Implementing the challenge handler in Windows 8.1 Universal and Windows 10 UWP applications

Overview

Prerequisite: Make sure to read the **CredentialsValidationSecurityCheck**'s challenge handler implementation (../../credentials-validation/android) tutorial.

The challenge handler will demonstrate a few additional features (APIs) such as the preemptive `Login`, `Logout` and `ObtainAccessToken`.

Login

In this example, `UserLoginSecurityCheck` expects *key:value*s called `username` and `password`. Optionally, it also accepts a boolean `rememberMe` key that will tell the security check to remember this user for a longer period. In the sample application, this is collected using a boolean value from a checkbox in the login form.

`credentials` is a `JSONObject` containing `username`, `password` and `rememberMe`:

```
public override void SubmitChallengeAnswer(object answer)
{
   challengeAnswer = (JObject)answer;
}
```

You may also want to login a user without any challenge being received. For example, showing a login screen as the first screen of the application, or showing a login screen after a logout, or a login failure. We call those scenarios **preemptive logins**.

You cannot call the `challengeAnswer` API if there is no challenge to answer. For those scenarios, the MobileFirst Foundation SDK includes the `Login` API:

```
WorklightResponse response = await Worklight.WorklightClient.CreateInstance().AuthorizationManager.Login(String securityCheckName, JObject credentials);
```

If the credentials are wrong, the security check will send back a **challenge**.

It is the developer's responsibility to know when to use `Login` vs `challengeAnswer` based on the application's needs. One way to achieve this is to define a boolean flag, for example `isChallenged`, and set it to `true` when reaching `HandleChallenge` or set it to `false` in any other cases (failure, success, initializing, etc).

When the user clicks the **Login** button, you can dynamically choose which API to use:

```
public async void login(JSONObject credentials)
{
   if(isChallenged)
   {
      challengeAnswer= credentials;
   }
   else
   {
      WorklightResponse response = await Worklight.WorklightClient.CreateInstance().AuthorizationMana
ger.Login(securityCheckName, credentials);
   }
}
```

Obtaining an access token

Since this security check supports *remember me* functionality, it would be useful to check if the client is currently logged in, during the application startup.

The MobileFirst Foundation SDK provides the `ObtainAccessToken` API to ask the server for a valid token:

```
WorklightAccessToken accessToken = await Worklight.WorklightClient.CreateInstance().AuthorizationMa
nager.ObtainAccessToken(String scope);

if(accessToken.IsValidToken && accessToken.Value != null && accessToken.Value != "")
{
  Debug.WriteLine("Auto login success");
}
else
{
  Debug.WriteLine("Auto login failed");
}
```

If the client is already logged-in or is in the *remembered* state, the API will trigger a success. If the client is not logged in, the security check will send back a challenge.

The `ObtainAccessToken` API takes in a **scope**. The scope can be the name of your **security check**.

> Learn more about **scope** in the Authorization concepts (../../authorization-concepts) tutorial

Retrieving the authenticated user

The challenge handler's `HandleSuccess` method receives a `JObject identity` as a parameter. If the security check sets an `AuthenticatedUser`, this object will contain the user's properties. You can use `HandleSuccess` to save the current user:

```
public override void HandleSuccess(JObject identity)
{
    isChallenged = false;
    try
    {
        //Save the current user
        var localSettings = Windows.Storage.ApplicationData.Current.LocalSettings;
        localSettings.Values["useridentity"] = identity.GetValue("user");

    } catch (Exception e) {
        Debug.WriteLine(e.StackTrace);
    }
}
```

Here, `identity` has a key called `user` which itself contains a `JObject` representing the `AuthenticatedUser`:

```
{
  "user": {
    "id": "john",
    "displayName": "john",
    "authenticatedAt": 1455803338008,
    "authenticatedBy": "UserLogin"
  }
}
```

Logout

The MobileFirst Foundation SDK also provides a `Logout` API to logout from a specific security check:

```
WorklightResponse response = await Worklight.WorklightClient.CreateInstance().AuthorizationManager.Logout(securityCheckName);
```

Sample applications

There are two samples associated with this tutorial:

- **PreemptiveLoginWin**: An application that always starts with a login screen, using the preemptive `Login` API.
- **RememberMeWin**: An application with a *Remember Me* checkbox. The user can bypass the login screen the next time the application is opened.

Both samples use the same `UserLoginSecurityCheck` from the **SecurityCheckAdapters** adapter Maven project.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/SecurityCheckAdapters/tree/release80) the SecurityCheckAdapters Maven project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/RememberMeWin8/tree/release80) the Remember Me Win8 project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/RememberMeWin10/tree/release80) the Remember Me Win10 project.
Click to download (https://github.com/MobileFirst-Platform-Developer-

Center/PreemptiveLoginWin8/tree/release80) the PreemptiveLogin Win8 project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PreemptiveLoginWin10/tree/release80) the PreemptiveLoginWin10 project.

Sample usage

- Use either Maven, MobileFirst CLI or your IDE of choice to build and deploy the available **ResourceAdapter** and **UserLogin** adapters (../../../adapters/creating-adapters/).
- From a **Command-line** window, navigate to the project's root folder and run the command: `mfpdev app register`.
- Map the `accessRestricted` scope to the `UserLogin` security check:
  - In the MobileFirst Operations Console, under **Applications → [your-application] → Security → Scope-Elements Mapping**, add a scope mapping from `accessRestricted` to `UserLogin`.
  - Alternatively, from the **Command-line**, navigate to the project's root folder and run the command: `mfpdev app push`.

> Learn more about the mfpdev app push/push commands in the Using MobileFirst CLI to manage MobilefFirst artifacts (../../../using-the-mfpf-sdk/using-mobilefirst-cli-to-manage-mobilefirst-artifacts).