Agenda

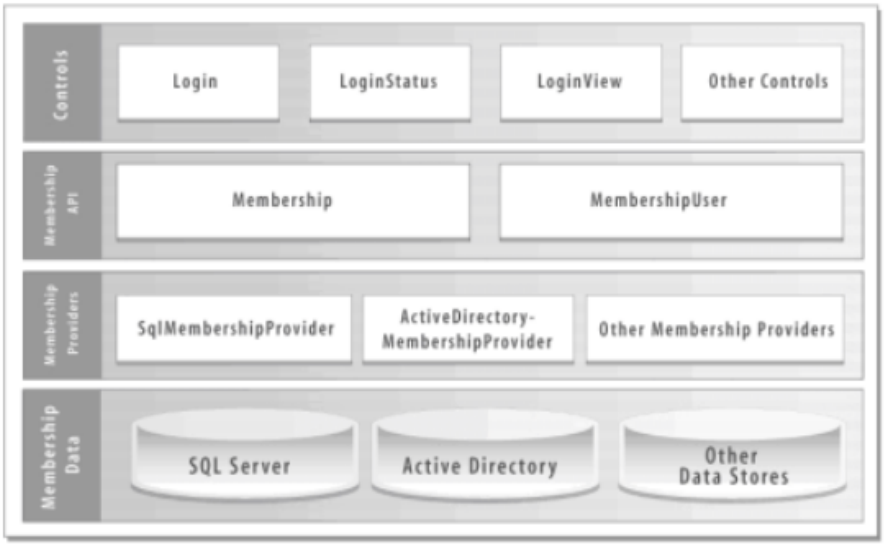
- ASP.NET Providers Introduction
- Membership Providers
- Role Providers
- Writing Custom Providers
- Profile Providers
- Web Parts Personalization Providers

Introduction to Providers

A *provider* is a software module that provides a uniform interface between a service and a data source.

The Provider Model:

Following figure depicts the provider model as it applies to the ASP.NET membership service. In the top layer are the login controls: *Login*, *LoginView*, and others that provide UIs for logging in users, recovering lost passwords, and more. Underneath the login controls sits the membership service, which provides the public API used by the controls and that can be used by application code as well. The membership service stores login credentials and other information in a membership data source, but rather than access the data source directly, it interacts with it through a membership provider. Thus, the login controls and the membership service itself can be adapted to different types of data sources (for example, Oracle databases) simply by adding new providers.



Working with Login Controls

Folder structure	
/MyWebApp/web.config -- <authentication mode="Forms"/> /MyWebApp/Login.aspx Login Control CreateUserUrl, CreateUserText RecoverPasswordUrl, RecoverPasswordText /MyWebApp/Register.aspx CreateUserWizard Control CreatedUser Event Response.Redirect("Default.aspx")	/MyWebApp/Secured/Web.Config <authorization> <deny users="?"/> </authorization> /MyWebApp/Secured/Default.aspx LoginStatus Control Link to ChangePassword.aspx /MyWebApp/Default.aspx LoginStatus Control LoginView Control --Anonymous Template

/MyWebApp/RecoverPassword.aspx PasswordRecovery Control /MyWebApp/Secured/ChangePassword.aspx ChangePassword Control	--LoggedIn Template LoginName Control /Secured/Default.aspx

To Disable UnObstrusive Validation:

```
<appSettings>
        <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
</appSettings>
```

Running the application first time creates the security database in the folder "App_Data". The filename is ASPNETDB.MDF (SQLServer Express Database)

All login controls by default use the MemberShipProvider **"AspNetSqlMembershipProvider"** as mentioned in "machine.config". This membership provider refers to the connectionStringName=**"LocalSqlServer"** which furthers has the information required to connect to the database in the folder "App_Data".

To Replace ASPNETDB.MDF with our own custom SQL Server Database:

Step 1: Steps for configuring an Custom SQL Server database for ASP.NET Application Services

Run the utility program **aspnet_regsql.exe** (from framework folder - C:\WINDOWS\Microsoft.NET\Framework\v4.0.50727\aspnet_regsql.exe)

Step 2: Copy <connectionStrings> and <membership> from Machine.config and paste in root directory Web.Config

Changes in Web.Config

```
<configuration>
 <connectionStrings>
  <add name="csTestingDB"  connectionString="data
source=.\SQLEXPRESS;uid=sa;pwd=dss;database=DemoWebApp"
providerName="System.Data.SqlClient"/>
 </connectionStrings>
 <system.web>
  <membership defaultProvider="SDSqlMembershipProvider">
   <providers>
    <add connectionStringName="csTestingDB" enablePasswordRetrieval="false"
name="SDSqlMembershipProvider" applicationName="/DemoWebApp" enablePasswordReset="true"
requiresQuestionAndAnswer="true" requiresUniqueEmail="false" passwordFormat="Hashed"
maxInvalidPasswordAttempts="5" minRequiredPasswordLength="7" minRequiredNonalphanumericCharacters="1"
passwordAttemptWindow="10" passwordStrengthRegularExpression=""
      type="System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.0.0,
```

```
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    </providers>
  </membership>
```

Run the application and verity that the new database is being used.

Step 3: Copy <roleManager> from "Machine.Config" and paste in root directory "web.config":

Changes in Web.Config

```
  <roleManager enabled="true" defaultProvider="SDSqlRoleProvider">
    <providers>
      <add connectionStringName="csTestingDB" applicationName="/DemoWebApp"
name="SDSqlRoleProvider" type="System.Web.Security.SqlRoleProvider, System.Web, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    </providers>
  </roleManager>
```

Step 4: Goto **ASP.NET Configuration** (Icon in Solution Explorer) → Security → Create or Manage Roles → Create roles as required and also attach users to these roles.

Step 5: In /Secured/Web.config

<deny roles="r1"/>

Step 6: Run the web application and see that the user with role "r1" is not allowed to view the content of secured folder.

Step 7: Using the class "MemberShip" and "Roles", we can manage the users and their roles in our database. These classes are pre-programmed to use the default MemberShip and Role Providers.

Programatically working on Providers

```
protected void btnCreateRole_Click(object sender, System.EventArgs e)
{
   Roles.CreateRole(txtRole.Text);
}
protected void btnGetAllRoles_Click(object sender, System.EventArgs e)
{
   foreach (string role in Roles.GetAllRoles())
   {
      Response.Write(role + " ");
   }
}
protected void btnAssignRoleToUser_Click(object sender, System.EventArgs e)
{
```

```
    Roles.AddUserToRole(txtUserName.Text, txtRoleName.Text);
}
protected void btnGetAllUsers_Click(object sender, System.EventArgs e)
{
    foreach (MembershipUser user in Membership.GetAllUsers())
    {
        Response.Write(user.UserName + " ");
    }
}
protected void btnGetRolesOfUser_Click(object sender, System.EventArgs e)
{
    foreach (string role in Roles.GetRolesForUser(txtUserName.Text))
    {
        Response.Write(role + " ");
    }
}
```

Custom Membership and Role Provider

1. Create a ClassLibrary Project in the exiting solution.
2. Add Reference to System.Web.Dll & System.Configuration.Dll
3. For Custom MemberShip Provider write a class inherited from **System.Web.Security.MemberShipProvider**
4. For Custom Role Provider write a class inherited from **System.Web.Security.RoleProvider**

To Use the Custom Providers in the WebApplication:

1. Add the Reference to the Custom Providers classLibrary
2. In Web.Config, in <membership> and <roleManager> tags set the Type attribute of the <add> tag to our class names.

Note: We can use Login Controls only for their UI. i.e. by handling their events we can make them functional even without using providers. For example we can handle “Authenticate” event of Login control to authenticate the provided username and password.

Working with Profile Provider

The Profile data is persistent and is stored in the database. The values of these properties are different for every identity and are loaded based on the logged in user.

SetProfile.aspx

```
RealName <asp:TextBox ID="txtRealName" runat="server"/>
```

```
Color <asp:TextBox ID="txtColor" runat="server"/>
Line1 <asp:TextBox ID="txtLine1" runat="server"/>
Street   <asp:TextBox ID="txtStreet" runat="server"/>
City <asp:TextBox ID="txtCity" runat="server"/>
<asp:Button ID="btnSetProfile" runat="server" Text="Set Profile" />
```

SetProfile.aspx.cs

```
protected void btnGetProfile_Click(object sender, System.EventArgs e)
{
   lblRealName.Text = Profile.RealName;
   lblColor.Text = Profile.BackColor.ToString();
   lblLine1.Text = Profile.Address.Line1;
   lblStreet.Text = Profile.Address.Street;
   lblCity.Text = Profile.Address.City;
}
```

GetProfile.aspx

```
<asp:Button ID="btnGetProfile" runat="server" Text="Profile" />
Real Name: <asp:Label ID="lblRealName" runat="server"/>
Color: <asp:Label ID="lblColor" runat="server" />
Line1: <asp:Label ID="lblLine1" runat="server" />
Street: <asp:Label ID="lblStreet" runat="server"/>
City: <asp:Label ID="lblCity" runat="server"/>
```

GetProfile.aspx.cs

```
protected void btnSetProfile_Click(object sender, System.EventArgs e)
{
   Profile.RealName = txtRealName.Text;
   Profile.BackColor = System.Drawing.Color.FromName(txtColor.Text);
   Profile.Address.Line1 = txtLine1.Text;
   Profile.Address.Street = txtStreet.Text;
   Profile.Address.City = txtCity.Text;
}
```

In Web.config

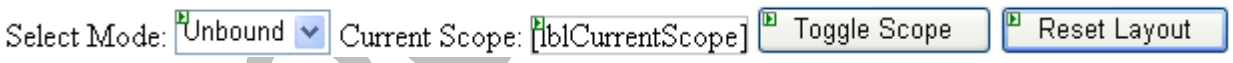
```
<anonymousIdentification enabled="true"/>

<profile enabled="true" defaultProvider="SDProfileProvider">
```

```
 <providers>
  <add name="SDProfileProvider" connectionStringName="csTestingDB" applicationName="/ DemoWebApp"
type="System.Web.Profile.SqlProfileProvider, System.Web, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"/>
 </providers>
 <properties>
  <add name="RealName"  allowAnonymous="true"/>
  <add name="BackColor" type="System.Drawing.Color" allowAnonymous="true" serializeAs="Binary"/>
  <group name="Address">
   <add name="Line1" type="System.String" allowAnonymous="true"/>
   <add name="Street" type="System.String" allowAnonymous="true"/>
   <add name="City" type="System.String" allowAnonymous="true"/>
  </group>
 </properties>
</profile>
```

Personalization using WebParts

1. In default.aspx add table with 1 row and 3 columns. Set width of 1st and 3rd column to 250px.
2. In the 1st and 3rd column add the control WebPartZone (Width="100%")
3. In the zones add other controls as needed. Each control added will behave like a webpart.
4. For every control add an attribute "Title". This will become title of the webpart.
5. Add the following on top of default.aspx



Example: In Default.aspx file

```
<asp:WebPartManager ID="wpManager" runat="server"/><br/>
Select Mode:<asp:DropDownList ID="ddlModes" runat="server" AutoPostBack="True"/>
Current Scope: <asp:Label ID="lblCurrentScope" runat="server"></asp:Label>
<asp:Button ID="btnToggleScope" runat="server" Text="Toggle Scope"  OnClick="btnToggleScope_Click" />
<asp:Button ID="btnResetLayout" runat="server" Text="Reset Layout" OnClick="btnResetLayout_Click" />
```

6. Write the following code in the Default.aspx.cs file

Default.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
```

```
        foreach (WebPartDisplayMode mode in wpManager.SupportedDisplayModes)
        {
            if (mode.IsEnabled(wpManager))
            {
                ListItem li = new ListItem(mode.Name);
                ddlModes.Items.Add(li);
            }
        }
    }
    protected void ddlModes_SelectedIndexChanged(object sender, EventArgs e)
    {
        wpManager.DisplayMode = wpManager.SupportedDisplayModes[ddlModes.SelectedValue];
    }
    protected void btnToggleScope_Click(object sender, EventArgs e)
    {
        if (wpManager.Personalization.CanEnterSharedScope)
            wpManager.Personalization.ToggleScope();
    }
    protected void btnResetLayout_Click(object sender, EventArgs e)
    {
        wpManager.Personalization.ResetPersonalizationState();
    }
    protected override void OnPreRender(EventArgs e)
    {
        lblCurrentScope.Text = wpManager.Personalization.Scope.ToString();
        base.OnPreRender(e);
    }
```

7. In root directory web.config add the following

Changes in Web.config

```
<authentication mode="Forms" />
<authorization>
 <!-- Root Directory content is Secured -->
 <deny users="?"/>
</authorization>
<webParts>
 <personalization>
  <authorization>
   <allow users="admin" verbs="enterSharedScope"/>
```

```
    <deny users="*" verbs="enterSharedScope"/>
    <allow users="*" verbs="modifyState"/>
   </authorization>
  </personalization>
</webParts>
```

8. To the project add “Login.aspx” and add to it a Login Control
9. Handle “Authenticate” event of login control

Handling Authenticate event

```
protected void Login1_Authenticate(object sender, AuthenticateEventArgs e)
{
   if (Login1.UserName == Login1.Password)
      e.Authenticated = true;
   else
      e.Authenticated = false;
}
```

10. Run the webform → Login using u1/u1 as u/p → Change to Design Mode → Drag and Drop webparts → Close the browser
11. Run the webform again → Login again → See that the Layout as set earlier is retained.

    Types of Scopes:

    Shared Scope: Default layout for all users

    User Scope: Layout for a particular user

    In Shared Scope the layout created becomes the default layout for all users unless they change the layout for themselves in User Scope.
12. Run the webform again → Login using admin/admin as u/p → Toggle scope to Shared → Customize the layout.

Summary:

In this article we have studied various types of providers that can be used for authentication and authorization. Web parts were also covered along with sample programs.