView.
CaptionAlign	left/center/right. Gets or sets the horizontal position of the GridView caption.
State Properties of GridView Control	
Columns	Gets the collection of objects that represent the columns in the GridView.
EditIndex	Gets or sets the 0-based index that identifies the row currently to be edited.
FooterRow	Returns a GridViewRow object that represents the footer of the GridView.
HeaderRow	Returns a GridViewRow object that represents the header of the GridView.
PageCount	Gets the number of the pages required to display the reocrds of the data source.
PageIndex	Gets or sets the 0-based page index.
PageIndex	Gets or sets the number of records to display in one page of GridView.
Rows	Gets a collection of GridViewRow objects that represents the currently displayed rows in the GridView.
DataKeyNames	Gets an array that contains the names of the primary key field of the currently displayed rows in the GridView.
DataKeys	Gets a collection of DataKey objects that represent the value of the primary key fields set in DataKeyNames property of the GridView.

Events associated with GridView Control	
PageIndexChanging, PageIndexChanged	Both events occur when the page link is clicked. They fire before and after GridView handles the paging operation respectively.
RowCancelingEdit	Fires when Cancel button is clicked in Edit mode of GridView.
RowCommand	Fires when a button is clicked on any row of GridView.
RowCreated	Fires when a new row is created in GridView.
RowDataBound	Fires when row is bound to the data in GridView.
RowDeleting,RowDeleted	Both events fires when Delete button of a row is clicked. They fire before and after GridView handles deleting operaton of the row respectively.
RowEditing	Fires when a Edit button of a row is clicked but before the GridView hanldes the Edit operation.
RowUpdating, RowUpdated	Both events fire when a update button of a row is clicked. They fire before and after GridView control update operation respectively.
Sorting, Sorted	Both events fire when column header link is clicked. They fire before and after the GridView handler the Sort operation respectively.

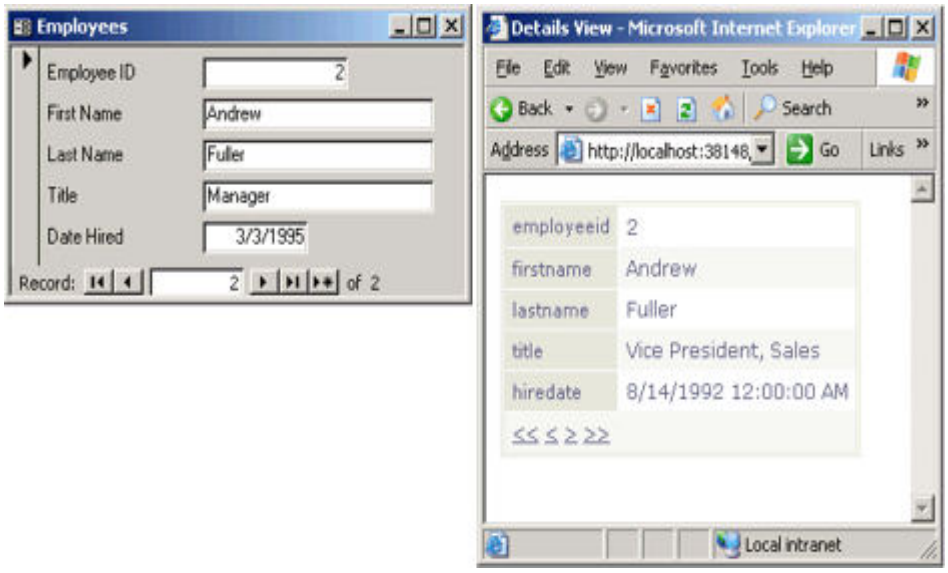
A DetailsView Control

In ASP.NET 2.0, **DetailsView** is a data-bound control that renders a single record at a time from its associated data source. It can optionally provide paging buttons to navigate between records, and a command bar to execute basic operations on the current record (Insert, Update, Delete). **DetailsView** generates a user interface similar to the Form View of a Microsoft Access database, and is typically used for updating/deleting any currently displayed record or for inserting new records.

The key aspects of a **DetailsView** control:

- Be a composite control and act as a naming container.
- Be data-bindable to enumerable data sources.
- Support some style properties.
- Provide a navigation bar (pager).
- Support replaceable views of the record fields.

- Provide a command bar for common operations.



Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc.

Important properties

Behavior Properties of the DetailsView Control	
AllowPaging	true/false. Indicate whether the control should support navigation.
DataSource	Gets or sets the data source object that contains the data to populate the control.
DataSourceID	Indicate the bound data source control to use (Generally used when we are using SqlDataSource or AccessDataSource to bind the data, See 1st Grid example).
AutoGenerateEditButton	true/false. Indicates whether a separate column with edit link/button should be added to edit the record.
AutoGenerateDeleteButton	true/false. Indicates whether a separate column with delete link/button should be added to delete the record.
AutoGenerateRows	true/false. Indicate whether rows are automatically created for each field of the data source. The default is true.
DefaultMode	read-only/insert/edit. Indicate the default display mode.

Style Properties of the DetailsView Control	
AlternatingRowStyle	Defines the style properties for every alternate row in the DetailsView.
EditRowStyle	Defines the style properties for the row in EditView (When you click Edit button for a row, the row will appear in this style).
RowStyle	Defines the style properties of the rows of the DetailsView.
PagerStyle	Defines the style properties of Pager of the DetailsView. (If AllowPaging=true, the page number row appears in this style)
EmptyDataRowStyle	Defines the style properties of the empty row, which appears if there is no records in the data source.
HeaderStyle	Defines the style properties of the header of the DetailsView. (The column header appears in this style.)
FooterStyle	Defines the style properties of the footer of DetailsView.
Appearance Properties of the DetailsView Control	
CellPadding	Indicates the amount of space in pixel between the cells and the border of the DetailsView.
CellSpacing	Indicates the amount of space in pixel between cells.
GridLines	Both/Horizontal/Vertical/None. Indicates whether GrdiLines should appear or not, if yes Horizontal, Vertical or Both.
HorizontalAlign	Indicates the horizontal alignment of the DetailsView.
EmptyDataText	Indicates the text to appear when there is no record in the data source.
BackImageUrl	Indicates the location of the image that should display as a background of the DetailsView.
Caption	Gets or sets the caption of the DetailsView.
CaptionAlign	left/center/right. Gets or sets the horizontal position of the DetailsView caption.
State Properties of DetailsView Control	
Rows	Gets the collection of objects that represent the rows in the DetailsView.

FooterRow	Returns a DetailsViewRow object that represents the footer of the DetailsView.
HeaderRow	Returns a DetailsViewRow object that represents the header of the DetailsView.
PageCount	Gets the number of the pages required to display the records of the data source.
PageIndex	Gets or sets the 0-based page index.
DataKeyNames	Gets an array that contains the names of the primary key field of the currently displayed rows in the DetailsViewRow.
DataKeys	Gets a collection of DataKey objects that represent the value of the primary key fields set in DataKeyNames property of the DetailsViewRow.
Events of the DetailsView Control	
ItemCommand	Fires when any clickable element on the control is clicked.
ItemCreated	Fires after DetailsView fully creates all rows of the record.
ItemDeleting, ItemDeleted	Both event fires when current record is deleted. The first one fires before and other fires after record is deleted.
ItemInserting, ItemInserted	Both event fires when an item is inserted. The first one fires before and second after the item is created.
ItemUpdating, ItemUpdated	Both event fires when an item is updated. The first one fires before and second fires after the record is updated.
ModeChanging, ModeChanged	Both event fires when DetailsView change its display mode. The first one fires before and second fires after display mode is changed.
PageIndexChanging, PageIndexChanged	Both event fires when the DetailsView move to another record. The first one fires before and second fires after page is changed.

FormView Control

The FormView control is used to display a single record from database. It's greater flexiblity is, it displays user-defined templates instead of row fields.

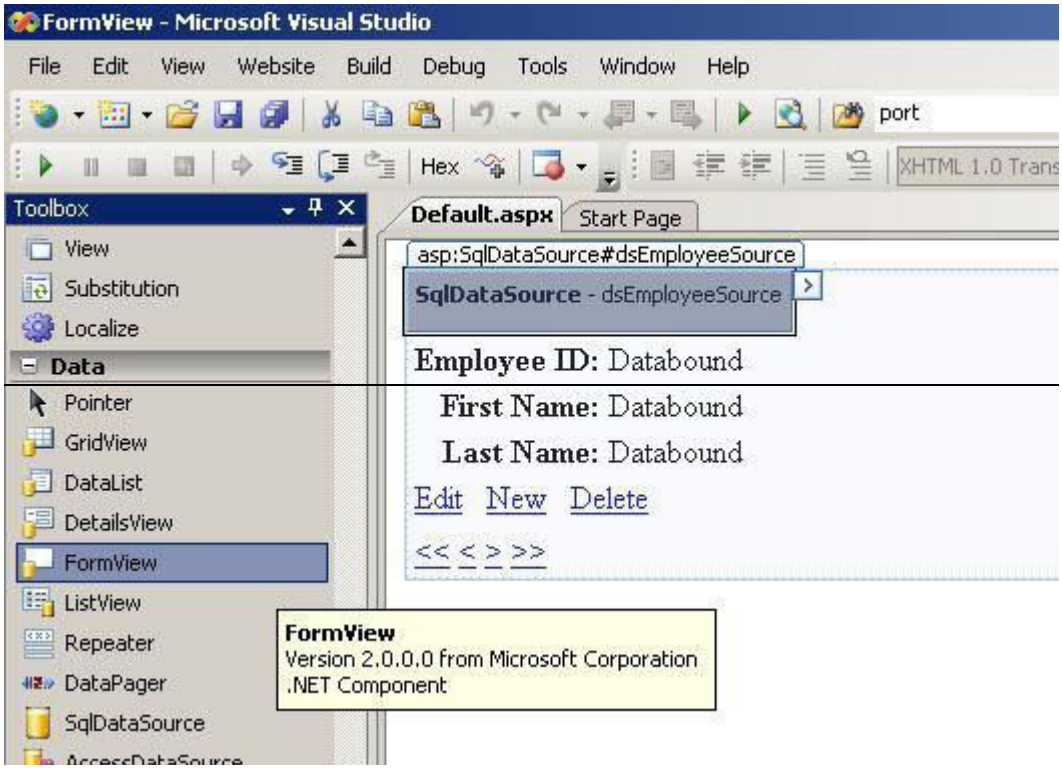
The difference between the FormView and the DetailsView controls is that the DetailsView control uses a tabular layout where each field of the record is displayed as a row of its own. In contrast, the FormView control does not specify a predefined layout for displaying the record. Instead, you create a template containing controls to display individual fields from the record. The template contains the formatting, controls, and binding expressions used to create the form.

The FormView control is typically used for updating and inserting new records, and is often used in master/detail scenarios where the selected record of the master control determines the record to display in the FormView control.

It has the following features:

- Binding to data source controls, such as SqlDataSource and ObjectDataSource.
- Built-in inserting capabilities.
- Built-in updating and deleting capabilities.
- Built-in paging capabilities.
- It's properties, handle events can be set dynamically with FormView object model.
- It can be customized through templates, themes and styles.

- Template are used to display/edit the FormView control.



Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc.

Important properties

HeaderTemplate	Displays the content at header row of the template. The header row is displayed at the top of the FormView control when the HeaderText or HeaderTemplate property is set
EmptyDataTemplate	This is used display the content when datasource control does not contain any records. It alerts the ser that datasource has no records.
ItemTemplate	It is used to display the content when Formview is in read-only mode. The item template usually contains controls to display the field values of a record, as well as command buttons to edit, insert, and delete a record.
EditItemTemplate	Displays the content for the data row when the FormView control is in edit mode. This template usually contains input controls and command buttons with which the user can edit an existing record.

InsertItemTemplate	Displays the content for the data row when the FormView control is in insert mode. This template usually contains input controls and command buttons with which the user can add a new record.
PagerTemplate	Displays the content for the pager row displayed when the paging feature is enabled (when the AllowPaging property is set to true). This template usually contains controls with which the user can navigate to another record.
FooterTemplate	The footer row is displayed at the bottom of the FormView control when the FooterText or FooterTemplate property is set. If both the FooterText and FooterTemplate properties are set, the FooterTemplate property takes precedence.

Methods of the FormView Control	
ChangeMode	ReadOnly/Insert/Edit. Change the working mode of the control from the current to the defined FormViewMode type.
InsertItem	Used to insert the record into database. This method must be called when the DetailsView control is in insert mode.
UpdateItem	Used to update the current record into database. This method must be called when DetailsView control is in edit mode.
DeleteItem	Used to delete the current record from database.

ItemCommand	Occurs when a button within a FormView control is clicked. This event is often used to perform a task when a button is clicked in the control.
ItemCreated	Occurs after all FormViewRow objects are created in the FormView control. This event is often used to modify the values of a record before it is displayed.
ItemDeleted	Occurs when a Delete button (a button with its CommandName property set to "Delete") is clicked, but after the FormView control deletes the record from the data source. This event is often used to check the results of the delete operation.
ItemDeleting	Occurs when a Delete button is clicked, but before the FormView control deletes the record from the data source. This event is often used to cancel the delete operation.

ItemInserted	Occurs when an Insert button (a button with its CommandName property set to "Insert") is clicked, but after the FormView control inserts the record. This event is often used to check the results of the insert operation.
ItemInserting	Occurs when an Insert button is clicked, but before the FormView control inserts the record. This event is often used to cancel the insert operation.
ItemUpdated	Occurs when an Update button (a button with its CommandName property set to "Update") is clicked, but after the FormView control updates the row. This event is often used to check the results of the update operation.
ItemUpdating	Occurs when an Update button is clicked, but before the FormView control updates the record. This event is often used to cancel the update operation.
ModeChanged	Occurs after the FormView control changes modes (to edit, insert, or read-only mode). This event is often used to perform a task when the FormView control changes modes.
ModeChanging	Occurs before the FormView control changes modes (to edit, insert, or read-only mode). This event is often used to cancel a mode change.

Crystal Reports in ASP.NET

Crystal Reports is the standard reporting tool for Visual Studio .NET used to display data of presentation quality. You can display multiple-level totals, charts to analyze data, and much more in Crystal Reports. Creating a Crystal Report requires minimal coding since it is created in Designer interface. It is available as an integrated feature of Microsoft Visual Studio .NET, Borland Delphi, and C#Builder.

Advantages of Crystal Reports

Some of the major advantages of using Crystal Reports are:

1. Rapid report development since the designer interface would ease the coding work for the programmer.

2. Can extend it to complicated reports with interactive charts and enhance the understanding of the business model

3. Exposes a report object model, can interact with other controls on the ASP.NET Web form

4. Can programmatically export the reports into widely used formats like .pdf, .doc, .xls, .html and .rtf

Implementation Models

Crystal Reports need database drivers to connect to the data source for accessing data. Crystal Reports in .net support two methods to access data from a data source:

The Pull Method

When this model is used to access data from the data source, the database driver directly retrieves the data from the data source. This model does not require the developer to write code for creating a connection and retrieving data from the data source. It is the Crystal report that manages the SQL commands for connecting by using the specified driver.

The Push Method

When this model is used to access data from data source, the developer writes the code to connect to the data source and retrieve data. The data from the data source is cached in dataset and multiple crystal reports accesses data from the dataset. The performance can be optimized in this manner by using connection sharing and manually limiting the number of records that are passed on to the report.

Crystal Reports Types

Crystal Report Designer can load reports that are included into the project as well as those that are independent of the project.

Strongly-typed Report

When you add a report file into the project, it becomes a "strongly-typed" report. In this case, you will have the advantage of directly creating an instance of the report object, which could reduce a few lines of code, and cache it to improve performance. The related .vb file, which is hidden, can be viewed using the editor's "show all files" icon in the Solution Explorer.

Un-Typed Report

Those reports that are not included into the project are "un-typed" reports. In this case, you will have to create an instance of the Crystal Report Engine's "ReportDocument" object and manually load the report into it.

Creating Crystal Reports

You can create a Crystal Report by using three methods:

1. Manually i.e. from a blank document
2. Using Standard Report Expert
3. From an existing report

Using Pull Method

Creating Crystal Reports Manually.

We would use the following steps to implement Crystal Reports using the Pull Model:

1. **Create the .rpt file** (from scratch) and set the necessary database connections using the Crystal Report Designer interface.

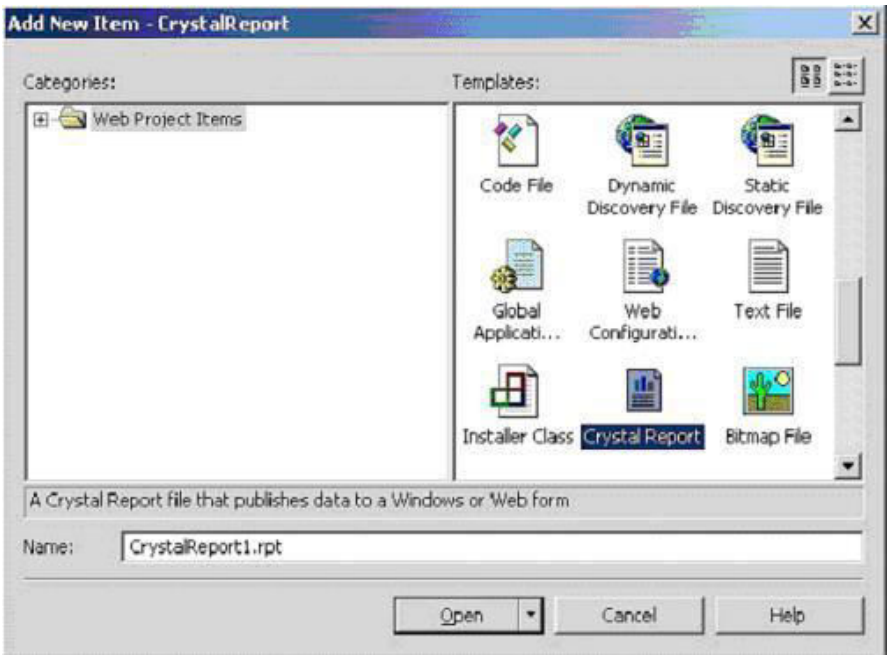
2. **Place a CrystalReportViewer control** from the toolbox on the .aspx page and set its properties to point to the .rpt file that we created in the previous step.

3. Call the databind method from your code behind page.

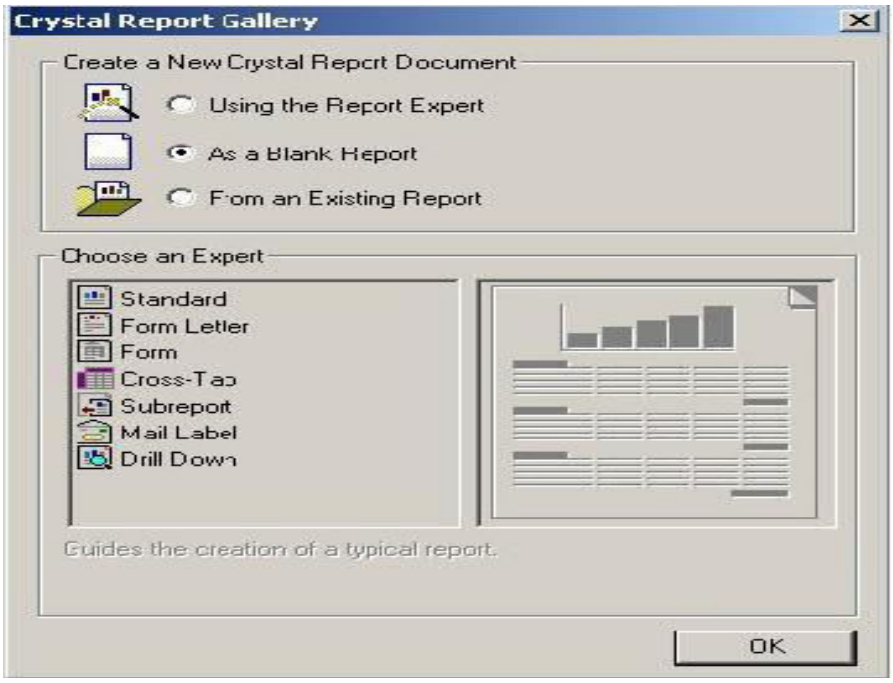
Creating Crystal Reports

I. Steps to create the report i.e. the .rpt file

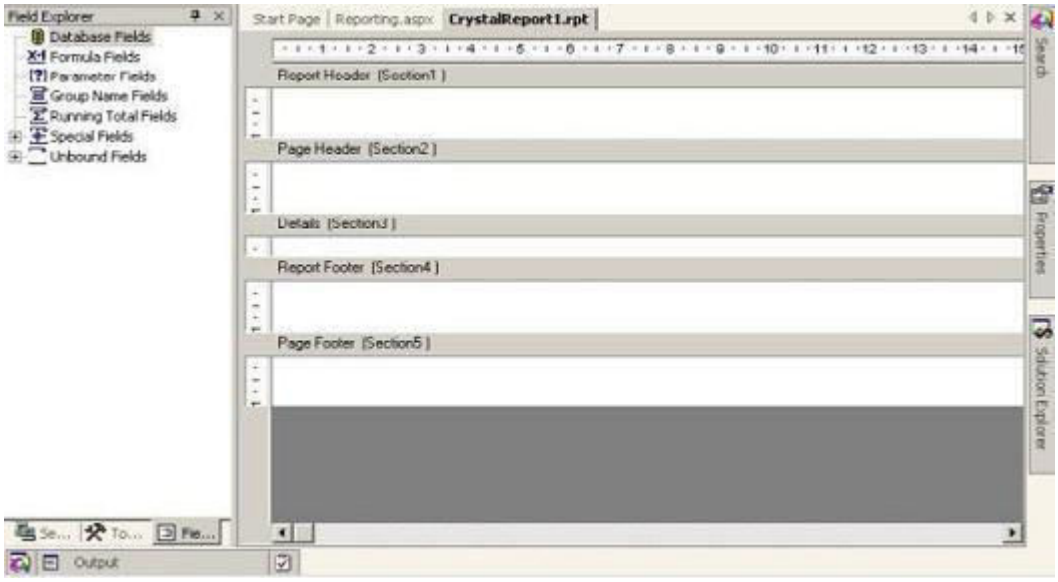
1) Add a new Crystal Report to the web form by right clicking on the "Solution Explorer", selecting "Add" --> "Add New Item" --> "CrystalReport".



2) On the "Crystal Report Gallery" pop up, select the "As a Blank Report"radio button and click "ok".

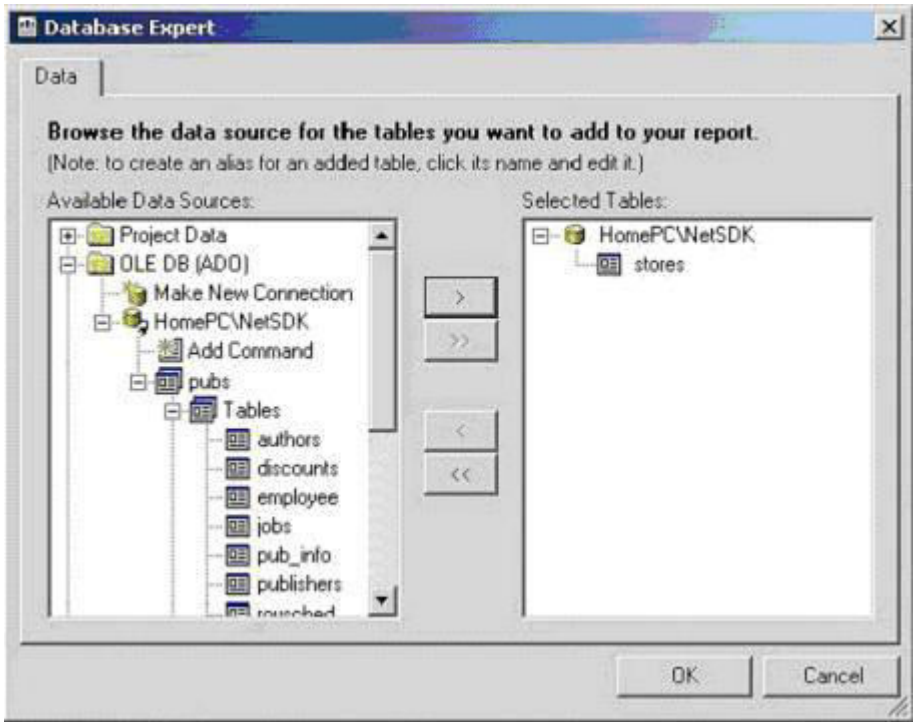


3) This should open up the Report File in the Crystal Report Designer.



4) Right click on the "Details Section" of the report, and select "Database" - "Add/Remove Database".

5) In the "Database Expert" pop up window, expand the "OLE DB (ADO)"option by clicking the "+" sign, which should bring up another "OLE DB (ADO)" pop up.
6) In the "OLE DB (ADO)" pop up, Select "Microsoft OLE DB Provider for SQL Server" and click Next.
7) Specify the connection information.
8) Click "Next" and then click "Finish".
9) Now you should be able to see the Database Expert showing the table that have been selected.
10) Expand the "Pubs" database, expand the "Tables", select the "Stores" table and click on ">" to include it into the "Selected Tables" section.

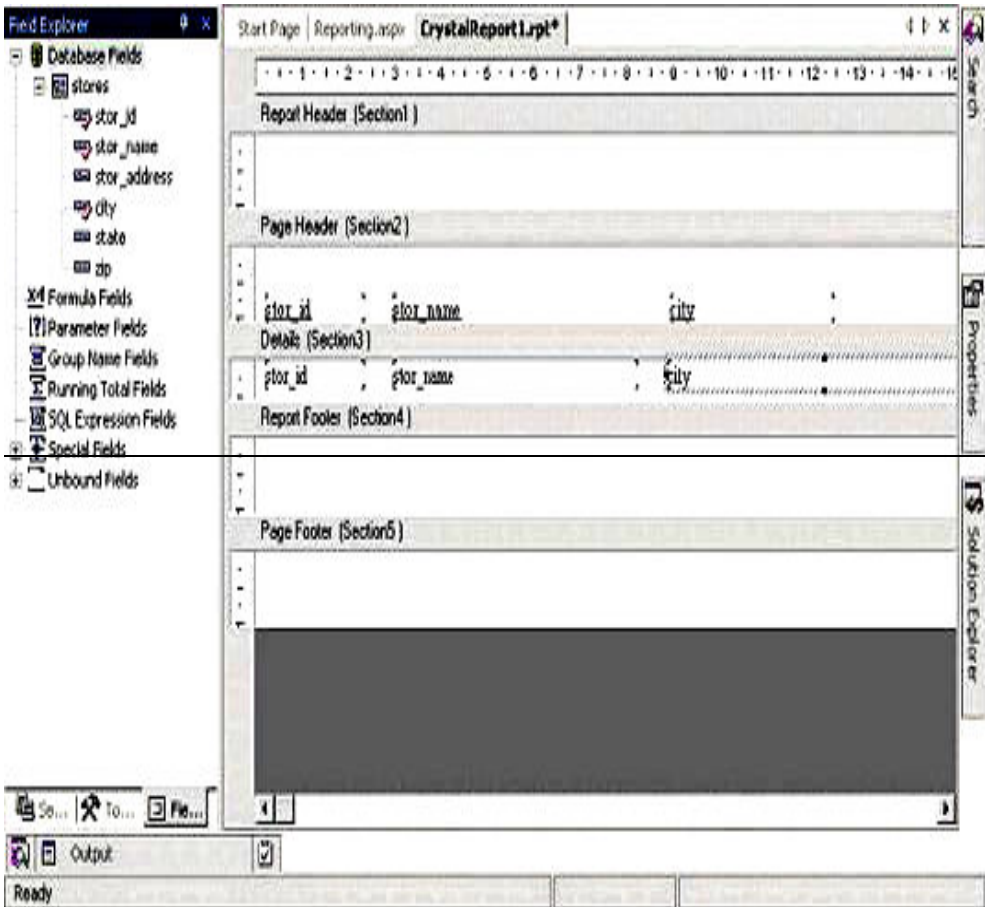


Note: If you add more than one table in the database Expert and the added tables have matching fields, when you click the OK button after adding the tables, the links between the added tables is displayed under the Links tab. You can remove the link by clicking the Clear Links button.

11) Now the Field Explorer should show you the selected table and its fields under the "Database Fields" section, in the left window.

12) Drag and drop the required fields into the "Details" section of the report. The field names would automatically appear in the "Page Header" section of the report. If you want to modify the header text then right click on the text of the "Page Header" section, select "Edit Text Object" option and edit it.

13) Save it.



Role of ADO.NET in Distributed Applications

The rapid development of web applications makes software development companies review the existing methods of working with data sources and adapt them to the web application specifications. The unpredicted growth of the number of clients makes web developers move from client-server to three-tier architecture, which sometimes brings out problems. Databases are unable to support the unlimited number of active connections limiting the availability of the site and causing losses. The ADO.NET (ActiveX Data Objects) technology can solve these problems and at the same time keep convenience and simplicity of programming.

Advantages and innovations in ADO.NET technology

- **Using the disconnected model for accessing the data.** ADO.NET application development technology offers an alternative to a traditional data access model. Normally, client-server applications use the technology of access to the data source where the connection with the base is maintained all the time. However, after the wide spread of the Internet based applications some vulnerabilities of this approach have been

discovered. The experience of web developers has shown that the applications with the constant connection with the data source are difficult in scaling. All these problems are produced by the constant connection with database and are solved in ADO.NET. ADO.NET technology makes use of another data access model. ADO.NET access model establishes the connection only for some limited time when it's necessary to take some actions with the database. Thus, ADO.NET allows sidestepping these limitations of web application development process.

- **Data string in the DataSet objects.** In general, DataSet is a simplified relational database and can perform the most typical for these bases operations. Owing to ADO.NET application development technology, in contrast to Recordset, we can store several tables in one DataSet as well as the relations between them, perform the operations of selecting, deleting and updating the data. ADO.NET gives an opportunity any minute to get the latest information from the database using the call function FillDataSet. Thus, ADO.NET application development technology makes DataSet extremely convenient for most web applications. ADO.NET application development technology allows us to extract the data from the base and somehow handle it whenever it is necessary.
- **Deep integration with XML.** XML, a widely spread language, plays an important role in ADO.NET and brings some more benefits to ADO.NET application development technology in comparison with the traditional approach. It isn't necessary for a programmer working with ADO.NET to have the experience of working with XML or the knowledge about this language. ADO.NET makes all the operations transparent for web developers. XML (eXtensible Markup Language) represents an industrial standard supported by practically any modern platform, which allows transmitting data to any component that can work with XML and can be executed under any operating system. Thus, deep integration of ADO.NET with XML provides .NET application developers with ample opportunities.

Many application developers have already noticed the simplicity and convenience of the ADO.NET technology. ADO.NET application development technology provides an intuitive interface and logical set of objects. All these features make ADO.NET more appealing to.NET web developers.