Contents

The process of logging in a user involves a handshake between the client application and the Entity Server. Here we'll discuss what you can do **on the server** to authenticate users of your application.

DevForce supplies built-in login processing on the server if using ASP.NET security. When not using ASP.NET security you must provide a custom implementation of the *IEntityLoginManager* interface, otherwise all users will be logged in as "guest" users.

Implementing IEntityLoginManager

The *IEntityLoginManager* is responsible for verifying a user's identity. Add a class which implements the interface's methods, and ensure that DevForce can find the class by placing the assembly in the same folder as the executable or bin folder. The assembly needs to be deployed only on the server, since login authentication processing is not performed on the client.

The interface prescribes two methods: *Login* and *Logout*.

Login

Your Login method will be passed the *ILoginCredential* that the client used in the *Login* call. Note if you call *Login* without credentials, or allow an implicit login to take place, that the credential will be null. Your code should handle this.

An *EntityManager* is also passed to the method to allow you to easily query your domain model. The *EntityManager* here is a special, server-side *EntityManager* instance which is already "connected" to the EntityServer and does not require a login; it is not the same EntityManager from which you called Login on the client. This *EntityManager* is not an instance of your sub-typed domain-specific *EntityManager* either, but rather of the base *EntityManager* class.

From your *Login* method, you should return a type implementing *IPrincipal*. Common implementations are *GenericPrincipal*, *WindowsPrincipal*, or *UserBase*; but any serializable type is allowed. Neither *GenericPrincipal* nor *WindowsPrincipal* is available in Silverlight applications, but you can use the DevForce *UserBase* class or implement a custom *IPrincipal*. If the credentials supplied are not valid, you should throw a *LoginException* indicating the cause of the failure.

Logout

The *Logout* method allows you to perform any processing needed when the user logs off. You might find this useful to perform session-level auditing or resource cleanup. Even if you have no need for logout processing, you must still implement an empty method.

Sample

Here's a sample class implementing the *IEntityLoginManager* interface. It returns a *UserBase* from the Login method, and requires no special Logout processing.

Note that this sample performs no actual authentication of the user credentials, and should therefore not be used in a real application! It also throws an exception if a credential was not supplied. The sample is intended only to show the simple mechanics of the interface.

```
public class LoginManager : IEntityLoginManager {
 public IPrincipal Login(ILoginCredential credential,
   EntityManager entityManager) {
  if (credential == null) {
   throw new LoginException(LoginExceptionType.NoCredentials,
     "No credentials supplied");
   }
  // Return an IPrincipal. Any serializable type may be returned,
  // here we use the DevForce UserIdentity and UserBase classes.
  var identity = new UserIdentity(credential.UserName, "Custom", true);
   var principal = new UserBase(identity);
  return principal;
 }
 public void Logout(IPrincipal principal,
   EntityManager entityManager) {
  // No special processing needed.
```

```
 }
}
```

```
Public Class LoginManager
 Implements IEntityLoginManager
 Public Function Login(ByVal credential As ILoginCredential, _
  ByVal entityManager_Renamed As EntityManager) As IPrincipal
  If credential Is Nothing Then
    Throw New LoginException(LoginExceptionType.NoCredentials, _
      "No credentials supplied")
  End If
  ' Return an IPrincipal. Any serializable type may be returned,
  ' here we use the DevForce UserIdentity and UserBase classes.
  Dim identity = New UserIdentity(credential.UserName, "Custom", True)
  Dim principal = New UserBase(identity)
  Return principal
 End Function
 Public Sub Logout(ByVal principal As IPrincipal, _
  ByVal entityManager_Renamed As EntityManager)
  ' No special processing needed.
 End Sub
End Class
```

Ensuring that Login will fail if an IEntityLoginManager class is not deployed

To ensure that your custom *IEntityLoginManager* is found and used, you can set a configuration option which will cause DevForce to throw an exception when not found. By default the *EntityServer* will resort to it's own authentication processing when a custom implementation is not found and ASP.NET security is not in use: that processing ignores supplied credentials and logs all users in as guests. This is rarely what your applications wants, except in those rare cases where only anonymous access is provided in the application.

If you've supplied an *IEntityLoginManager* and you want to ensure that it is in fact being used, set the *LoginManagerRequired* flag in the web.config or server .config to true:

```
<objectServer>
  <serverSettings loginManagerRequired="true" />
</objectServer>
```

Allowing guest users

A guest user is an anonymous user, in other words, one which has not supplied credentials. By default in DevForce, a guest user will have a name beginning with "Guest" followed by an incrementing number (for example "Guest - 1"), an authentication type of "Anonymous", and will not be authenticated.

DevForce allows guest users by default, but if your application requires that users authenticate to use its features you should disable this setting. You do so by setting the *AllowAnonymousLogin* flag in the web.config or server .config:

```
<objectServer>
  <serverSettings allowAnonymousLogin="false" />
</objectServer>
```

When guest access is disallowed and you do not have a custom *IEntityLoginManager*, a *LoginException* will be thrown when a guest login is attempted.

Note that this setting has nothing to do with the IIS Anonymous Authentication setting, and is used only within DevForce.