**Data Controls**

To use the data source controls you must have a data-bound control to bind them to. There are several new data-bound controls in ASP.NET 2.0, including the GridView, DetailsView and FormView controls. If you are fond of the ASP.NET 1.*x* DataGrid control, you will love the ASP.NET 2.0 GridView control. The GridView is like a DataGrid on steroids as it can be bound to the new data source controls, and can be used to implement sorting, editing, and paging, all with much less code than the DataGrid required.

To bind a GridView to a data source control, you set the GridView's DataSourceID property to the ID of the data source control. Other controls, such as the DropDownList, can also be bound to the data source controls. For example, a DropDownList control can be bound to a SqlDataSource control that retrieves a list of employees. The employee's full name could be displayed in the DropDownList while the EmployeeID could be bound to the control as its underlying data value field. The following example defines a DropDownList that will display a list of customer names that can be selected. Binding a control to a data source control is quite simple in ASP.NET 2.0 and does not require any code in a codebehind. However, you can still write code to explicitly bind to the controls if you want to. In fact, the data source and DataMember proper-ties of data-bound controls have been brought forward from ASP.NET 1.x, as well. There are several new data source controls in ASP.NET 2.0, such as the SqlDataSource, ObjectDataSource, XmlDataSource, AccessDataSource, and SiteMapDataSource (shown in **Figure 1**). They all can be used to retrieve data from their respective types of data sources and can be bound to various data-bound controls. Data source controls simplify the amount of custom code that needs to be written to retrieve and bind data, and even to sort, page through, or edit data.

* Figure 1 ASP.NET 2.0 Data Source Controls

|  |  |
| --- | --- |
| **Data Source Control** | **Retrieves Its Data From** |
| SqlDataSource | A SQL Server, Oracle, or other OLE DB accessible database |
| AccessDataSource | An access database |
| ObjectDataSource | A business object or method |
| XmlDataSource | An XML file or stream (used for hierarchical data) |
| SiteMapDataSource | An XML-based sitemap source |

Each data source control has similar properties that allow it to interact with its respective data source. The SiteMapDataSource and the XmlDataSource are built to retrieve hierarchical data while the other data source controls are a built to retrieve set-based data with columns and rows.

The AccessDataSource is built specifically to retrieve data from Access databases. The SqlDataSource may sound like it works only with SQL Server™, but that's not the case. It can actually be used to retrieve data from any OLE DB or ODBC-compliant data source.

**Validation Controls**

**Validation Controls in Asp.net**

**Introduction:**

   Validation is important part of any web application. User's input must always be validated before sending across different layers of the application. In this article we will take a look at the validation controls provided by Asp.net.

**Required Field Validator:**

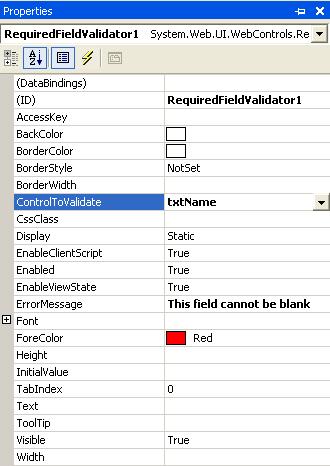
Required field validate control is used on controls where user must enter some data. If the data is is not entered and the required field validator is set than it stop the processing of the page until the data is entered in that control. Let's see how we can use Required Field Validator control.



Above is the image of the [TextBox](http://www.codersource.net/asp_net_validation_controls.aspx" \t "undefined)

[http://kona.kontera.com/javascript/lib/imgs/grey_loader.gif](http://www.codersource.net/asp_net_validation_controls.aspx" \t "undefined)

that we will validate. The validation will be fired when the submit button is clicked. First let's add a Required field validator.



As you can see in the image above that I have set the ErrorMessage property to "This field cannot be blank" which is the message that will be displayed when the user does not input any data in the textbox. The property ControlToValidate let's the asp.net know that which control we are validating. In this case we are validating a textbox control whose name is "txtName". After setting these properties let's implement the button click code that will fire the validation event.

|  |
| --- |
| private void Button1\_Click(object sender, System.EventArgs e)  {  if(Page.IsValid == true)  {  // Do some processing here  }  } |

In the button click event we are checking that if the page is valid or not by using Page.IsValid property. If Page.IsValid property is true than the page will be processed further. If the Page.IsValid property is false than the Required field message will be automatically displayed on the screen.

Required field validators are not only used for the TextBox controls but can also be used for DropDownList and other controls. Let's see how we can use Validation on the DropDownList control.



As you can see in the image above that we have a DropDownList which has several items. The first choice is "Select any Item". You don't want the user to submit the page with "Select any item" being selected. But you want the user to choose between Item1, Item2 or Item3. We can easily do this using the Required Field Validator control.

Required Field Validator control have a property named "[Initial Value](http://www.codersource.net/asp_net_validation_controls.aspx)" which can be set to "Select any item" for the item that you don't want to be selected. Now if "Select any Item" is selected than you will receive a [error message](http://www.codersource.net/asp_net_validation_controls.aspx" \t "undefined)

[http://kona.kontera.com/javascript/lib/imgs/grey_loader.gif](http://www.codersource.net/asp_net_validation_controls.aspx" \t "undefined)

saying that "Please select a different Item". You can display any error message that you want.

**Compare Validator Control:**

Compare validators are used when you need to compare two fields. It can most commonly be used when you need to confirm password entered by the user at the registration time. As you can see in the image below that the two fields are not equal. In order for the compare validator to return true both the fields must be exactly same. It is case sensitive that means "codersource" and "Codersource" are not equal. The button click code will be the same which checks for the Page.IsValid property to be true.

http://www.codersource.net/images/validation_Image4.JPG

**Range Validator Control:**

Range validator control is used when you need to check for a certain range. Suppose you have a textbox and you want to check that user enters a number in the textbox that is greater than 25. For this purpose you can use the range validator control. The Range validator control exposes properties "Minimum Value" and "Maximum Value" which can be used to check for certain range. Let's activate this range validators controls programmatically:

|  |
| --- |
| private void Page\_Load(object sender, System.EventArgs e)  {  RangeValidator1.MinimumValue = 25;  RangeValidator1.MaximumValue = 500;  } |

As you can see that using RangeValidator control is pretty simple. We just set the MinimumValue and the MaximumValue and now if the user enters a number which is less than or greater than MinimumValue and MaximumValue respectively he will receive an error message.

**Regular Expression Validator Control:**

[Regular Expression](http://www.codersource.net/asp_net_validation_controls.aspx) Validator is the coolest of all validators controls. It is used to validate complex expressions. These expressions can be [phone number](http://www.codersource.net/asp_net_validation_controls.aspx" \t "undefined)

[http://kona.kontera.com/javascript/lib/imgs/grey_loader.gif](http://www.codersource.net/asp_net_validation_controls.aspx" \t "undefined)

, email address, zip code and many more. Using Regular Expression Validator is also very straight forward. Simply set the ValidationExpression property to any type of expression you want and it will validate it. If you find that it does not contain the expression that you are looking for than you can make the expression yourself by using the using System.Text.RegularExpressions; namespace.

**Validation Summary Control:**

Validation Summary Control is used to display the validation error messages in one place. Suppose you want to display all the validation messages in one page to make the page look better and cleaner than you can use Validation Summary Control. In order to work with the Validation Summary Control you must set the Text property of the other validation controls. If you don't set the Text property of the other Validation controls than the validation control won't know that what to display when any validation error occurs. So what ever you put in the Text property of the validation control it will be displayed in the Validation Summary control.

**Mixing different Expressions:**

Its a common practice to mix different validation controls together to make a more strong validation. Suppose you have a textbox on the page which you want to validate as an email address and that is also a required field. So first of all you will use a Required Field Validator control so that the field is not left blank and than you will use the Regular Expression Validator to validate it against the email address expression.

**Additional Tip:**

Here is a very cool tip for you. Suppose you are using many user controls on a single page. Each user control has a button but you don't want the validation to be fired when the button from the user control is clicked. For this you can set the CausesValidation property to false for the button that you don't want the validation to fire.

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user. You can add input validation to ASP.NET Web pages using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validation—for example, testing for valid dates or values within a range—plus ways to provide custom-written validation. In addition, validation controls allow you to customize how error information is displayed to the user.

Validation controls can be used with any controls you put on an ASP.NET Web page, including both HTML and Web server controls.

You enable validation of user input by adding validation controls to your page as you would add other server controls. There are controls for different types of validation, such as range checking or pattern matching. For a complete list of validation types, see [Types of Validation for ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/bwd43d0x.aspx). Each validation control references an input control (a server control) elsewhere on the page. When user input is being processed (for example, when a page is submitted), the validation control tests the user input and sets a property to indicate whether the entry passed the test. After all of the validation controls have been called, a property on the page is set indicating whether any validation check has failed.

Validation controls can be associated into validation groups so that validation controls belonging to a common group are validated together. You can use validation groups to selectively enable or disable validation for related controls on a page. Other validation operations, such as displaying a [ValidationSummary](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.validationsummary.aspx) control or calling the [GetValidators](http://msdn.microsoft.com/en-us/library/system.web.ui.page.getvalidators.aspx) method, can reference the validation group.

You can test the state of the page and of individual controls in your own code. For example, you would test the state of the validation controls before updating a data record with information entered by the user. If you detect an invalid state, you bypass the update. Typically, if any validation checks fail, you skip all of your own processing and return the page to the user. Validation controls that detect errors then produce an error message that appears on the page. You can display all validation errors in one place using a [ValidationSummary](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.validationsummary.aspx) control.

Validation controls perform input checking in server code. When the user submits a page to the server, the validation controls are invoked to check the user input, control by control. If a validation error is detected in any of the input controls, the page itself is set to an invalid state so you can test for validity before your code runs. Validation occurs after page initialization (that is, after view state and postback data have been processed) but before any change or click event handlers are called.

If the user is working with a browser that supports ECMAScript (Javascript), the validation controls can also perform validation using client script. This can improve response time in the page because errors are detected immediately and error messages are displayed as soon as the user leaves the control containing the error. If client-side validation is available, you have greater control over the layout of error messages and can display an error summary in a message box. For more information, see [Client-Side Validation for ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/yb52a4x0.aspx).

ASP.NET performs validation on the server even if the validation controls have already performed it on the client, so that you can test for validity within your server-based event handlers. In addition, re-testing on the server helps prevent users from being able to bypass validation by disabling or changing the client script check.

Each validation control typically performs one test. However, you might want to check for multiple conditions. For example, you might want to specify both that a user entry is required and that the user entry is limited to accepting dates within a specific range.

You can attach more than one validation control to an input control on a page. In that case, the tests performed by the controls are resolved using a logical AND operator, which means that the data entered by the user must pass all of the tests in order to be considered valid.

In some instances, entries in several different formats might be valid. For example, if you are prompting for a phone number, you might allow users to enter a local number, a long-distance number, or an international number. Using multiple validation controls would not work in this instance because the user input must pass all tests to be valid. To perform this type of test—a logical OR operation where only one test must pass—use the [RegularExpressionValidator](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.regularexpressionvalidator.aspx) validation control and specify multiple valid patterns within the control. Alternatively, you can use the [CustomValidator](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.customvalidator.aspx) validation control and write your own validation code.

**Customizing Validation**

You can customize the validation process in the following ways:

* You can specify the format, text, and location of error messages. In addition, you can specify whether the error messages appear individually or as a summary.
* You can create custom validation using [CustomValidator](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.customvalidator.aspx) control. The control calls your logic but otherwise functions like other validation controls in setting error state, displaying error messages, and so on. This provides an easy way to create custom validation logic while still using in the validation framework of the page.
* For client-side validation, you can intercept the validation call and substitute or add your own validation logic.

**Login Controls**

The ASP.NET login controls provide a robust login solution for ASP.NET Web applications without requiring programming. By default, login controls integrate with ASP.NET membership and forms authentication to help automate user authentication for a Web site.By default, the ASP.NET login controls work in plain text over HTTP. If you are concerned about security, use HTTPS with SSL encryption.

The Login Control

The [Login](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control displays a user interface for user authentication. The [Login](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control contains text boxes for the user name and password and a check box that allows users to indicate whether they want the server to store their identity using ASP.NET membership and automatically be authenticated the next time they visit the site.

The [Login](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control has properties for customized display, for customized messages, and for links to other pages where users can change their password or recover a forgotten password. The [Login](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control can be used as a standalone control on a main or home page, or you can use it on a dedicated login page.

If you use the [Login](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control with ASP.NET membership, you do not need to write code to perform authentication. However, if you want to create your own authentication logic, you can handle the [Login](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control's [Authenticate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.authenticate.aspx) event and add custom authentication code.

 The LoginView Control

The [LoginView](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginview.aspx) control allows you to display different information to anonymous and logged-in users. The control displays one of two templates: the [AnonymousTemplate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginview.anonymoustemplate.aspx) or the [LoggedInTemplate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginview.loggedintemplate.aspx). In the templates, you can add markup and controls that display information appropriate for anonymous users and authenticated users, respectively.

The [LoginView](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginview.aspx) control also includes events for [ViewChanging](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginview.viewchanging.aspx) and [ViewChanged](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginview.viewchanged.aspx), which allow you to write handlers for when the user logs in and changes status.

The LoginStatus Control

The [LoginStatus](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginstatus.aspx) control displays a login link for users who are not authenticated and a logout link for users who are authenticated. The login link takes the user to a login page. The logout link resets the current user's identity to be an anonymous user.

You can customize the appearance of the [LoginStatus](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginstatus.aspx) control by setting the [LoginText](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginstatus.logintext.aspx) and [LoginImageUrl](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginstatus.loginimageurl.aspx) properties.

The LoginName Control

The [LoginName](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.loginname.aspx) control displays a user's login name if the user has logged in using ASP.NET membership. Alternatively, if your site uses integrated Windows authentication, the control displays the user's Windows account name.

The PasswordRecovery Control

The [PasswordRecovery](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.aspx) control allows user passwords to be retrieved based on the e-mail address that was used when the account was created. The [PasswordRecovery](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.aspx) control sends an e-mail message containing a password to the user.

You can configure ASP.NET membership to store passwords using non-reversible encryption. In that case, the [PasswordRecovery](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.aspx) control generates a new password instead of sending the original password to the user.

You can also configure membership to include a security question that the user must answer to recover a password. If you do, the [PasswordRecovery](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.aspx) control asks the question and checks the answer before recovering the password.

The [PasswordRecovery](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.aspx) control requires that your application can forward e-mail message to a Simple Mail Transfer Protocol (SMTP) server. You can customize the text and format of the e-mail message sent to the user by setting the [MailDefinition](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.maildefinition.aspx) property.

The ChangePassword Control

The [ChangePassword](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.aspx) control allows users to change their password. The user must first supply the original password and then create and confirm the new password. If the original password is correct, the user password is changed to the new password. The control also includes support for sending an e-mail message about the new password.

The [ChangePassword](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.aspx) control includes two templated views that are displayed to the user. The first is the [ChangePasswordTemplate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.changepasswordtemplate.aspx), which displays the user interface used to gather the data required to change the user password. The second template is the [SuccessTemplate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.successtemplate.aspx), which defines the user interface that is displayed after a user password has been successfully changed.

The [ChangePassword](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.aspx) control works with authenticated and non-authenticated users. If a user has not been authenticated, the control prompts the user for a login name. If the user is authenticated, the control populates the text box with the user's login name.

**Web parts controls**

ASP.NET Web Parts is an integrated set of controls for creating Web sites that enable end users to modify the content, appearance, and behavior of Web pages directly from a browser. The modifications can be applied to all users on the site or to individual users. When users modify pages and controls, the settings can be saved to retain a user's personal preferences across future browser sessions, a feature called personalization. These Web Parts capabilities mean that developers can empower end users to personalize a Web application dynamically, without developer or administrator intervention.

Using the Web Parts control set, you as a developer can enable end users to:

* Personalize page content. Users can add new Web Parts controls to a page, remove them, hide them, or minimize them like ordinary windows.
* Personalize page layout. Users can drag a Web Parts control to a different zone on a page, or change its appearance, properties, and behavior.
* Export and import controls. Users can import or export Web Parts control settings for use in other pages or sites, retaining the properties, appearance, and even the data in the controls. This reduces data entry and configuration demands on end users.
* Create connections. Users can establish connections between controls so that, for example, a chart control could display a graph for the data in a stock ticker control. Users could personalize not only the connection itself, but the appearance and details of how the chart control displays the data.
* Manage and personalize site-level settings. Authorized users can configure site-level settings, determine who can access a site or page, set role-based access to controls, and so on. For example, a user in an administrative role could set a Web Parts control to be shared by all users, and prevent users who are not administrators from personalizing the shared control.

Developer Scenarios for Using Web Parts

You will typically work with Web Parts in one of three ways: creating pages that use Web Parts controls, creating individual Web Parts controls, or creating complete, personalizable Web applications, such as a portal.

### Page Development

Page developers can use visual design tools such as Microsoft Visual Studio 2005 to create pages that use Web Parts. One advantage in using a tool such as Visual Studio is that the Web Parts control set provides features for drag-and-drop creation and configuration of Web Parts controls in a visual designer. For example, you can use the designer to drag a Web Parts zone, or a Web Parts editor control, onto the design surface, and then configure the control right in the designer using the UI provided by the Web Parts control set. This can speed development of Web Parts applications and reduce the amount of code you have to write.

### Control Development

You can use any existing ASP.NET control as a Web Parts control, including standard Web server controls, custom server controls, and user controls. For maximum programmatic control of your environment, you can also create custom Web Parts controls that derive from the [WebPart](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.webparts.webpart.aspx) class. For individual Web Parts control development, you will typically either create a user control and use it as a Web Parts control, or develop a custom Web Parts control.

As an example of developing a custom Web Parts control, you could create a control to provide any of the features provided by other ASP.NET server controls that might be useful to package as a personalizable Web Parts control: calendars, lists, financial information, news, calculators, rich text controls for updating content, editable grids that connect to databases, charts that dynamically update their displays, or weather and travel information. If you provide a visual designer with your control, then any page developer using Visual Studio can simply drag your control into a Web Parts zone and configure it at design time without having to write additional code.

### Web Application Development

Developing fully integrated and personalizable Web applications--such as a portal-- involves the most comprehensive use of Web Parts. You can develop a Web site that allows extensive user personalization of the UI and content--with features similar to [MSN](http://www.msn.com/). Or you can even develop a packaged application that can be shipped and used by companies or fee-based ISPs that provide portal hosting services.

In a Web application scenario, you could offer a complete solution for end users to manage and personalize the application. This could include a set of Web Parts controls that provide the desired features for the site, a consistent set of themes and styles that allow end users to personalize the UI in a consistent way, catalogs of Web Parts controls from which users can select the ones they want to appear on a page, authentication services, and role-based management (for example, allowing administrative users to personalize Web Parts controls and site settings for all users).

For each part of your application, you can extend the Web Parts control set as needed to provide greater control over the environment. For example, besides authoring custom Web Parts controls for the primary UI of your pages, you might also want to develop a custom Web Parts catalog that is consistent with the look and feel of your application, and gives users more flexibility to choose how controls are added to a page. Or you could extend a zone control to provide additional UI options for the Web Parts controls it contains. You could also write a custom personalization provider to give more flexibility and control over how the personalization data is stored and managed.