

Custom Authentication in native Windows Phone 8 applications

Relevant to:



Native Windows Phone 8

Overview

This tutorial illustrates the native Windows Phone 8 client-side authentication components for custom authentication. Make sure you read [Custom Authentication](#) first.

Creating the client-side authentication components

Create a native Windows Phone 8 application and add the MobileFirst native APIs following the documentation.

CustomChallengeHandler

Create a `CustomChallengeHandler` class as a subclass of `ChallengeHandler`.

`CustomChallengeHandler` should implement

- `isCustomResponse`
- `handleChallenge`

`isCustomResponse` checks every custom response received from MobileFirst Server to see if this is the challenge we are expecting.

```
public override bool isCustomResponse(WLResponse response)
{
    if(response == null ||
        response.getResponseJSON() == null)
    {
        return false;
    }

    if(response.ToString().IndexOf("authStatus") > -1)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

`handleChallenge` method, is called after the `isCustomResponse` method returned true.

Within this method we present our login form. Different approaches may be adopted to present the login form.

```
public override void handleChallenge(JObject response)
{
    Deployment.Current.Dispatcher.BeginInvoke(() =>
    {
        MainPage._this.NavigationService.Navigate(new Uri("/LoginPage.xaml",
            UriKind.Relative));
    });
}
```

From the login form , credentials are passed to the `CustomChallengeHandler` class. The `submitLoginForm()` method is used to send our input data to the authenticator.

```
public void submitLogin(string username, string password)
{
    Dictionary<String, String> parms = new Dictionary<String, String>();
    parms.Add("username", username);
    parms.Add("password", password);
    submitLoginForm("/my_custom_auth_request_url", parms, null, 10000, "post");
}
```

MainPage

Within the `MainPage` class connect to MobileFirst server, register your `challengeHandler` and invoke the protected adapter procedure.

The procedure invocation will trigger MobileFirst server to send a challenge that will trigger our `challengeHandler`.

```
WLClient client;
client = WLClient.GetInstance();
challengeHandler = new WindowsChallengeHandler();
client.registerChallengeHandler((BaseChallengeHandler<JObject>)challengeHandler);
client.connect(new MyConnectResponseListener(this));
```

Since the native API not protected by a defined security test, there is no login form presented during server connection.

Invoke the protected adapter procedure and the login form is presented by the `challengeHandler`.

```
WLProcedureInvocationData invokeData = new
    WLProcedureInvocationData("AuthAdapter", "getSecretData");
WLRequestOptions options = new WLRequestOptions();
client.invokeProcedure(invokeData, new MyResponseListener(this), options);
```

Worklight Protocol

If your custom authenticator uses `WorklightProtocolAuthenticator`, some simplifications can be made:

- Subclass your challenge handler using `WLChallengeHandler` instead of `ChallengeHandler`. Note the `WL`.
- You no longer need to implement `isCustomResponse` as the challenge handler will automatically check that the realm name matches.
- `handleChallenge` will receive the challenge as a parameter, not the entire response object.
- Instead of `submitLoginForm`, use `submitChallengeAnswer` to send your challenge response as a JSON.
- There is no need to call `submitSuccess` or `submitFailure` as the framework will do it for you.

For an example that uses `WorklightProtocolAuthenticator`, see the [Remember Me](#) tutorial or [this video blog post](#).

Sample application

[Click to download](#) the MobileFirst project.

[Click to download](#) the Native project.

- The `CustomAuth` project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The `CustomAuthWP8` project contains a native WP8 application that uses a MobileFirst native API library.

- Make sure to update the `worklight.plist` file in the native project with the relevant server settings.

