Contents

- Login
  - Login credentials
  - LoginOptions
- Logout
- Single sign-on / sign-off
- Authenticate without logging in

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

Authentication begins on the client, when *Login* is called. Until a *Login* successfully completes, an *EntityManager* may not communicate with the *EntityServer* for data or services.

Login

At its simplest, gather user credentials into an *ILoginCredential* and call the **Authenticator** *Login* or *LoginAsync* method.

```
IAuthenticationContext Login(ILoginCredential credential = null, LoginOptions options = null)
Task<IAuthenticationContext> LoginAsync(ILoginCredential credential = null, LoginOptions options = null)
Task<IAuthenticationContext> LoginAsync(ILoginCredential credential, LoginOptions options, CancellationToken cancellationToken)
```

It's worth noting that the credential is passed in clear text. Use a secure channel (such as SSL) or provide your own encryption of the credential data if secure communications are required.

After the login completes successfully you can use the returned *IAuthenticationContext* to set the *DefaultAuthenticationContext* for the application, or set the *AuthenticationContext* for a specific *EntityManager*.

When the ***DefaultAuthenticationContext*** is set, by default all EntityManagers are considered "logged in" and may communicate with an *EntityServer*. The *Principal* property indicates the logged in user identity and roles.

If the *Login* failed, a *LoginException* is thrown.

Login credentials

DevForce provides two implementations of the *ILoginCredential*: the *LoginCredential* and FormsAuthenticationLoginCredential. If necessary, you can sub-type one of these, or create your own implementation too.

What if you don't have a credential? For example, you're using Windows credentials, or allow guest access, or you're using ASP.NET security with Windows authentication or a persistent or ambient authentication ticket. In these cases, calling the Login method without a supplied credential indicates that the EntityServer will determine how to handle the login request. When using ASP.NET security, the built-in implementation will try to use the current user information from the HttpContext.

LoginOptions

The *LoginOptions* can be passed with a login call to indicate information about the *EntityServer* which will perform the authentication. You can specify the data source extension, composition context, and other information, to tell the *Authenticator* which *EntityServer* to use. If you don't specify the *LoginOptions*, by default an *EntityServer* without a data source extension or composition context is used.

Logout

The *Logout* method performs another handshake with the EntityServer, telling it to remove session information for the user. Further requests to the server from a "logged out" EntityManager are disallowed until it again logs in.

```
void Logout(IAuthenticationContext context)
Task LogoutAsync(IAuthenticationContext context)
Task LogoutAsync(IAuthenticationContext context, CancellationToken cancellationToken)
```

Many applications do not explicitly call Logout, and it's not strictly necessary. The user session information on the server will expire after a period of inactivity. If your application is available on public computers it's generally a good idea to logout either by user request or when the application closes.

Note that the user session information stored on the server has a very small footprint. It's simply a bundle containing encrypted credentials and a few additional fields tracking last activity; no other data is stored.

Single sign-on / sign-off

DevForce implements single sign-on/sign-off by default. Every EntityManager by default uses a shared "authentication context", the *Authenticator.DefaultAuthenticationContext* .

You can set the *DefaultAuthenticationContext* using the IAuthenticationContext returned by a call to *Login* or *LoginAsync*.

```
// With a sync login:
var cred = new LoginCredential("max", "headroom", string.Empty);
Authenticator.Instance.DefaultAuthenticationContext = Authenticator.Instance.Login(cred);
// Or async login:
var cred = new LoginCredential("max", "headroom", string.Empty);
var context = await Authenticator.Instance.LoginAsync(cred);
Authenticator.Instance.DefaultAuthenticationContext = context;
```

An EntityManager can opt out of this shared context by setting its *Options.UseDefaultAuthenticationContext* to false.

When an *AuthenticationContext* is logged out, any EntityManager using that context is logged out.

Authenticate without logging in

Now that we've covered the mechanics of client login, it's time to cover other ways in which an *EntityManager* can be authenticated.

There are several ways, and they're based on sharing an existing *AuthenticationContext* . Remember that by default, an *EntityManager* will use the *Authenticator.DefaultAuthenticationContext*.

1. **Create an EntityManager via the copy constructor**

```
var newEntityManager = new EntityManager(anotherEntityManager);
```

```
Dim newEntityManager = New EntityManager(anotherEntityManager)
```

The new *EntityManager* will have the same connection state, and share the user credentials from the other *EntityManager*.

2. **Set the *EntityManager.AuthenticationContext***

You can share an *AuthenticationContext* among *EntityManagers* by setting the *EntityManager.AuthenticationContext* property.

```
var authContext = Authenticator.Instance.Login();
...
var entityManager = new EntityManager();
entityManager.AuthenticationContext = authContext;
```