Resource request from Native Windows 8.1 Universal and Windows 10 UWP applications

Overview

MobileFirst applications can access resources using the `WorklightResourceRequest` REST API.
The REST API works with all adapters and external resources.

Prerequisites:

- Ensure you have added the MobileFirst Platform SDK to your Native Windows 8.1 Universal (../../adding-the-mfpf-sdk/windows-8) or Windows 10 UWP (../../adding-the-mfpf-sdk/windows-10) project.
- Learn how to create adapters (../../adapters/adapters-overview/).

WLResourceRequest

The `WorklightResourceRequest` class handles resource requests to adapters or external resources.

Create a `WorklightResourceRequest` object and specify the path to the resource and the HTTP method. Available methods are: `GET`, `POST`, `PUT` and `DELETE`.

```
URI adapterPath = new URI("/adapters/RSSReader/getFeed",UriKind.Relative);
WorklightResourceRequest request = WorklightClient.ResourceRequest(adapterPath,"GET");
```

- For **JavaScript adapters**, use `/adapters/{AdapterName}/{procedureName}`
- For **Java adapters**, use `/adapters/{AdapterName}/{path}`. The `path` depends on how you defined your `@Path` annotations in your Java code. This would also include any `@PathParam` you used.
- To access resources outside of the project, use the full URL as per the requirements of the external server.
- **timeout**: Optional, request timeout in milliseconds
- **scope**: Optional, if you know which scope is protecting the resource - specifying this scope could make the request more efficient.

Sending the request

Request the resource by using the `.send()` method.

```
WorklightResponse response = await request.send();
```

Use the `WorklightResponse response` object to get the data that is retrieved from the adapter.

The `response` object contains the response data and you can use its methods and properties to retrieve the required information. Commonly used properties are `ResponseText`, `ResponseJSON` (if the response is in JSON) , `Success` (if the invoke was successful or failure) and `HTTPStatus` (the HTTP status of the response).

Parameters

Before sending your request, you may want to add parameters as needed.

Path parameters

As explained above, **path** parameters (`/path/value1/value2`) are set during the creation of the `WorklightResourceRequest` object:

```
URI adapterPath = new URI("/adapters/JavaAdapter/users/value1/value2",UriKind.Relative);
WorklightResourceRequest request = WorklightClient.createInstance(adapterPath,"GET");
```

Query parameters

To send **query** parameters (`/path?param1=value1...`) use the `SetQueryParameter` method for each parameter:

```
request.SetQueryParameter("param1","value1");
request.SetQueryParameter("param2","value2");
```

Form parameters

To send form parameters in the body, use `.Send(Dictionary<string, string> formParameters)` instead of `.Send()` :

```
Dictionary<string,string> formParams = new Dictionary<string,string>();
formParams.Add("height", height.getText().toString());
request.Send(formParams);
```

Header parameters

To send a parameter as an HTTP header use `.AddHeader()` API:

```
request.AddHeader(System.Net.WebHeaderCollection header);
```

Other custom body parameters

- `.Send(requestBody)` allows you to set an arbitrary String in the body.
- `.Send(JObject json)` allows you to set an arbitrary dictionary in the body.
- `.Send(byte[] data)` allows you to set an arbitrary byte array in the body.

Javascript Adapters

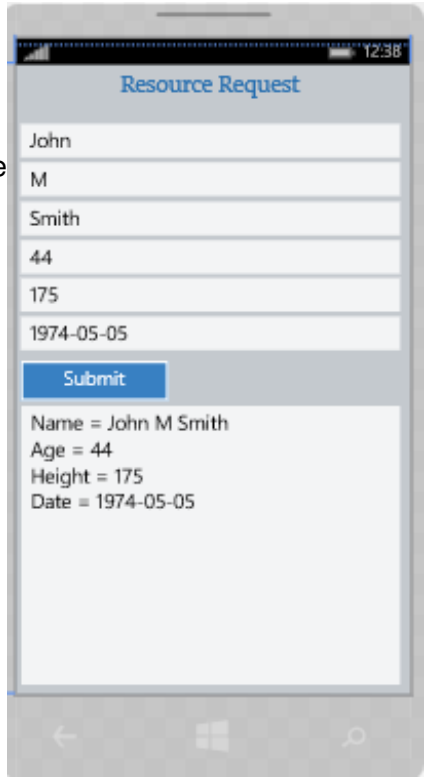
JavaScript adapters use ordered nameless parameters. To pass parameters to a Javascript adapter, set an array of parameters with the name `params` :

```
request.SetQueryParameter("params","['param1', 'param2']");
```

For more information

> For more information about WLResourceRequest, refer to the user documentation.

Sample application



The ResourceRequestWin8 and ResourceRequestWin10 projects contain a native Windows 8 Universal/Windows 10 UWP application that makes a resource request using a Java adapter.
The adapter Maven project contains the Java adapter to be used during the resource request call.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/ResourceRequestWin8/tree/release80) the Native project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/ResourceRequestWin10/tree/release80) the Native project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/Adapters/tree/release80) the adapter Maven project.

Sample usage

1. From the command line, navigate to the Visual Studio project.
2. Ensure the sample is registered in the MobileFirst Server by running the command: `mfpdev app register`.
3. The sample uses the `JavaAdapter` contained in the Adapters Maven project. Use either Maven or MobileFirst Developer CLI to build and deploy the adapter (../../adapters/creating-adapters/).
4. To test or debug an adapter, see the testing and debugging adapters (../../adapters/testing-and-debugging-adapters) tutorial.
5. import the project to Visual Studio, and run the sample by clicking the **Run* button.