COMP 3111: Software Engineering

Validating Data in ASP.NET
Tutorial and Lab Notes

Data validation is the process of ensuring that a program operates on clean, correct and useful data. Data validation checks that data are valid, sensible, reasonable and secure before they are processed by using routines, called "validation rules" or "check routines", that check for correctness, meaningfulness and security of data that are input to a system[1]. A validation process involves two distinct steps: (a) validation check and (b) post-check action. The validation check step uses one or more computational rules to determine if the data is valid. The post-check action provides feedback to help enforce validation. The absence of data validation can lead to data corruption or security vulnerability[2].

The simplest data validation verifies that input characters come from a valid set. For example, telephone numbers should include digits and possibly the characters +, -, (, and) (plus, minus, and parentheses). More sophisticated data validation would check whether the user had entered a valid country code (i.e., that the number of digits and/or characters entered matched the convention for the country or area specified).

RESTRICTING THE INPUT LENGTH

One of the easiest validation checks is to ensure that the length of any data input (e.g., a text string) does not exceed that allowed for the field. This is particularly important if the data will be stored in a database, since most database attributes, such as character string attributes, will have a maximum length specified for them. To limit the number of characters that a user can enter in an input server control, such as a TextBox control, you can set the MaxLength property of the control in ASP.NET.

For the ASU "StudentRegistrationPage.aspx" web form, the MaxLength property of the TextBox controls should be set to that specified in Table 1.

TextBox Control ID	MaxLength
StudentId	8
FirstName	20
LastName	20
Address	50
UserName	15
Email	25
Password	15
ConfirmPassword	15

Table 1: MaxLength TextBox control properties for Student Registration Page.

SETTING THE TEXTMODE PROPERTY

ASP.NET has a TextMode property that supports HTML5 types, which restricts the type of input allowed in a TextBox control. The TextMode property can be set to the following values[3].

- *Color* – used for color entries.
- *Date* – allows the user to select a date.
- *DateTime* – allows the user to select a date and time with time zone.
- *DateTimeLocal* – allows the user to select a date and time without the time zone.
- *Email* – requires valid email address to be input.
- *Month* – allows the user to select a month and year without time zone.
- *MultiLine* – allows text to be entered on multiple lines.
- *Number* – requires a numeric value to be input.
- *Password* – masks the data input.
- *Phone* – requires a telephone value to be input.
- *Range* – requires a value from a range of numbers to be input.
- *Search* – used for search fields and behaves like a regular text field.
- *SingleLine* – allows text to be entered on a single line (the default).
- *Time* – allows the user to select a time.
- *Url* – requires a URL address to be input.
- *Week* – allows the user to select a week and year without time zone.

For the ASU "StudentRegistrationPage.aspx" web form, set the TextMode property of the "Email" TextBox to "Email".

[1] By default, ASP.NET web pages automatically validate that malicious users are not attempting to send script or HTML elements to your application. See Script Exploits Overview for more information.
[2] From http://www.ask.com/wiki/Data_validation.
[3] Many TextMode values are not supported in IE (see http://www.codeproject.com/Tips/776764/Introducing-Textmode-in-ASP-NET).

ASP.NET VALIDATION SERVER CONTROLS

ASP.NET validation server controls, described in the Appendix, allow you to validate an associated input server control and display a custom message when validation fails[4]. Each validation control performs a specific type of validation. Multiple validation controls can be associated with the same input control. For example, a RequiredFieldValidator can be used to ensure that something is input to a control, while at the same time, a RangeValidator can be used to ensure that the input is within a specified data range. The CustomValidator control allows you to define your own validation criteria. Since the error message can be displayed in the validation control, you can control where the message is displayed on the web page by placing the validation control at the desired location. You can also display a summary of the results from all of a page's validation controls by using the ValidationSummary control.

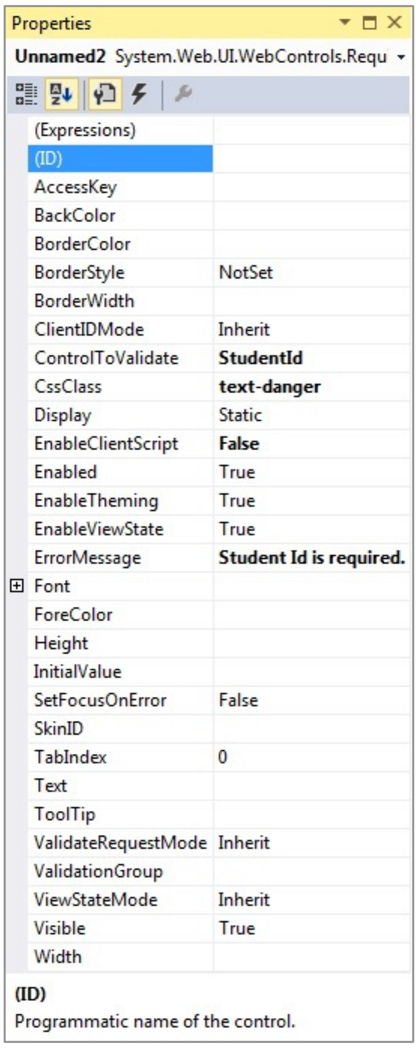
By default, page validation is performed when a button control, such as Button, ImageButton or LinkButton, is clicked. You can prevent validation from being performed by setting the CausesValidation property of the control to "False". This property is normally set to false for a cancel or clear button to prevent validation from being performed when the button is clicked.

The following properties apply to all validation server controls.

- *ControlToValidate* – The ID of the input control to validate.
- *CssClass* – The Css class to use to display the error message.
- *Display* – The display behaviour of the validation control:

None –	The validation control is never displayed inline.
Static –	The space for the error message is allocated on the web page even if the input control passes validation. Multiple validation controls for the same input control must occupy different physical locations on the page.
Dynamic –	The space for the error message is allocated dynamically on the web page when validation fails. Multiple validation controls for the same input control can share the same physical page location.

- *EnableClientScript* – Indicates whether client-side validation is enabled.
- *Enabled* – Indicates whether the validation control is enabled.
- *ErrorMessage* – The error message to display in the ValidationSummary control if validation fails. If the Text property of the validation control is not set, this text is also displayed in the validation control when validation fails.
- *ForeColor* – Specifies the colour used to display the inline message when validation fails (superseded by CssClass value).
- *IsValid* – Indicates whether the input control specified by the ControlToValidate property is determined to be valid.
- *Text* – The message to display in the validation control when validation fails. If this property is not set, the text specified in the ErrorMessage property is displayed in the control.
- *Type* – For validation controls that perform typed comparisons, specifies the data type of the values being compared. The values are converted to this data type before the comparison is made.



(Expressions)	
(ID)	
AccessKey	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
ClientIDMode	Inherit
ControlToValidate	**StudentId**
CssClass	**text-danger**
Display	Static
EnableClientScript	**False**
Enabled	True
EnableTheming	True
EnableViewState	True
ErrorMessage	**Student Id is required.**
Font	
ForeColor	
Height	
InitialValue	
SetFocusOnError	False
SkinID	
TabIndex	0
Text	
ToolTip	
ValidateRequestMode	Inherit
ValidationGroup	
ViewStateMode	Inherit
Visible	True
Width	

(ID)
Programmatic name of the control.

Figure 1: RequiredFieldValidator control properties for "StudentId" TextBox control.

Adding Validation Controls to a Web Form

We will demonstrate how to add several kinds of validation controls to the "StudentRegistrationPage.aspx" web form created in a previous tutorial and test them.

[4] While client-side validation can also be used to check user input before a page is submitted by writing a function in ECMAScript (JavaScript) that duplicates the logic of the server-side method, you should perform server-side validation even if you use a client-side check. Server-side validation helps prevent users from bypassing validation by disabling or changing the client script.

Control to Validate	RequiredFieldValidator Control Property and Value	
StudentId TextBox	**ID:** **ControlToValidate:** StudentId **ErrorMessage:** Student Id is required.	**CssClass:** text-danger **EnableClientScript:** False
FirstName TextBox	**ID:** **ControlToValidate:** FirstName **ErrorMessage:** First Name is required.	**CssClass:** text-danger **EnableClientScript:** False
LastName TextBox	**ID:** **ControlToValidate:** LastName **ErrorMessage:** Last Name is required.	**CssClass:** text-danger **EnableClientScript:** False
Address TextBox	**ID:** **ControlToValidate:** Address **ErrorMessage:** Address is required.	**CssClass:** text-danger **EnableClientScript:** False
UserName TextBox	**ID:** **ControlToValidate:** UserName **ErrorMessage:** User Name is required.	**CssClass:** text-danger **EnableClientScript:** False
Email TextBox	**ID:** **ControlToValidate:** Email **ErrorMessage:** Email address is required.	**CssClass:** text-danger **EnableClientScript:** False
Password Textbox	**ID:** **ControlToValidate:** Password **ErrorMessage:** Password is required.	**CssClass:** text-danger **EnableClientScript:** False
ConfirmPassword TextBox	**ID:** **ControlToValidate:** ConfirmPassword **ErrorMessage:** Confirm Password is required.	**CssClass:** text-danger **EnableClientScript:** False
Country DropDownList	**ID:** **ControlToValidate:** Country **ErrorMessage:** Please select a country. **InitialValue:** -- Select country --	**CssClass:** text-danger **EnableClientScript:** False
State DropDownList	**ID:** **ControlToValidate:** State **ErrorMessage:** Please select a state/province. **InitialValue:** -- Select state/province --	**CssClass:** text-danger **EnableClientScript:** False

Table 2: “StudentRegistrationPage.aspx” web form RequiredFieldValidator control properties.

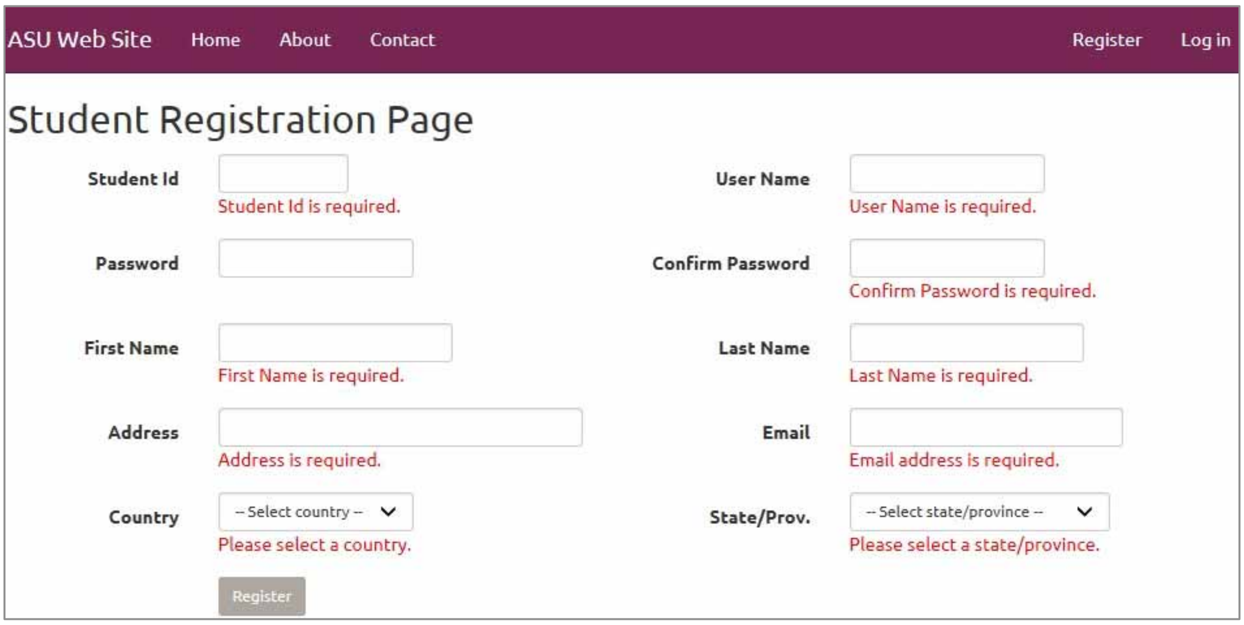
Adding RequiredFieldValidator Controls

The RequiredFieldValidator control makes an input control a mandatory field. The input control fails validation if the value it contains does not change from its initial value when validation is performed. This prevents the user from leaving the input control unchanged. By default, the initial value is an empty string (""), which indicates that a value must be entered in the input control for it to pass validation. Extra spaces at the beginning and end of the input value are removed before validation is performed to prevent a space being entered in the input control from passing validation.

For the “StudentRegistrationPage.aspx” web form, all fields require that values be entered. Therefore, add RequiredFieldValidator controls to the web form by doing the following.

1. Add a RequiredFieldValidator control from the `Toolbox` Validation palette to the right of each TextBox control.
2. Set the properties for each validation control to that shown in Table 2. Figure 1 shows the properties set for the RequiredFieldValidator control for the “StudentId” TextBox control.

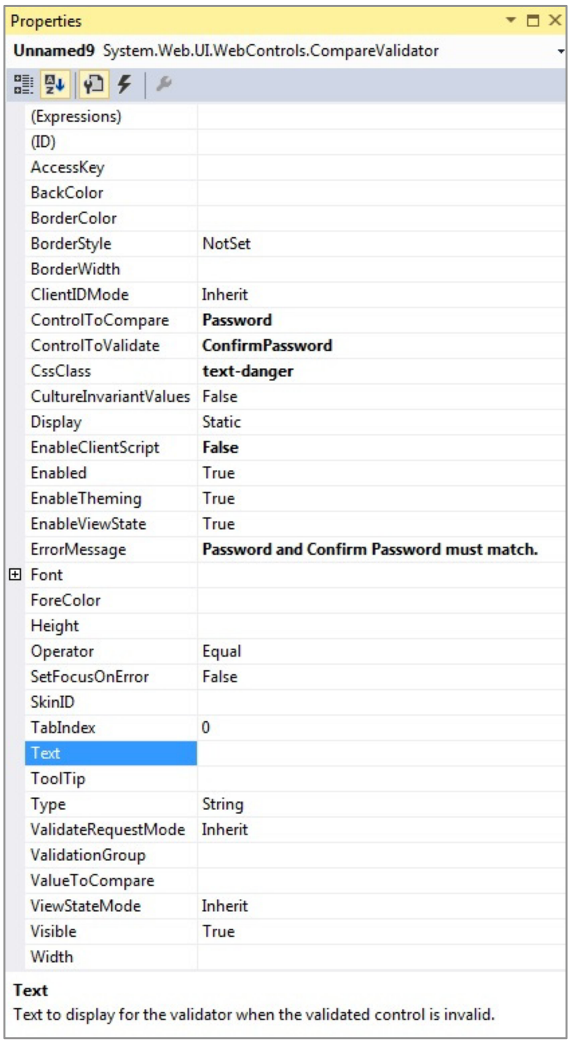
Although there are initial values for the “Country” and “State” DropDownList controls, these are not valid country and state values. Therefore, it is required that another value be selected from their respective dropdown lists. To specify this requirement, do the following.



Student Registration Page

Student Id	Student Id is required.	User Name	User Name is required.
Password		Confirm Password	Confirm Password is required.
First Name	First Name is required.	Last Name	Last Name is required.
Address	Address is required.	Email	Email address is required.
Country	-- Select country -- Please select a country.	State/Prov.	-- Select state/province -- Please select a state/province.

Figure 2: “StudentRegistrationPage.aspx” web form showing RequiredFieldValidator control messages.



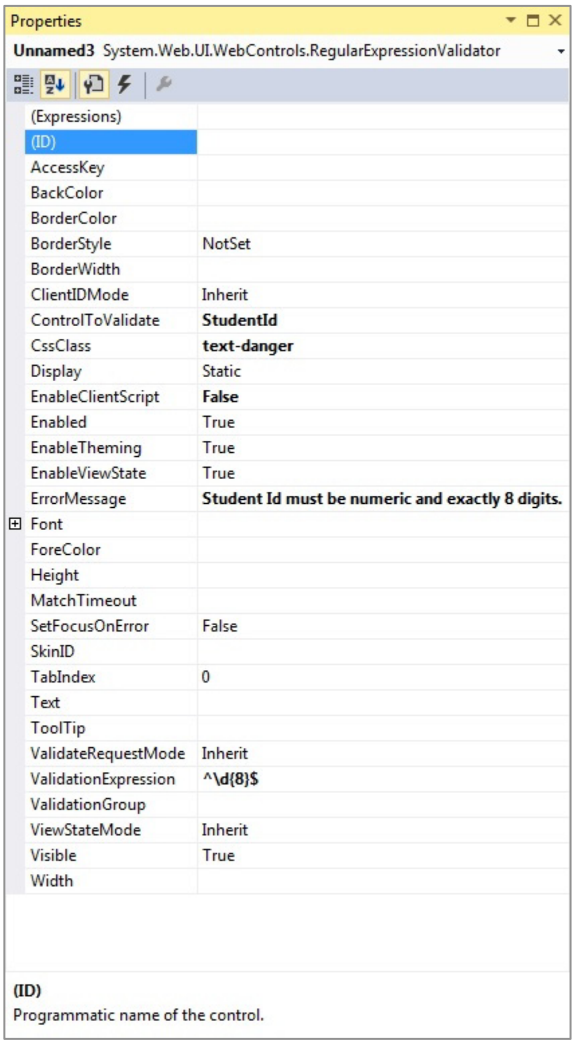
Properties

Unnamed9 System.Web.UI.WebControls.CompareValidator

(Expressions)	
(ID)	
AccessKey	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
ClientIDMode	Inherit
ControlToCompare	**Password**
ControlToValidate	**ConfirmPassword**
CssClass	**text-danger**
CultureInvariantValues	False
Display	Static
EnableClientScript	**False**
Enabled	True
EnableTheming	True
EnableViewState	True
ErrorMessage	**Password and Confirm Password must match.**
⊞ Font	
ForeColor	
Height	
Operator	Equal
SetFocusOnError	False
SkinID	
TabIndex	0
Text	
ToolTip	
Type	String
ValidateRequestMode	Inherit
ValidationGroup	
ValueToCompare	
ViewStateMode	Inherit
Visible	True
Width	

Text
Text to display for the validator when the validated control is invalid.

(a) CompareValidator control to compare “Password” and “ConfirmPassword” controls.



Properties

Unnamed3 System.Web.UI.WebControls.RegularExpressionValidator

(Expressions)	
(ID)	
AccessKey	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
ClientIDMode	Inherit
ControlToValidate	**StudentId**
CssClass	**text-danger**
Display	Static
EnableClientScript	**False**
Enabled	True
EnableTheming	True
EnableViewState	True
ErrorMessage	**Student Id must be numeric and exactly 8 digits.**
⊞ Font	
ForeColor	
Height	
MatchTimeout	
SetFocusOnError	False
SkinID	
TabIndex	0
Text	
ToolTip	
ValidateRequestMode	Inherit
ValidationExpression	**^\d{8}$**
ValidationGroup	
ViewStateMode	Inherit
Visible	True
Width	

(ID)
Programmatic name of the control.

(b) RegularExpressionValidator control to validate “StudentId” TextBox control.

Figure 3: Validation control properties.

1. Add a RequiredFieldValidator control to the left of each DropDownList control.
2. Set their properties as shown in Table 2[5].

By setting the InitialValue property of the DropDownList control, if the DropDownList control is still set to this initial value, then validation will fail causing the error message to display as shown in Figure 2. The RequiredFieldValidator controls should look as shown in Figure 2.

Save and test the Student Registration Page. If you click the Register button without entering a value in any of the required fields, the text in the ErrorMessage property will display in the RequiredFieldValidator control for that field as shown in Figure 2.

Adding CompareValidator Controls

The CompareValidator control evaluates the value of an input control against a constant value or the value of another input control to determine whether the two values match the relationship specified by a comparison operator. The CompareValidator control can also be used to determine whether the value entered into an input control can be converted to the data type specified by the Type property[6].

For the "StudentRegistrationPage.aspx" web form, we want the "Password" and "Confirm Password" TextBox controls to contain identical values. To meet this requirement, add a CompareValidator control to the right of the RequiredFieldValidator control of the "Confirm Password" TextBox control and set its properties as shown in Table 3 (see also Figure 3(a)).

Save and test the page again. Enter different values in the "Password" and "Confirm Password" textboxes and notice the result.

Property	Value
ID	
ControlToCompare	Password
ControlToValidate	ConfirmPassword
CssClass	text-danger
EnableClientScript	False
ErrorMessage	Password and Confirm Password do not match.

Table 3: Properties of CompareValidator control for "ConfirmPassword" TextBox control.

Adding RegularExpressionValidator Controls

The RegularExpressionValidator control determines whether the value of an input control matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in HKID numbers, email addresses, telephone numbers, postal codes, etc.

For the student registration page, we want the following.

- The "StudentId" TextBox control should contain only digits and have exactly eight digits.

To meet these requirements, add a RegularExpressionValidator control to the right of the RequiredFieldValidator control of the "StudentId" TextBox control and set its properties as shown in Table 4 (see also Figure 3(b)).

TextBox Control	Property	Value
StudentId	ID	
	ControlToValidate	StudentId
	CssClass	text-danger
	EnableClientScript	False
	ErrorMessage	Student Id must be numeric and exactly 8 digits.
	ForeColor	Red
	ValidationExpression	^\d{8}$

Table 4: Properties of RegularExpressionValidator controls.

Save and test the page. Each control should be validated properly. View the source HTML from the browser. Note the client-side code that has been generated by the validation controls inside the `<script>` tags.

[5] If the EnableClientScript property is not set to "False" for all validation controls, then any validation code for a CustomValidator control in the code-behind file will not be called.
[6] By default the Type property is set to "String" and the Operator to "Equal".

Adding a CustomValidator Control

The CustomValidator control creates a validation control with customized validation logic that can be placed in the code-behind file for a web form.

For the "StudentRegistrationPage.aspx" web form, we want to do the following additional validation, which will require custom validators.

- The password should have at least 6 characters and should contain at least one non-alphanumeric character.
- Since student id should be unique, it should not be allowed to add a record with an existing student id.

For now, we will only add a CustomValidator control for the "Password" TextBox control. Adding the CustomValidator for the "StudentId" TextBox control will be done in the section Adding Validation Controls to Editing and Inserting Interfaces.

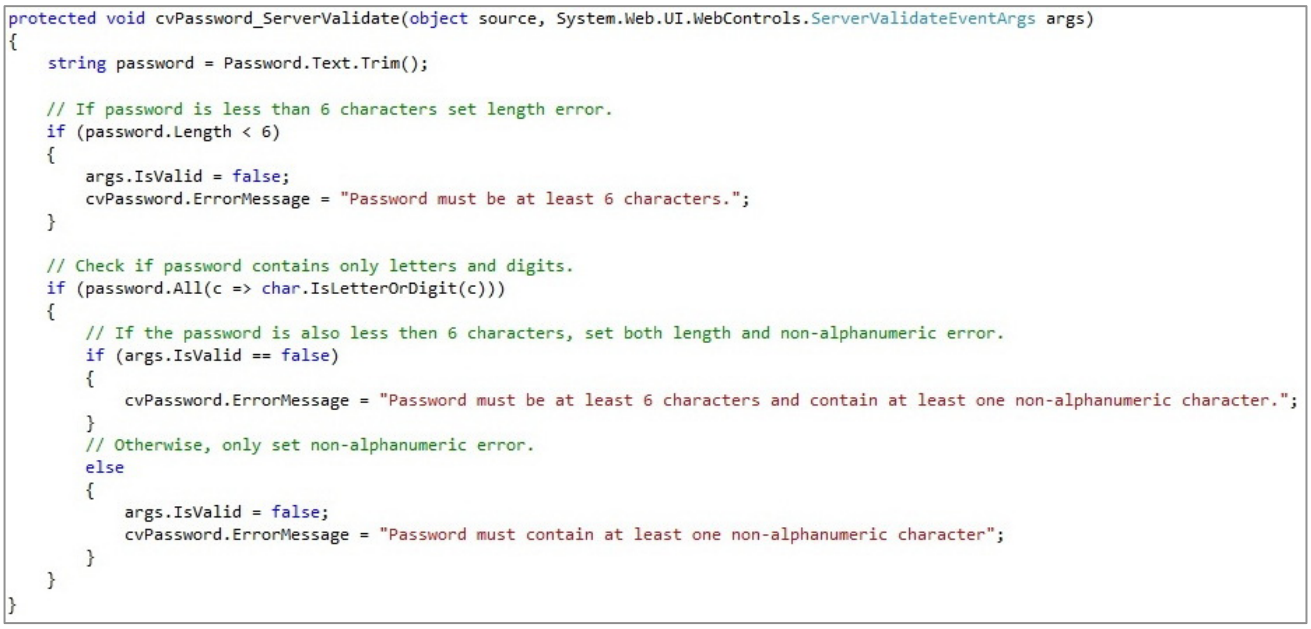
To add the CustomValidator control validation code for the "Password" TextBox control, do the following.

1. Add a CustomValidator control to the right of the "Password" TextBox control.
2. Set its properties as shown in Table 5.
3. Download the "ASPNETValidationTutorial-Download.zip" file from the Tutorial and Lab Schedule section of the course web page and unzip it.
4. Double-click the "cvPassword" CustomValidator control and add the code in the "cvPassword_ServerValidate.txt" file into the "cvPassword_ServerValidate" event handler. This code, shown in Figure 4, validates the password and sets the IsValid property of the args ServerValidateEventArgs object to "False" if validation fails. (Its default value is "True".)

Property	**Value**
ID	cvPassword
ControlToValidate	Password
CssClass	text-danger
EnableClientScript	False
ErrorMessage	Password should have at least 6 characaters.
ValidateEmptyText[7]	True

Table 5: Properties of CustomValidator control for "Password" TextBox control.

Save and test the page. The "Password" textbox should not allow an empty password, a password of less than 6 characters or a password that does not contain a non-alphanumeric character as shown in Figure 5[8].

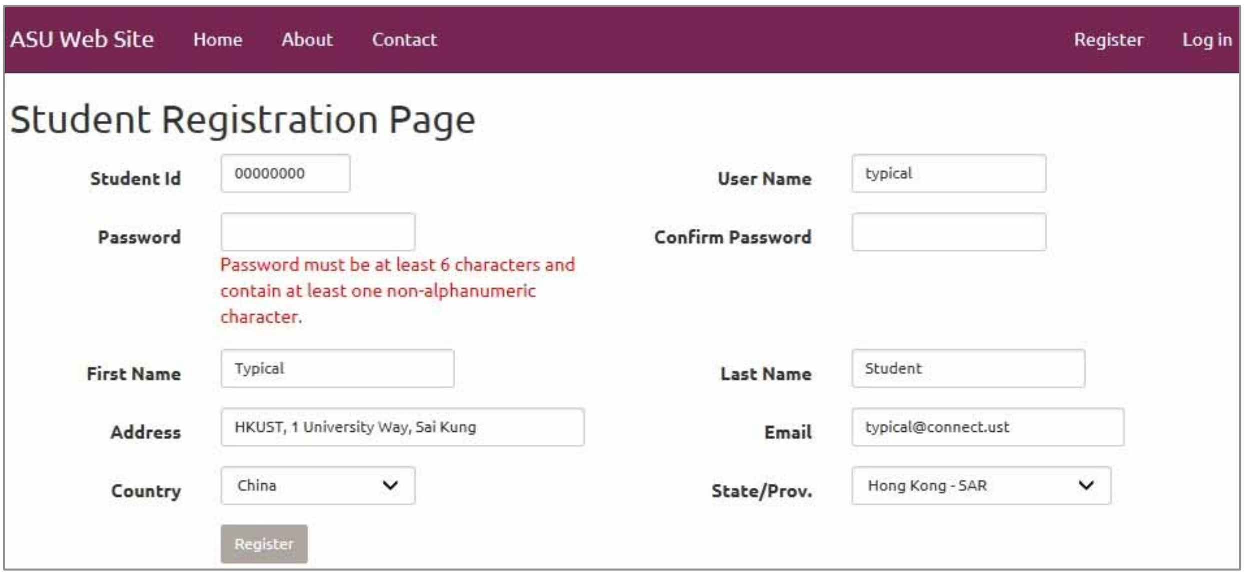


```
protected void cvPassword_ServerValidate(object source, System.Web.UI.WebControls.ServerValidateEventArgs args)
{
    string password = Password.Text.Trim();

    // If password is less than 6 characters set length error.
    if (password.Length < 6)
    {
        args.IsValid = false;
        cvPassword.ErrorMessage = "Password must be at least 6 characters.";
    }

    // Check if password contains only letters and digits.
    if (password.All(c => char.IsLetterOrDigit(c)))
    {
        // If the password is also less then 6 characters, set both length and non-alphanumeric error.
        if (args.IsValid == false)
        {
            cvPassword.ErrorMessage = "Password must be at least 6 characters and contain at least one non-alphanumeric character.";
        }
        // Otherwise, only set non-alphanumeric error.
        else
        {
            args.IsValid = false;
            cvPassword.ErrorMessage = "Password must contain at least one non-alphanumeric character";
        }
    }
}
```

Figure 4: Code to validate the "Password" TextBox control.

[7] Setting the ValidateEmptyText property to "True" prevents an empty password from being entered.

[8] You will notice that when you select a value in one of the DropDownLists, the "Password" and Confirm Password" fields are cleared. This is done by design in ASP.NET to protect sensitive data from being disclosed in the HTML of a web form.



Student Registration Page

Student Id	00000000
User Name	typical
Password	
Confirm Password	
First Name	Typical
Last Name	Student
Address	HKUST, 1 University Way, Sai Kung
Email	typical@connect.ust
Country	China
State/Prov.	Hong Kong - SAR

Password must be at least 6 characters and contain at least one non-alphanumeric character.

Figure 5: Custom validation for the “Password” TextBox control.

Adding a ValidationSummary Control

The ValidationSummary control summarizes the error messages from all the validation controls on a web page either in a single location on the web page or in a browser message box. Add a ValidationSummary control at the top of the “StudentRegistrationPage.aspx” web form below the first `<div>` tag and set its properties as follows.

Property	Value
CssClass	text-danger
DisplayMode	BulletList
EnableClientScript	False
ShowSummary	True

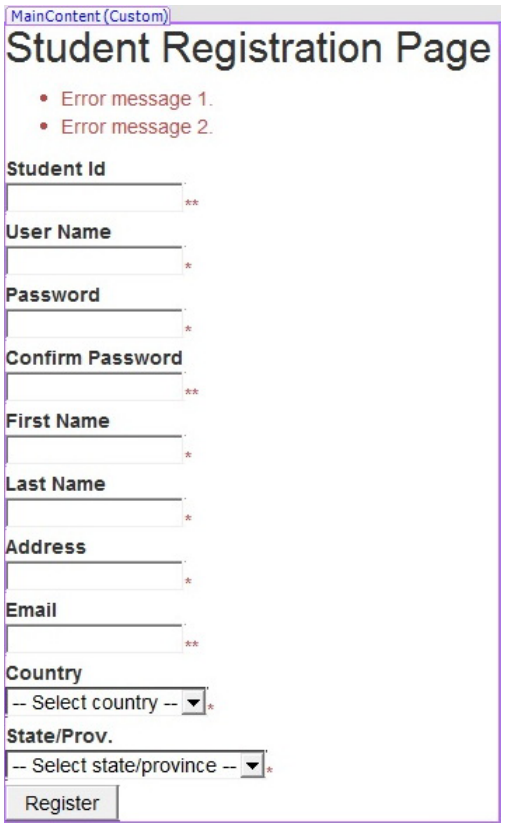
Set the Text property to “*” for all of the validation controls as shown in Figure 6.

Viewing the Web Form

Save the web form and test the page again. All error messages now appear in the summary area and the problematic fields have a “*” below them as shown in Figure 7(a).

Set the ShowMessageBox property on the ValidationSummary control to “True”, the ShowSummary property to “False” and set EnableClientScript to “True” for all validation controls[9].

Save the web form and test the page again. A message box now pops up with the error messages while the summary area has no error messages as shown in Figure 7(b).

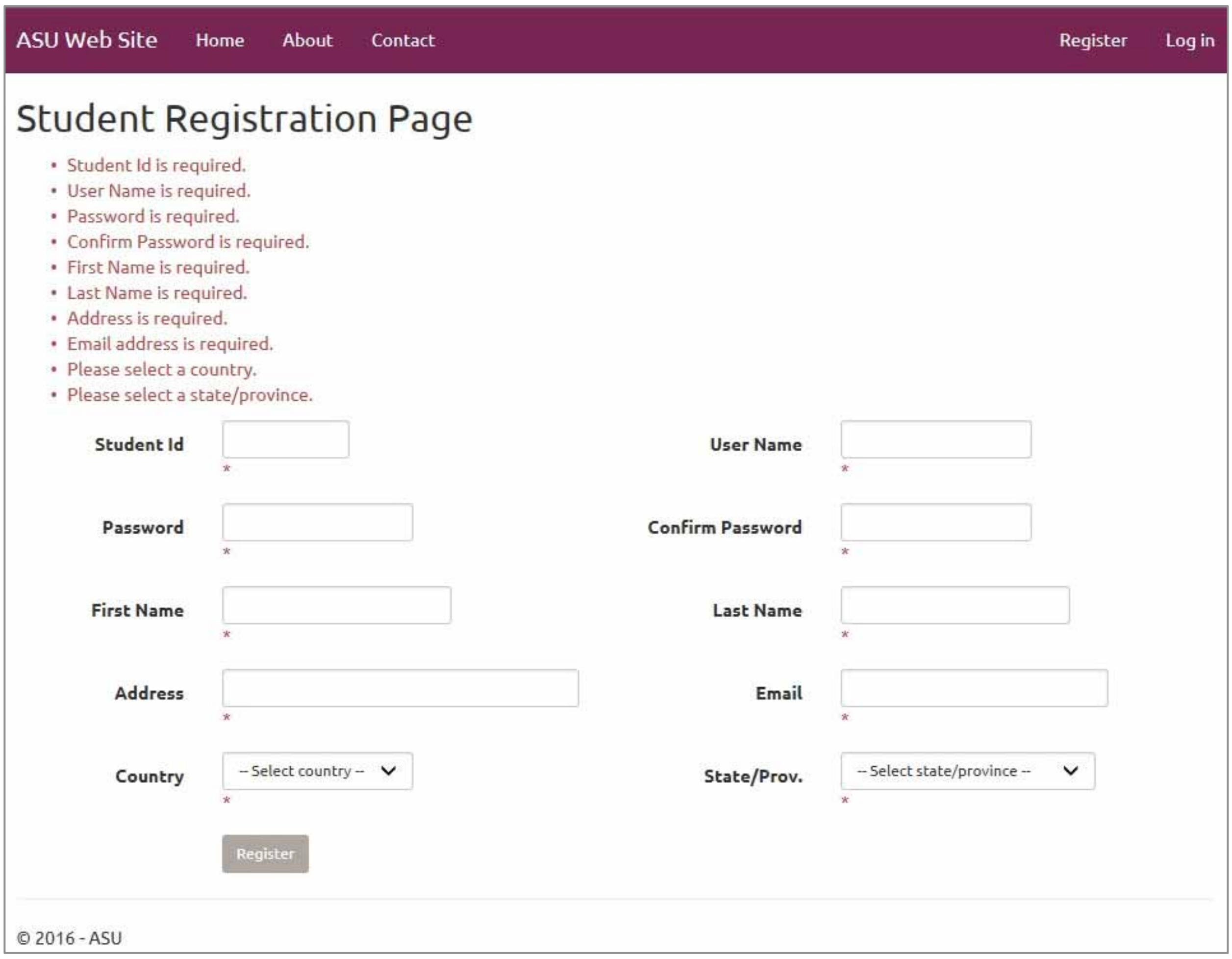


Student Registration Page

- Error message 1.
- Error message 2.

Figure 6: Adding a ValidationSummary control.

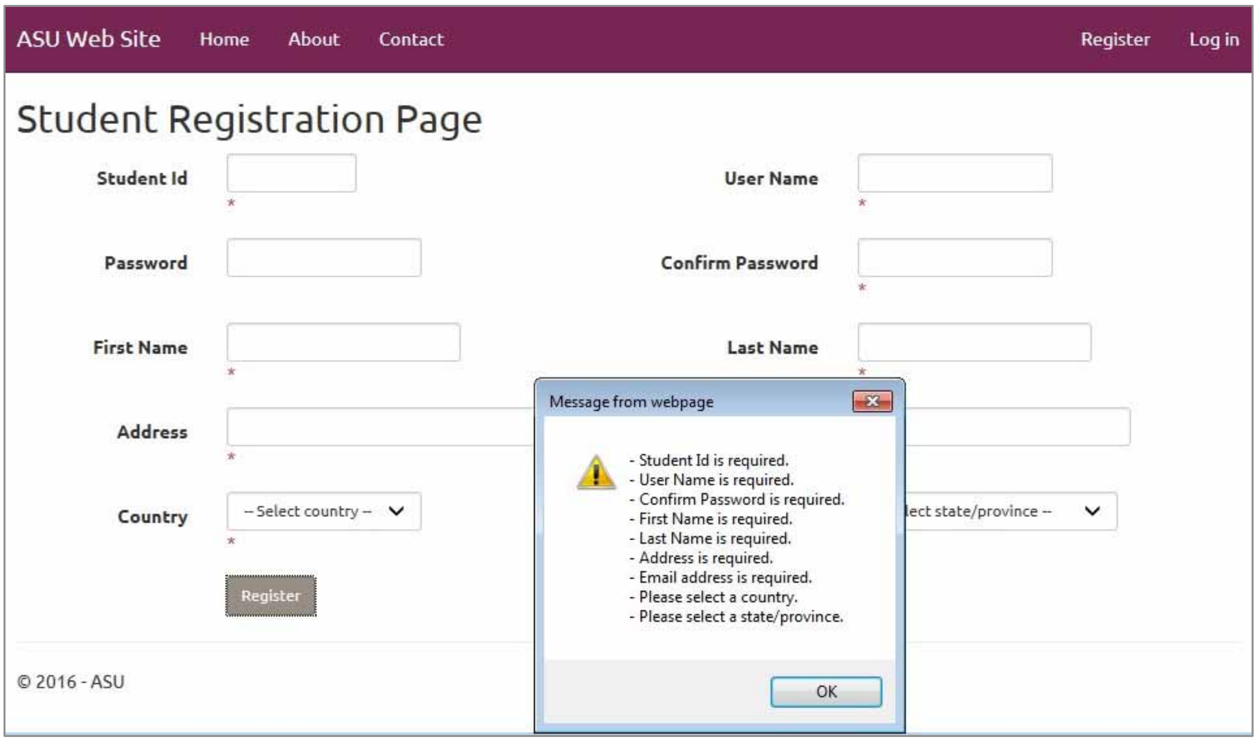
[9] It is a known bug in ASP.NET that the message box will not display if EnableClientScript is set to “False” for any validation control. On the other hand, if EnableClientScript is set to “True” for any validation control, then any custom validation code in the code-behind file will not be called. For example, while the “Password” TextBox control is validated by the custom validation code in Figure 7(a), when EnableClientScript is set to “False”, it is not validated when EnableClientScript is set to “True” as is the case in Figure 7(b). Therefore, if you are using a ValidationSummary control and have any custom validation code, EnableClientScript should be set to “False” for all validation controls, ShowMessageBox should be set to “False” and ShowSummary should be set to “True”.



Student Registration Page

- Student Id is required.
- User Name is required.
- Password is required.
- Confirm Password is required.
- First Name is required.
- Last Name is required.
- Address is required.
- Email address is required.
- Please select a country.
- Please select a state/province.

(a) Using the ValidationSummary control.



- Student Id is required.
- User Name is required.
- Confirm Password is required.
- First Name is required.
- Last Name is required.
- Address is required.
- Email address is required.
- Please select a country.
- Please select a state/province.

(b) Using a message box.

Figure 7: Displaying error messages.

ADDING VALIDATION CONTROLS TO EDITING AND INSERTING INTERFACES[10]

Most GridView and DetailsView controls we have used in the tutorial and lab notes have been composed of BoundFields. When editing a row in a GridView or DetailsView control, BoundFields that are not read-only are converted into textboxes from which the user can modify the existing data. Similarly, when inserting a new record into a DetailsView control, BoundFields whose InsertVisible property is set to True (the default) are rendered as empty textboxes into which the user can provide the new record's field values.

While the default editing and inserting interfaces for a BoundField can be helpful, they lack any sort of validation. If a user makes a data-entry mistake, such as omitting the value for the student id field or entering an invalid value for a course code, an exception will be raised. While this exception can be gracefully handled, ideally the editing or inserting interface should include validation controls to prevent a user from entering such invalid data in the first place.

In order to provide a customized editing or inserting interface, we need to replace a BoundField with a TemplateField[11]. TemplateFields can consist of multiple templates defining separate interfaces for different row states (e.g., editing or inserting states). The TemplateField's ItemTemplate is used when rendering read-only fields or rows in the GridView or DetailsView controls, whereas the TemplateField's EditItemTemplate and InsertItemTemplate render the interfaces to use for the editing and inserting modes, respectively. Validation controls can be added to a TemplateField's EditItemTemplate and InsertItemTemplate to provide a more robust and secure user interface.

Converting BoundFields into TemplateFields

To add validation controls to editing and inserting interfaces, the BoundFields used by the GridView and DetailsView controls need to be converted into TemplateFields. For the “StudentRecords.aspx” web form, we will add validation controls to the “gvStudentRecords” GridView control and the “dvStudentRecords” DetailsView control. Before doing so, make the following changes to the “gvStudentRecords” GridView control and the “dvStudentRecords” DetailsView control.

1. Uncheck “Enable Selection” and check “Enable Editing” and “Enable Deleting” in the smart tag drop down menu of the “gvStudentRecords” GridView control.
2. Uncheck “Enable Editing” and “Enable Deleting” in the smart tag drop down menu of the “dvStudentRecords” DetailsView control.
3. In the “dvStudentRecords” DetailsView control’s property window, set the DefaultMode property to “Insert” by selecting it from the property’s dropdown list.

With this change, the GridView control provides the interface for editing and deleting student records while the DetailsView control provides the interface only for inserting a new student record.

To convert the BoundFields in the “gvStudentRecords” GridView control into TemplateFields, do the following.

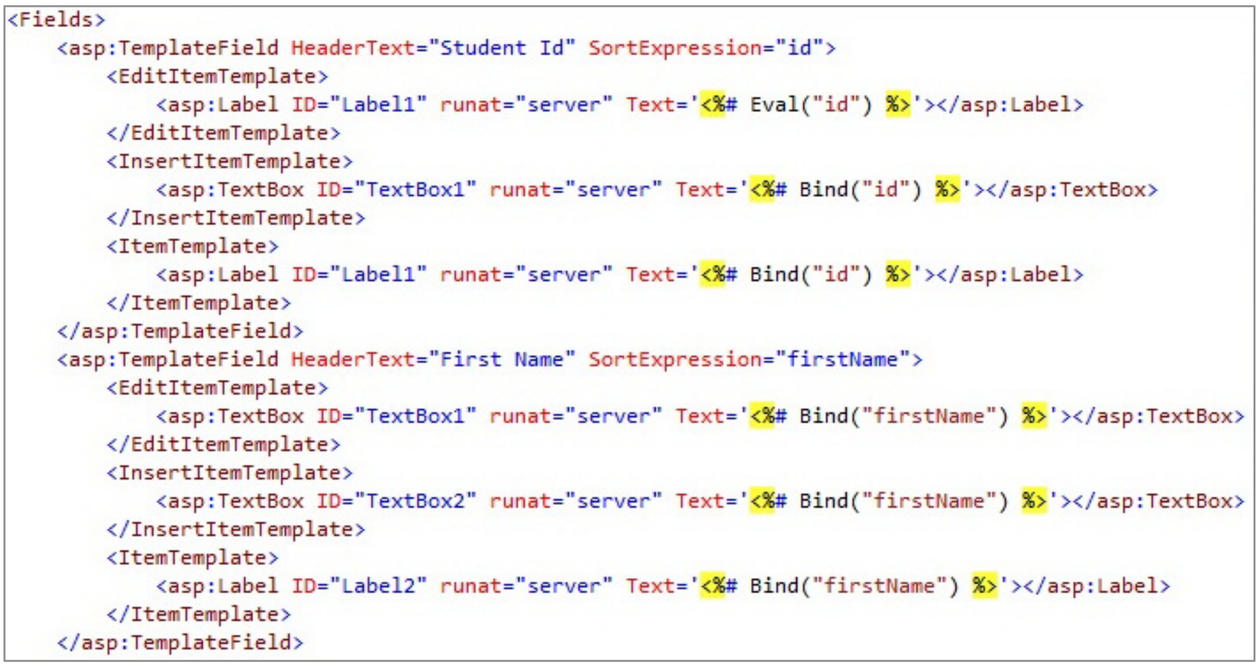
1. Select “Edit Columns…” in the smart tag drop down menu of the “gvStudentRecords” GridView control.
2. Select each of the BoundFields “First Name”, “Last Name”, “Address”, “State” and “Country” in the *Selected fields:* pane of the `Fields` window and, for each field, click the “Convert this field into a TemplateField” link[12].

Similarly, to convert the BoundFields in the “dvStudentRecords” DetailsView control into TemplateFields, do the following.

1. Select “Edit Fields…” in the smart tag drop down menu of the “dvStudentRecords” DetailsView control.
2. Select each of the BoundFields “Student Id”, “First Name”, “Last Name”, “Address”, “State” and “Country” in the *Selected fields:* pane of the `Fields` window and, for each field, click the “Convert this field into a TemplateField” link.

If you inspect the HTML in the Source tab of the `Document` window you will see that for the “gvStudentRecords” GridView control each TemplateField has two templates automatically created, an ItemTemplate and an EditItemTemplate. For the “dvStudentRecords” DetailsView control each TemplateField has three templates created, an ItemTemplate, an EditItemTemplate and an InsertItemTemplate as shown in Figure 8. The ItemTemplate displays a single data-field value for an attribute using a Label control, while the EditItemTemplate and InsertItemTemplate present the data-field value in a TextBox control that associates the data field with the TextBox's Text property using two-way data-binding.

[10] Adapted from Tutorial 19: Adding Validation Controls to the Editing and Inserting Interfaces.
[11] A TemplateField is used by data-bound controls, such as GridView and DetailsView, to display custom content that is not provided by one of the predefined data control fields.
[12] The “StudentId” BoundField is not converted into a TemplateField since it is the primary key of the *Student* relation and so cannot be edited.



```
<Fields>
    <asp:TemplateField HeaderText="Student Id" SortExpression="id">
        <EditItemTemplate>
            <asp:Label ID="Label1" runat="server" Text='<%# Eval("id") %>'></asp:Label>
        </EditItemTemplate>
        <InsertItemTemplate>
            <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("id") %>'></asp:TextBox>
        </InsertItemTemplate>
        <ItemTemplate>
            <asp:Label ID="Label1" runat="server" Text='<%# Bind("id") %>'></asp:Label>
        </ItemTemplate>
    </asp:TemplateField>
    <asp:TemplateField HeaderText="First Name" SortExpression="firstName">
        <EditItemTemplate>
            <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("firstName") %>'></asp:TextBox>
        </EditItemTemplate>
        <InsertItemTemplate>
            <asp:TextBox ID="TextBox2" runat="server" Text='<%# Bind("firstName") %>'></asp:TextBox>
        </InsertItemTemplate>
        <ItemTemplate>
            <asp:Label ID="Label2" runat="server" Text='<%# Bind("firstName") %>'></asp:Label>
        </ItemTemplate>
    </asp:TemplateField>
```

Figure 8: Templates added to HTML code for TemplateFields of the "dvStudentRecords" DetailsView control.

Adding Validation Controls to a GridView's and DetailsView's TemplateFields

Validation controls can be added to the fields in an EditItemTemplate and InsertItemTemplate in a GridView or DetailsView control by editing the templates.

To add editing validation controls to the controls in the "gvStudentRecords" GridView control, do the following.

1. Select "Edit Templates" in the "gvStudentRecords" GridView control's smart tag drop down menu to bring up the template-editing interface shown in Figure 9.
2. For each field shown in Table 6[13] select "EditItemTemplate" in the *Display:* field of the GridView Tasks smart tag drop down menu shown in Figure 9 and do the following.

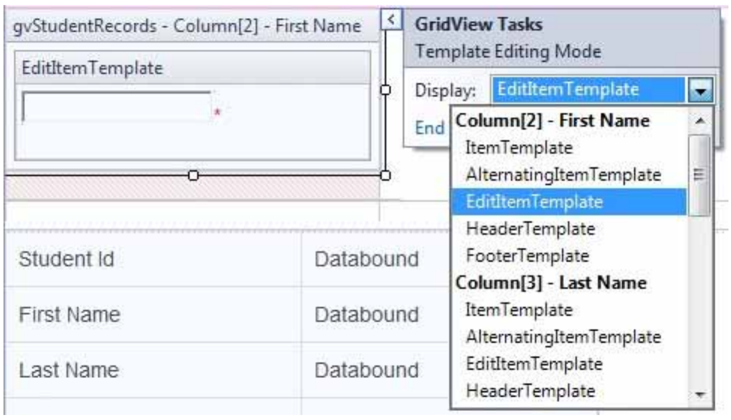


Figure 9: Template-editing interface.

   a. Change the ID and the MaxLength properties of the TextBox in the *EditItemTemplate* field to that shown in Table 6.
   b. Add a RequiredFieldValidator control to the `EditItemTemplate` window as shown in Figure 9 by placing the cursor to the right of the TextBox control and double-clicking a RequiredFieldValidator control in the `Toolbox`.
   c. Set the properties of the RequiredFieldValidator control as shown in Table 6 and also set the following properties of each RequiredFieldValidator control.

Property	Value
CssClass	text-danger
Display	Dynamic
EnableClientScript	False
Text	*

[13] The Student Id field is represented by a Label control in the EditItemTemplate since it cannot be edited.

EditItemTemplate Field	**TextBox control**	**RequiredFieldValidator control**	
		Property	**Value**
Field[1] – First Name	**ID:** txtEditFirstName **MaxLength:** 20	ID	
		ControlToValidate	txtEditFirstName
		ErrorMessage	First Name is required.
Field[2] – Last Name	**ID:** txtEditLastName **MaxLength:** 20	ID	
		ControlToValidate	txtEditLastName
		ErrorMessage	Last Name is required.
Field[3] – Address	**ID:** txtEditAddress **MaxLength:** 50	ID	
		ControlToValidate	txtEditAddress
		ErrorMessage	Address is required.
Field[4] – State/Province	**ID:** txtEditState **MaxLength:** 25	ID	
		ControlToValidate	txtEditState
		ErrorMessage	State/Province is required.
Field[5] – Country	**ID:** txtEditCountry **MaxLength:** 25	ID	
		ControlToValidate	txtEditCountry
		ErrorMessage	Country is required.

Table 6: Properties of RequiredFieldValidator controls for “gvStudentRecords” GridView control EditItemTemplates.

InsertItemTemplate Field	**TextBox control**	**RequiredFieldValidator control**	
		Property	**Value**
Field[0] – Student Id	**ID:** txtInsertStudentId **MaxLength:** 8	ID	
		ControlToValidate	txtInsertStudentId
		ErrorMessage	Student Id is required.
Field[1] – First Name	**ID:** txtInsertFirstName **MaxLength:** 20	ID	
		ControlToValidate	txtInsertFirstName
		ErrorMessage	First Name is required.
Field[2] – Last Name	**ID: txt**InsertLastName **MaxLength:** 20	ID	
		ControlToValidate	txtInsertLastName
		ErrorMessage	Last Name is required.
Field[3] – Address	**ID:** txtInsertAddress **MaxLength:** 50	ID	
		ControlToValidate	txtInsertAddress
		ErrorMessage	Address is required.
Field[4] – State/Province	**ID:** txtInsertState **MaxLength:** 25	ID	
		ControlToValidate	txtInsertState
		ErrorMessage	State/Province is required.
Field[5] – Country	**ID:** txtInsertCountry **MaxLength:** 25	ID	
		ControlToValidate	txtInsertCountry
		ErrorMessage	Country is required.

Table 7: Properties of RequiredFieldValidator controls for “dvStudentRecords” DetailsView control’s InsertItemTemplates.

To add inserting validation controls to the fields in the “dvStudentRecords” DetailsView control, do the following.

1. Select “Edit Templates” in the drop down menu of the “dvStudentRecords” DetailsView control to bring up the template-editing interface.

2. For each field shown in Table 7 select its "InsertItemTemplate" in the *Display:* field of the DetailsView Tasks smart tag drop down menu and do the following.
   a. Change the ID and the MaxLength properties of the TextBox in the *InsertItemTemplate* field to that shown in Table 7.
   b. Add a RequiredFieldValidator control to the `InsertItemTemplate` window by placing the cursor to the right of the TextBox control and double-clicking a RequiredFieldValidator control in the `Toolbox`.
   c. Set the properties of the RequiredFieldValidator control as shown in Table 7 and also set the following properties of the RequiredFieldValidator control.

Property	**Value**
CssClass	text-danger
Display	Dynamic
EnableClientScript	False
Text	*

3. For the "Student Id" field, add a RegularExpressionValidator control and set its properties as follows.

Property	**Value**
ID	
ControlToValidate	txtInsertStudentId
CssClass	text-danger
Display	Dynamic
EnableClientScript	False
ErrorMessage	Student Id must be numeric and exactly 8 digits.
Text	*
ValidationExpression	^\d{8}$

4. For the "Student Id" field, add a CustomValidator control and do the following.

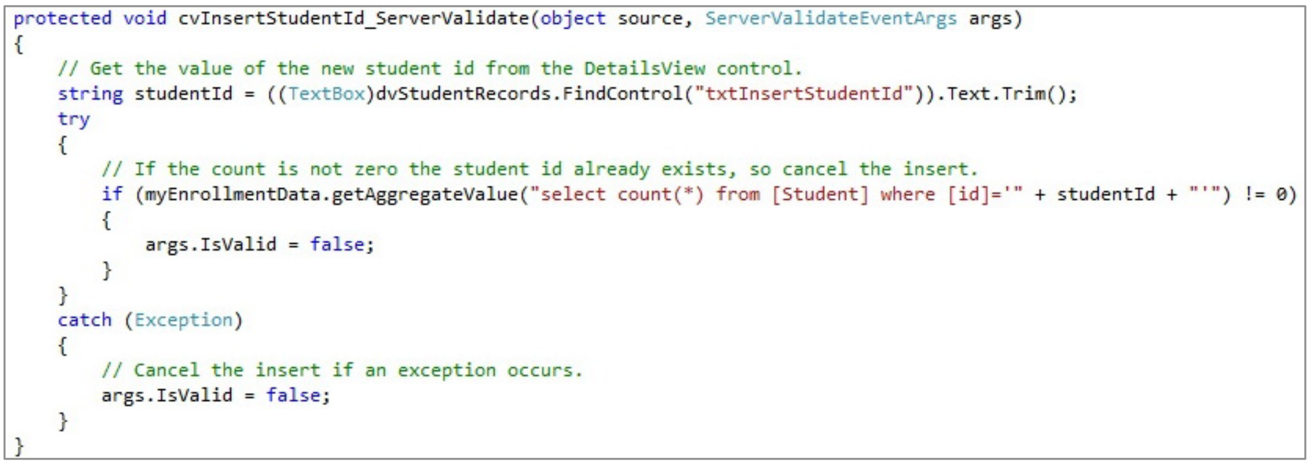
Property	**Value**
ID	cvInsertStudentId
ControlToValidate	txtInsertStudentId
CssClass	text-danger
Display	Dynamic
EnableClientScript	False
ErrorMessage	The student id already exists.
Text	*

5. Add the following line of code at the top of the "StudentRecords.aspx.cs" code-behind file.
```
using ASUWebApplication.Code_File;
```
6. Add the following line of code following the first "{" under the line "`public partial class StudentRecords : System.Web.UI.Page`".
```
EnrollmentData myEnrollmentData = new EnrollmentData();
```
7. Double-click the "cvInsertStudentId" CustomValidator control in the *InsertItemTemplate* and add the code from the file "cvInsertStudentId_ServerValidate.txt" in the "ASPNETValidationTutorial-Download" folder into the "cvInsertStudentId_ServerValidate" event handler in the code-behind file as shown in Figure 10.
8. Select "End Template Editing" in the DetailsView Tasks smart tag drop down menu.

Property	**Value**
CssClass	text-danger
DisplayMode	BulletList
EnableClientScript	False
HeaderText	The following errors occurred:
ShowSummary	True

Table 8: Properties of ValidationSummary control.

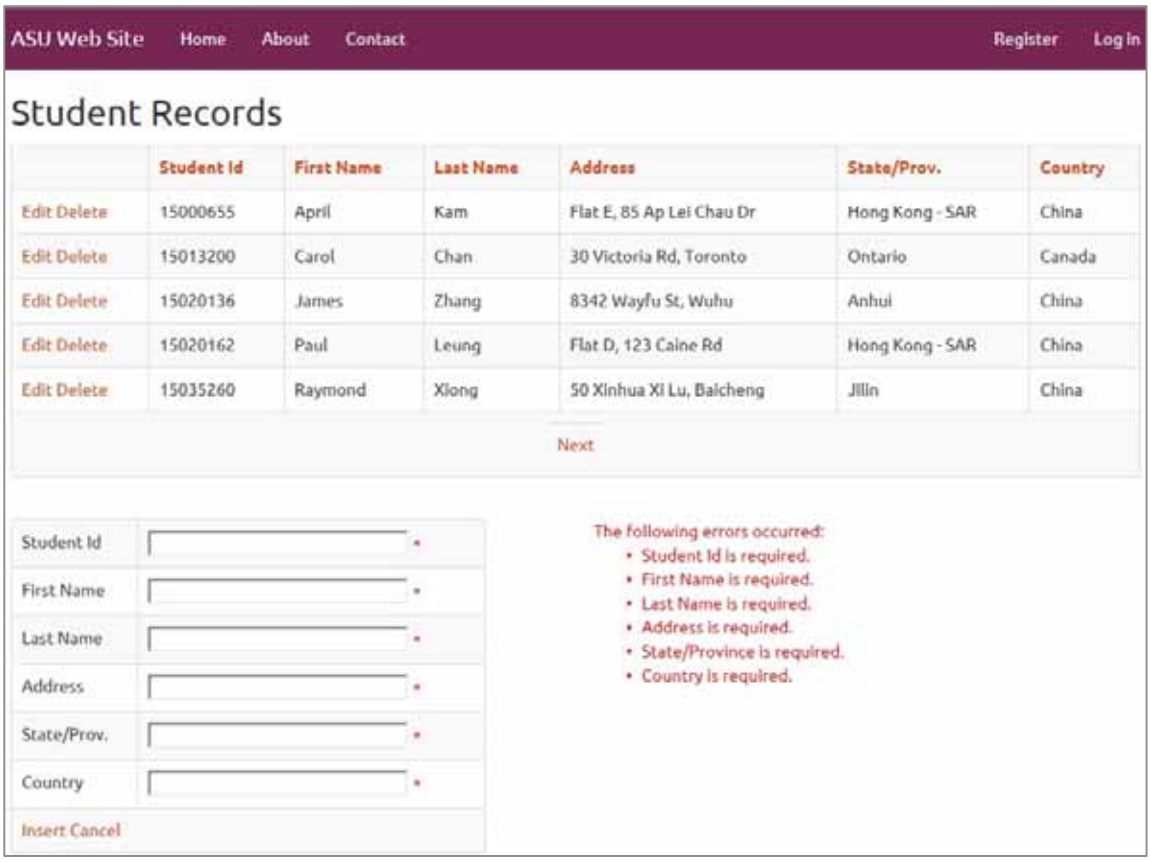
Add a ValidationSummary control after the "dvStudentRecords" DetailsView control and set its properties as shown in Table 8.



```
protected void cvInsertStudentId_ServerValidate(object source, ServerValidateEventArgs args)
{
    // Get the value of the new student id from the DetailsView control.
    string studentId = ((TextBox)dvStudentRecords.FindControl("txtInsertStudentId")).Text.Trim();
    try
    {
        // If the count is not zero the student id already exists, so cancel the insert.
        if (myEnrollmentData.getAggregateValue("select count(*) from [Student] where [id]='" + studentId + "'") != 0)
        {
            args.IsValid = false;
        }
    }
    catch (Exception)
    {
        // Cancel the insert if an exception occurs.
        args.IsValid = false;
    }
}
```

Figure 10: Custom validation code to check for duplicate student id.

Save the web form and test the page. If you try to insert a student record with no values provided, the error messages appear in the summary area and the problematic fields have a “*” beside them as shown in Figure 11. If you try to insert a record with a duplicate student id, an error message appears in the summary area indicating that the student id already exists as shown in Figure 12.



Student Records

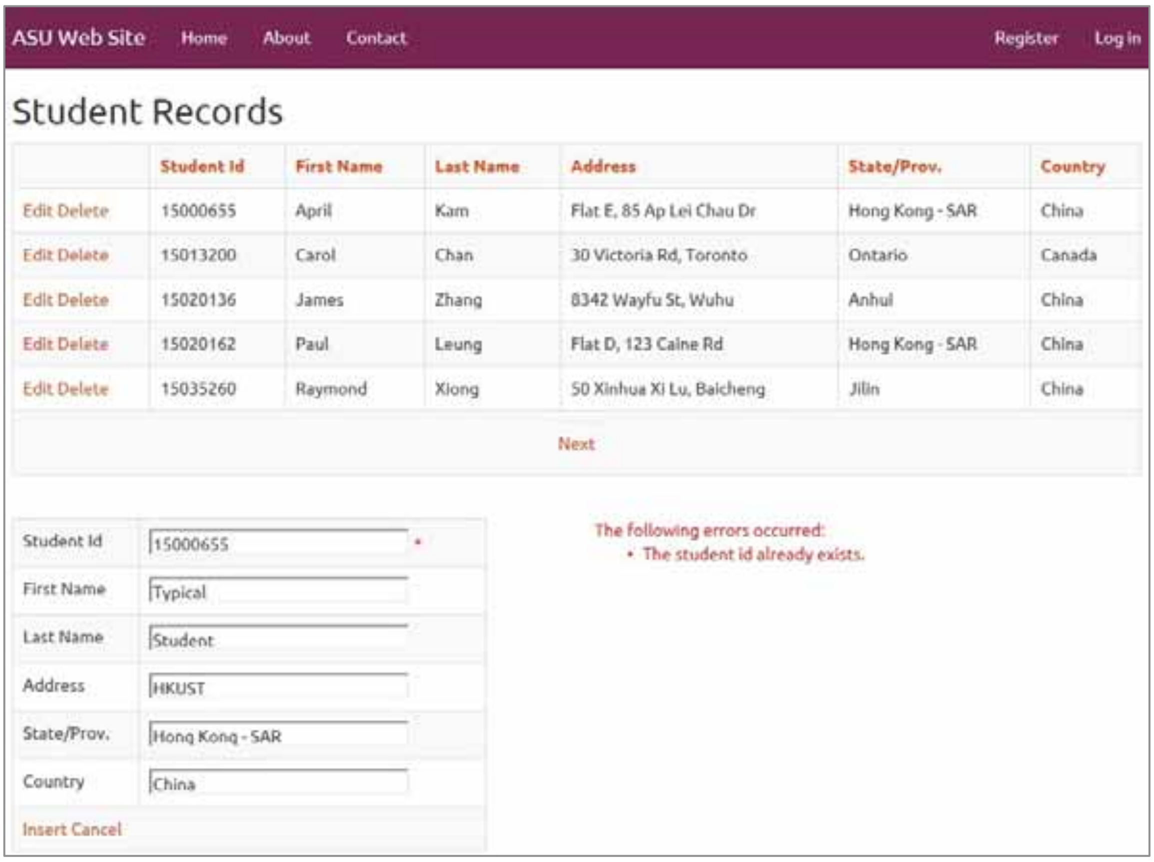
	Student Id	First Name	Last Name	Address	State/Prov.	Country
Edit Delete	15000655	April	Kam	Flat E, 85 Ap Lei Chau Dr	Hong Kong - SAR	China
Edit Delete	15013200	Carol	Chan	30 Victoria Rd, Toronto	Ontario	Canada
Edit Delete	15020136	James	Zhang	8342 Wayfu St, Wuhu	Anhui	China
Edit Delete	15020162	Paul	Leung	Flat D, 123 Caine Rd	Hong Kong - SAR	China
Edit Delete	15035260	Raymond	Xiong	50 Xinhua Xi Lu, Baicheng	Jilin	China

Next

Student Id	*
First Name	*
Last Name	*
Address	*
State/Prov.	*
Country	*
Insert Cancel	

The following errors occurred:
- Student Id is required.
- First Name is required.
- Last Name is required.
- Address is required.
- State/Province is required.
- Country is required.

Figure 11: DetailsView control showing required field error messages.



Student Records

	Student Id	First Name	Last Name	Address	State/Prov.	Country
Edit Delete	15000655	April	Kam	Flat E, 85 Ap Lei Chau Dr	Hong Kong - SAR	China
Edit Delete	15013200	Carol	Chan	30 Victoria Rd, Toronto	Ontario	Canada
Edit Delete	15020136	James	Zhang	8342 Wayfu St, Wuhu	Anhui	China
Edit Delete	15020162	Paul	Leung	Flat D, 123 Caine Rd	Hong Kong - SAR	China
Edit Delete	15035260	Raymond	Xiong	50 Xinhua Xi Lu, Baicheng	Jilin	China

Next

Student Id	15000655 *
First Name	Typical
Last Name	Student
Address	HKUST
State/Prov.	Hong Kong - SAR
Country	China
Insert Cancel	

The following errors occurred:
- The student id already exists.

Figure 12: DetailsView control showing duplicate student id error message.

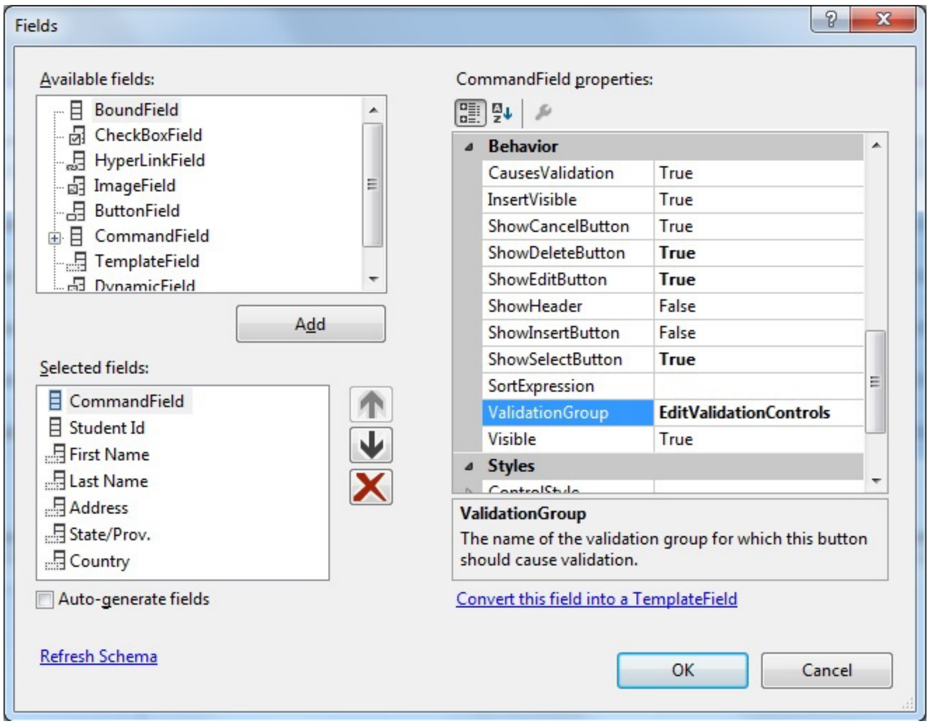
Partitioning Validation Controls into Validation Groups

The “StudentRecords.aspx” web form consists of two logically disparate sets of validation controls: those that correspond to the GridView's editing interface and those that correspond to the DetailsView's inserting interface. By default, when a postback occurs, *all* validation controls on a page are checked. However, when editing a record, we do not want the DetailsView control's inserting interface validation controls to validate. In particular, when a user is editing a record with perfectly legal values, clicking Update may cause a validation error because the values in the inserting interface are blank.

The validation controls in ASP.NET can be partitioned into validation groups through their ValidationGroup property. To associate a set of validation controls in a group, simply set their ValidationGroup property to the same value. For the “StudentRecords.aspx” web form, set the ValidationGroup properties of the validation controls of the “gvStudentRecords” GridView's TemplateFields to “EditValidationControls” and the ValidationGroup properties of the “dvStudentRecords” DetailsView's TemplateFields to “InsertValidationControls”.

In addition to the validation controls, the Button and Button-related controls in ASP.NET also include a ValidationGroup property. A validation group's validators are checked for validity only when a Button that has the same ValidationGroup property setting induces a postback. For example, in order for the “gvStudentRecords” GridView's Update button to trigger the “EditValidationControls” validation group we need to set the ValidationGroup property of the CommandField field in the `Fields` window to “EditValidationControls” as shown in Figure 13. Additionally, we need to set the DetailsView's CommandField field ValidationGroup property to “InsertValidationControls”.

The edit-specific validation controls fire only when the “gvStudentRecords” GridView's Update button is clicked and the insert-specific validation controls fire only when the “dvStudentRecords” DetailsView's Insert button is clicked. However, with this change the ValidationSummary control no longer displays when entering invalid data. The ValidationSummary control also contains a ValidationGroup property and only shows summary information for those validation controls in its validation group. Therefore, we need to have two ValidationSummary controls in this page: one for the “InsertValidationControls” validation group and one for the “EditValidationControls” validation group as shown in Figure 14.



Fields

Available fields:
- BoundField
- CheckBoxField
- HyperLinkField
- ImageField
- ButtonField
- CommandField
- TemplateField
- DynamicField

Add

Selected fields:
- CommandField
- Student Id
- First Name
- Last Name
- Address
- State/Prov.
- Country

Auto-generate fields

Refresh Schema

CommandField properties:

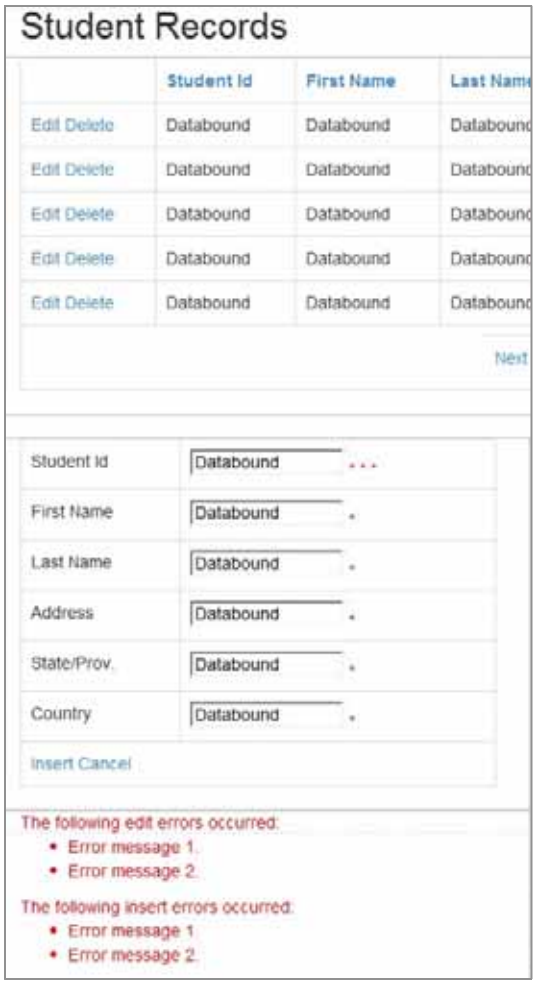
Behavior	
CausesValidation	True
InsertVisible	True
ShowCancelButton	True
ShowDeleteButton	**True**
ShowEditButton	**True**
ShowHeader	False
ShowInsertButton	False
ShowSelectButton	**True**
SortExpression	
ValidationGroup	**EditValidationControls**
Visible	True
Styles	

ValidationGroup
The name of the validation group for which this button should cause validation.

Convert this field into a TemplateField

OK Cancel

Figure 13: Validation group added to CommandField properties of the “dvStudentRecords” DetailsView control.



Student Records

	Student Id	First Name	Last Name
Edit Delete	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound

Next

Student Id	Databound
First Name	Databound
Last Name	Databound
Address	Databound
State/Prov.	Databound
Country	Databound

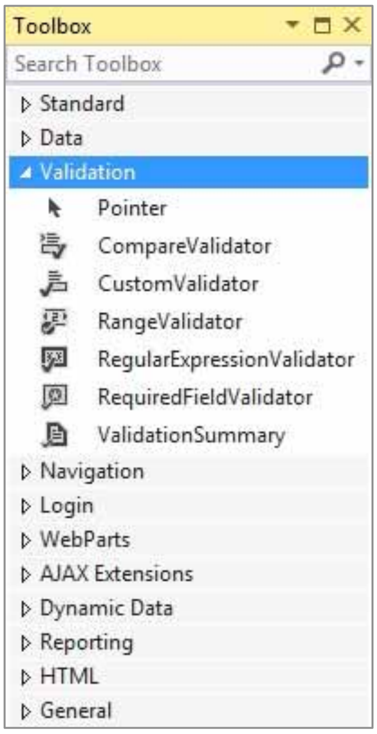
Insert Cancel

The following edit errors occurred:
- Error message 1.
- Error message 2.

The following insert errors occurred:
- Error message 1.
- Error message 2.

Figure 14: Different ValidationSummary controls for insert and edit validation groups.

APPENDIX: ASP.NET VALIDATION SERVER CONTROLS



CompareValidator

The CompareValidator control evaluates the value of an input control against a constant value (*ValueToCompare* property) or the value of another input control (*ControlToCompare* property) to determine whether the two values match the relationship specified by a comparison operator (*Operator* property). If the *Operator* property is set to ValidationCompareOperator.DataTypeCheck, the CompareValidator control ignores both the *ControlToCompare* and *ValueToCompare* properties and simply indicates whether the value entered into the input control can be converted to the data type specified by the *Type* property.

CustomValidator

The CustomValidator control creates a validation control with customized validation logic. For example, you can create a validation control that checks whether the value entered into a text box is an odd number.

RangeValidator

The RangeValidator control checks whether a user's entry is between a specified upper (*MaximumValue* property) and lower (*MinimumValue* property) boundary. Boundaries are expressed as constants. Ranges can be checked within pairs of numbers, alphabetic characters and dates.

RegularExpressionValidator

The RegularExpressionValidator control determines whether the value of an input control matches a pattern defined by the regular expression specified in the *ValidationExpression* property. This type of validation allows you to check for predictable sequences of characters, such as those in HKID numbers, email addresses, telephone numbers, postal codes, etc. The regular expression validation syntax is slightly different on the client than on the server. On the client, JScript regular expression syntax is used. On the server, Regex syntax is used. Since JScript regular expression syntax is a subset of Regex syntax, it is recommended that you use JScript regular expression syntax in order to yield the same results on both the client and the server.

RequiredFieldValidator

The RequiredFieldValidator control makes an input control a mandatory field. The input control fails validation if the value it contains does not change from its initial value, specified by the *InitialValue* property, when validation is performed. The *InitialValue* property does not set the default value for the input control. It simply indicates the value that you *do not* want the user to enter in the input control. The input control fails validation if it contains this value when validation is performed. This prevents the user from leaving the input control unchanged. By default, the initial value is an empty string (""), which indicates that a value must be entered in the input control for it to pass validation. Extra spaces at the beginning and end of the input value are removed before validation is performed to prevent a space being entered in the input control from passing validation.

ValidationSummary

The ValidationSummary control summarizes the error messages from all validation controls on a web page in a single location. The ValidationSummary control provides the following properties.

- *DisplayMode* – specifies whether the summary should be displayed as a list, a bulleted list or a single paragraph.
- *HeaderText* – specifies a custom title in the heading section of the ValidationSummary control.
- *ShowSummary* – specifies whether the ValidationSummary control is displayed or hidden.
- *ShowMessageBox* – specifies whether the summary should be displayed in a browser message box.