Custom Authentication in hybrid applications

Overview

This is a continuation of Custom Authentication (../).

Creating client-side authentication components

The application consists of two main *div* elements:
The *AppDiv* element is used to display the application content.
The *AuthDiv* element is used for authentication forms.
When authentication is required, the application hides *AppDiv* and shows *AuthDiv*. When authentication is complete, it does the opposite.

AppDiv

```
1 <div id="AppDiv">
2   <input type="button" id="getSecretDataButton" value="Call protected adapter proc" onclick="getSecretDat
3   <input type="button" class="appButton" value="Logout" onclick="WL.Client.logout('CustomAuthenticatorR
4   <div id="ResponseDiv"></div>
5 </div>
```

Buttons are used to call the *getSecretData* procedure and to log out.

AuthDiv

```
1  <div id="AuthDiv" style="display: none"
2    <p id="AuthInfo"></p>
3    <div id="loginForm">
4      <input type="text" id="AuthUsername" placeholder="Enter username" />
5      <br/>
6      <br/>
7      <input type="password" id="AuthPassword" placeholder="Enter password" />
8      <br/>
9      <input type="button" id="AuthSubmitButton" class="formButton" value="Login" />
10     <input type="button" id="AuthCancelButton" class="formButton" value="Cancel" />
11   </div>
12 </div>
```

AuthDiv is styled with *display:none* because it must not be displayed before the server requests the authentication.

Challenge Handler

Use *WL.Client.createChallengeHandler* to create a challenge handler object. Supply a realm name as a parameter.

```
1  var customAuthenticatorRealmChallengeHandler = WL.Client.createChallengeHandler("CustomAuthenticat
```

The *isCustomResponse* function of the challenge handler is called each time a response is received from the server.
It is used to detect whether the response contains data that is related to this challenge handler. It must return **true** or **false**.

```
1  customAuthenticatorRealmChallengeHandler.isCustomResponse = function(response) {}
```

If *isCustomResponse* returns true, the framework calls the *handleChallenge* function. This function is used to perform required actions, such as hide application screen and show login screen.

```
1  customAuthenticatorRealmChallengeHandler.handleChallenge = function(response){}
```

In addition to the methods that the developer must implement, the challenge handler contains functionality that the developer might want to use:

- *submitLoginForm* to send collected credentials to a specific URL. The developer can also specify request parameters, headers, and callback.
- *submitSuccess* to notify the framework that the authentication finished successfully. The framework then automatically issues the original request that triggered the authentication.
- *submitFailure* to notify the framework that the authentication completed with a failure. The framework then disposes of the original request that triggered the authentication

Note: Attach each of these functions to its object. For example:*myChallengeHandler.submitSucces()*

isCustomResponse

If the challenge JSON block contains the *authStatus* property, return *true*, otherwise return *false*.

```
1  customAuthenticatorRealmChallengeHandler.isCustomResponse = function(response) {
2      if (!response || !response.responseJSON) {
3          return false;
4      }
5      if (response.responseJSON.authStatus)
6          return true;
7      else
8          return false;
9  };
```

handleChallenge

If the *authStatus* property equals “required”, show the login form, clean up the password input field, and display the error message if applicable.
if *authStatus* equals “complete”, hide the login screen, return to the application, and notify the framework that authentication completed successfully.

```
customAuthenticatorRealmChallengeHandler.handleChallenge = function(response){
    var authStatus = response.responseJSON.authStatus;
    if (authStatus == "required"){
        $('#AppDiv').hide();
        $('#AuthDiv').show();
        $("#AuthInfo").empty();
        $('#AuthPassword').val('');
        if (response.responseJSON.errorMessage){
            $("#AuthInfo").html(response.responseJSON.errorMessage);
        }
    } else if (authStatus == "complete"){
        $('#AppDiv').show();
        $('#AuthDiv').hide();
        customAuthenticatorRealmChallengeHandler.submitSuccess();
    }
};
```

Clicking the **login** button triggers the function that collects the user name and password from HTML input fields and submits them to server. You can set request headers here and specify callback functions.

```
$('#AuthSubmitButton').bind('click', function () {
    var reqURL = '/my_custom_auth_request_url';
    var options = {};
    options.parameters = {
        username : $('#AuthUsername').val(),
        password : $('#AuthPassword').val()
    };
    options.headers = {};
    customAuthenticatorRealmChallengeHandler.submitLoginForm(reqURL, options, customAuthenticatorF
});
```

Clicking the **cancel** button hides *AuthDiv*, shows *AppDiv* and notifies the framework that authentication failed.

```
$('#AuthCancelButton').bind('click', function () {
    $('#AppDiv').show();
    $('#AuthDiv').hide();
    customAuthenticatorRealmChallengeHandler.submitFailure();
});
```

The **submitLoginFormCallback** function checks the response for the containing server challenge once again. If the challenge is found, the *handleChallenge* function is called again.

```
customAuthenticatorRealmChallengeHandler.submitLoginFormCallback = function(response) {
    var isLoginFormResponse = customAuthenticatorRealmChallengeHandler.isCustomResponse(response
    if (isLoginFormResponse){
        customAuthenticatorRealmChallengeHandler.handleChallenge(response);
    }
};
```

Worklight Protocol

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

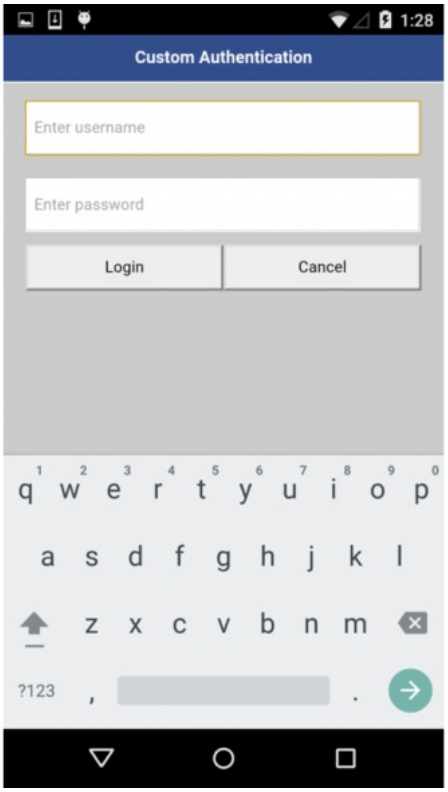
- Create the challenge handler using `WL.Client.create**WL**ChallengeHandler` instead of `WL.Client.createChallengeHandler`. Note the `WL` in the middle.
- You no longer need to implement `isCustomResponse` as the challenge handler will automatically check that the realm name matches.
- `handleChallenge` 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 (../../../advanced-topics/remember-me/) tutorial or this video blog post (file:////home/travis/build/MFPSamples/DevCenter/_site/blog/2015/05/29/ibm-mobilefirst-platform-foundation-custom-authenticators-and-login-modules/).

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomAuth) the MobileFirst project.


1:28
Custom Authentication
Call protected adapter proc
Logout


1:28
Custom Authentication
Enter username
Enter password
Login
Cancel
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123


1:29
Custom Authentication
Call protected adapter proc
Logout
{"isSuccessful":true,"secretData":"123456"}