Contents

- Enable the Authentication Service
- Using the Authentication Service

Most Enterprise Applications require a user to be authenticated and authorized.

Punch provides an authentication service that implements the required functionality for most types of applications. If desired, the authentication service can be sub-classed and extended. The purpose of the authentication service is to handle the client-side login process and to wire up all the necessary parts.

The authentication service works in conjunction with a DevForce LoginManager on the server and once a user is authenticated, access to data and services can be controlled using the DevForce authorization mechanisms.

See also:

- Understanding the Login Process
- Using ASP.NET Security
- Authorizing access to data and services

Enable the Authentication Service

The authentication service consist of two parts. The IAuthenticationService interface and the default AuthenticationService implementation. To enable the service, it simply needs to be made available to the application. This is typically accomplished in the application's bootstrapping logic.

```
public class AppBootstrapper : CocktailMefBootstrapper<ShellViewModel>
{
  protected override void PrepareCompositionContainer(CompositionBatch batch)
   {
     base.PrepareCompositionContainer(batch);
      batch.AddExportedValue<IAuthenticationService>(new AuthenticationService());
   }
}
```

The AuthenticationService can be configured with specific ConnectionOptions in order to control where it connects. If no ConnectionOptions are specified, *ConnectionOptions.Default* will be used.

The following snippet shows how to configure the AuthenticationService to connect to the Fake Backing Store.

```
public class AppBootstrapper : CocktailMefBootstrapper<ShellViewModel>
{
  protected override void PrepareCompositionContainer(CompositionBatch batch)
   {
     base.PrepareCompositionContainer(batch);
      batch.AddExportedValue<IAuthenticationService>(new AuthenticationService()
         .Configure(auth => auth.WithConnectionOptions(ConnectionOptions.Fake.Name)));
   }
}
```

Using the Authentication Service

Once the authentication service is enabled it can be used by simply injecting it where it is needed:

```
[Export]
public class LoginViewModel : Screen
{
  private readonly IAuthenticationService _authenticationService;
   [ImportingConstructor]
  public LoginViewModel(IAuthenticationService authenticationService,
                        [Import(RequiredCreationPolicy = CreationPolicy.NonShared)] IBusyWatcher busy)
   {
      Busy = busy;
     _authenticationService = authenticationService;
   }
  public IBusyWatcher Busy { get; private set; }
  public string Username
   {
     get { return _username; }
     set
```

```
        {
            _username = value;
            NotifyOfPropertyChange(() => Username);
        }
    }
    public string Password
    {
        get { return _password; }
        set
        {
            _password = value;
            NotifyOfPropertyChange(() => Password);
        }
    }
    public string FailureMessage
    {
        get { return _failureMessage; }
        set
        {
            _failureMessage = value;
            NotifyOfPropertyChange(() => FailureMessage);
        }
    }
    public async void Login()
    {
        using (Busy.GetTicket())
        {
            FailureMessage = "";
            var hash = CryptoHelper.GenerateKey(Password);
            var password = Encoding.UTF8.GetString(hash, 0, hash.Length);
            var credential = new LoginCredential(Username, password, null);
            // Clear username and password fields
            Username = null;
            Password = null;
            try
            {
                await _authenticationService.LoginAsync(credential);
                TryClose();
            }
            catch (Exception e)
            {
                FailureMessage = e.Message
            }
        }
    }
}
```