Contents

You can **customize the Login process** by providing your own implementations of many interfaces. DevForce will usually find your custom implementations without any extra work on your part. See the topic on extending DevForce for more information on how to ensure your custom types are discovered.

IEntityLoginManager

Most applications need their own *IEntityLoginManager* to validate the credentials supplied by users. Unless you're using ASP.NET security, DevForce does not provide an *IEntityLoginManager* out of the box and you will need to supply your own. It's easy, and you can find more information here.

If you're using ASP.NET security DevForce does provide a default login manager which is sufficient for many applications. You can sub-class this login manager, the *AspAuthenticatingLoginManager* if you require further customization.

ILoginCredential

Any serializable type implementing the *ILoginCredential* interface can be used. The *LoginCredential* and the *FormsAuthenticationLoginCredential* are the DevForce-supplied implementations. The credential supplied in the *Login* call on the client is the credential received in the *IEntityLoginManager.Login* method. The class defining the *ILoginCredential* must be available on both client and server.

IPrincipal

Any serializable type implementing System.Security.Principal.IPrincipal can be returned from the *IEntityLoginManager.Login* method. The object is returned to the client and is available via the *AuthenticationContext.Principal*. The *IPrincipal* is also available to other server methods. The class defining the IPrincipal must be available on both client and server. The *IIdentity* held by the IPrincipal must also be a serializable type and available on both client and server.

If you sub-type either *UserBase* or *UserIdentity*, the DevForce default implementations of *IPrincipal* and *IIdentity*, your classes must be:

1. Marked with the DataContract attribute. Also be sure to mark any custom properties which should be transmitted between tiers with the DataMember attribute.
2. Included on both the client and server! You won't be able to login if your custom classes aren't defined in assemblies deployed to both client and server tiers.

LoginException

Any serializable type extending *LoginException* can be thrown for login failures. To ensure that your custom exception is received correctly on the client, you must also implement a constructor accepting the message, and a constructor accepting a dictionary of any custom properties. DevForce will automatically serialize any custom properties via a Dictionary<string, object>, and will call the appropriate constructor when building the exception on the client. The class defining the custom exception must be available on both client and server. For example:

```
[DataContract]
public class CustomLoginException : LoginException {
 public CustomLoginException(string message, int severity) :
   base(message) {
    Severity = severity;
  }
 public CustomLoginException(string message,
    Dictionary<string, object> userData) : base(message) {
    Severity = (int)userData["Severity"];
  }
  [DataMember]
 public int Severity {
  get;
  internal set;
  }
}
```

```
<DataContract()> _
Public Class CustomLoginException
 Inherits LoginException
 Public Sub New(ByVal message As String, ByVal severity As Integer)
   MyBase.New(message)
   Me.Severity = severity
 End Sub
 Public Sub New(ByVal message As String, _
   ByVal userData As Dictionary(Of String, Object))
   MyBase.New(message)
    Severity = CInt(Fix(userData("Severity")))
 End Sub
 Private privateSeverity As Integer
  <DataMember()> _
 Public Property Severity() As Integer
   Get
     Return privateSeverity
   End Get
   Friend Set(ByVal value As Integer)
      privateSeverity = value
   End Set
 End Property
End Class
```