


```
value="Login" causesvalidation="false"  
onserverclick="btnLogin_Click"  
validationgroup="LoginGroup" />
```

```
CssClass="alertText"  
DisplayMode="BulletList"  
EnableClientScript="False"  
HeaderText="Error Summary"  
ValidationGroup="RegisterGroup" />
```

Login ID:

```
Columns="30" CssClass="LabelText" />  
Runat="server"  
ControlToValidate="txtNewLoginID"  
CssClass="alertText"  
Display="Dynamic"  
EnableClientScript="False"  
ErrorMessage="Login ID Is Required"  
ValidationGroup="RegisterGroup" >
```

Email Address:

```
Columns="30" CssClass="LabelText" />  
runat="server"  
controltovalidate="txtEmailAddress"  
cssclass="alertText"  
display="Dynamic"  
enableclientscript="False"  
ErrorMessage="Email Address Is Required"  
ValidationGroup="RegisterGroup" >
```

```
Runat="server"  
ControlToValidate="txtEmailAddress"  
CssClass="alertText"
```

```
Display="Dynamic"  
EnableClientScript="False"  
ValidationExpression="w+([-+.]w+)*@w+([-.]w+)*.w+([-.]w+)*"  
ErrorMessage="Invalid Email Address"  
ValidationGroup="RegisterGroup" >
```

Password:

```
TextMode="Password"  
Columns="30" CssClass="LabelText" />  
Runat="server"  
ControlToValidate="txtPassword1"  
CssClass="alertText"  
Display="Dynamic"  
EnableClientScript="False"  
ErrorMessage="Password Is Required"  
ValidationGroup="RegisterGroup" >
```

Re-enter Password:

```
TextMode="Password"  
Columns="30" CssClass="LabelText" />  
Runat="server"  
ControlToValidate="txtPassword2"  
CssClass="alertText"  
Display="Dynamic"  
EnableClientScript="False"  
ErrorMessage="Re-Entered Password Is Required"  
ValidationGroup="RegisterGroup" >
```

```
ControlToValidate="txtPassword2"  
ControlToCompare="txtPassword1"  
CssClass="alertText"  
Display="Dynamic"  
EnableClientScript="False"  
ErrorMessage="Both Passwords Must Match"  
ValidationGroup="RegisterGroup" >
```

```
value="Register" causesvalidation="false"  
onserverclick="btnRegister_ServerClick"  
validationgroup="RegisterGroup" />
```

```
using System;  
using System.Configuration;  
using System.Data;  
using System.Data.OleDb;  
using System.Web.UI;  
using System.Web.UI.WebControls;  
namespace ASPNetCookbook.CSExamples  
{  
    ///   
    /// This class provides the code behind for  
    /// CH03ProgrammaticValidationCS.aspx  
    ///   
    public partial class CH03ProgrammaticValidationCS : System.Web.UI.Page  
    {  
        ///*****  
        ///   
        /// This routine provides the event handler for the authentication server  
        /// validate event. It is responsible checking the login ID and password  
        /// in the database to authenticate the user.  
        ///   
        /// Set to the sender of the event  
        /// Set to the event arguments  
        protected void cvAuthentication_ServerValidate(Object source,  
        System.Web.UI.WebControls.ServerValidateEventArgs args)  
        {  
            OleDbConnection dbConn = null;  
            OleDbCommand dCmd = null;  
            String strConnection = null;  
            String strSQL = null;  
            try  
            {  
                // initially assume credentials are invalid  
                args.IsValid = false;  
                // get the connection string from web.config and open a connection  
                // to the database  
                strConnection = ConfigurationManager.  
                ConnectionStrings["dbConnectionString"].ConnectionString;  
                dbConn = new OleDbConnection(strConnection);  
                dbConn.Open( );  
                // build the query string and check to see if a user with the
```

```
// entered credentials exists in the database
strSQL = "SELECT AppUserID FROM AppUser " +
"WHERE LoginID=? AND " +
"Password=?";
dCmd = new OleDbCommand(strSQL, dbConn);
dCmd.Parameters.Add(new OleDbParameter("LoginID",
txtLoginID.Text));
dCmd.Parameters.Add(new OleDbParameter("Password",
txtPassword.Text));
// check to see if the user was found
if (dCmd.ExecuteScalar( ) != null)
{
args.IsValid = true;
}
} // try
finally
{
// cleanup
if (dbConn != null)
{
dbConn.Close( );
}
} // finally
} // cvAuthentication_ServerValidate
//*****
//
// This routine provides the event handler for the login button click
// event. It is responsible for providing access to the site for the
// user if authenticated.
//
// Set to the sender of the event
// Set to the event arguments
protected void btnLogin_Click(Object sender,
System.EventArgs e)
{
// check to see if all entered login data is valid
if (groupsValid("LoginGroup"))
{
// user has been authenticated so proceed with allowing access
// to the site
}
} //btnLogin_Click
//*****
//
// This routine provides the event handler for the register button click
// event. It is responsible for processing the form data for
// registration.
//
// Set to the sender of the event
```

```
/// Set to the event arguments
protected void btnRegister_ServerClick(Object sender,
System.EventArgs e)
{
// check to see if all entered registration data is valid
if (groupsIsValid("RegisterGroup"))
{
// all entered data is valid so proceed with registration
}
} //btnRegister_ServerClick
///*****
///
/// This routine iterates through validators for the passed group
/// performing the validation.
///
/// Set to the name of the validator
/// group to perform validation on.
///
/// True if all validators in the group are valid.
/// Else, False.
///
private Boolean groupsIsValid(String validatorGroup)
{
ValidatorCollection validators = null;
Boolean isValid = true;
// get the validators in the Register group
validators = Page.GetValidators(validatorGroup);
// iterate through the validators calling the Validate methods
// and checking to see if the validation was successful
foreach (IValidator validator in validators)
{
validator.Validate( );
if (!validator.IsValid)
{
isValid = false;
}
} // foreach validator
return (isValid);
} // groupsIsValid
} // CH03ProgrammaticValidationCS
}
```