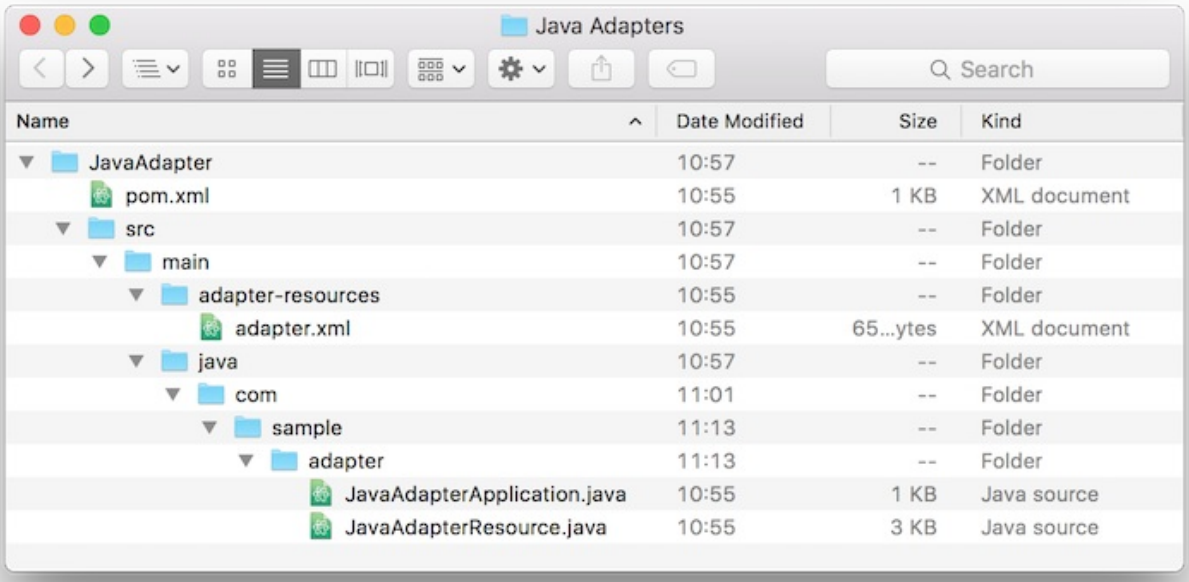
Java Adapters

Overview

Java adapters are based on the JAX-RS 2.0 specification. In other words, a Java adapter is a JAX-RS 2.0 service that can easily be deployed to a MobileFirst Server instance and has access to MobileFirst Server APIs.

Prerequisite: Make sure to read the Creating Java and JavaScript Adapters (../creating-adapters) tutorial first.

File structure



The adapter-resources folder

The `adapter-resources` folder contains an XML configuration file. In this configuration file, configure the class name of the JAX-RS 2.0 application for this adapter. In our case: `com.sample.adapter.JavaAdapterApplication`.

```
<?xml version="1.0" encoding="UTF-8"?>
<mfp:adapter name="JavaAdapter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>JavaAdapter</displayName>
    <description>JavaAdapter</description>

    <JAXRSApplicationClass>sample.JavaAdapterApplication</JAXRSApplicationClass>
</mfp:adapter>
```

The java folder

In this folder, put the Java sources of the JAX-RS 2.0 service. JAX-RS 2.0 services are composed of an application class (which extends `com.worklight.wink.extensions.MFPJAXRSApplication`) and resources classes. The JAX-RS 2.0 application and resources classes define the Java methods and their mapping to URLs. `com.sample.JavaAdapterApplication` is the JAX-RS 2.0 application class and `com.sample.JavaAdapterResource` is a JAX-RS 2.0 resource included in the application.

JAX-RS 2.0 application class

The JAX-RS 2.0 application class tells the JAX-RS 2.0 framework which resources are included in the application.

```
package com.sample.adapter;

import java.util.logging.Logger;
import com.worklight.wink.extensions.MFPJAXRSApplication;

public class JavaAdapterApplication extends MFPJAXRSApplication{

    static Logger logger = Logger.getLogger(JavaAdapterApplication.class.getName());

    @Override
    protected void init() throws Exception {
        logger.info("Adapter initialized!");
    }

    @Override
    protected String getPackageToScan() {
        //The package of this class will be scanned (recursively) to find JAX-RS 2.0 resources.
        return getClass().getPackage().getName();
    }
}
```

The `MFPJAXRSApplication` class scans the package for JAX-RS 2.0 resources and automatically creates a list. Additionally, its `init` method is called by MobileFirst Server as soon as the adapter is deployed (before it starts serving) and when the MobileFirst runtime starts up.

Implementing a JAX-RS 2.0 resource

JAX-RS 2.0 resource is a POJO (Plain Old Java Object) which is mapped to a root URL and has Java methods for serving requests to this root URL and its child URLs. Any resource can have a separate set of URLs.

```
package com.sample.adapter;

import java.util.logging.Logger;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;

import com.worklight.adapters.rest.api.WLServerAPI;
import com.worklight.adapters.rest.api.WLServerAPIProvider;

@Path("/")
public class JavaAdapterResource {

    //Define logger (Standard java.util.Logger)
    static Logger logger = Logger.getLogger(JavaAdapterResource.class.getName());

    //Define the server api to be able to perform server operations
    WLServerAPI api = WLServerAPIProvider.getWLServerAPI();


    //Path for method: "<server address>/Adapters/adapters/JavaAdapter/{username}"
    @GET
    @Path("/{username}")
    public String helloUser(@PathParam("username") String name){
        return "Hello " + name;
    }

}
```

`@Path("/")` before the class definition determines the root path of this resource. If you have multiple resource classes, you should set each resource a different path.

For example, if you have a `UserResource` with `@Path("/users")` to manage users of a blog, that resource is accessible via `http(s)://host:port/ProjectName/adapters/AdapterName/<em>users</em>/`.

That same adapter may contain another resource `PostResource` with `@Path("/posts")` to manage posts of a blog. It is accessible via the `http(s)://host:port/ProjectName/adapters/AdapterName/<em>posts</em>/` URL.

In the example above, because there it has only one resource class, it is set to `@Path("/")` so that it is accessible via `http(s)://host:port/Adapters/adapters/JavaAdapter<em>/</em>`.

Each method is preceded by one or more JAX-RS 2.0 annotations, for example an annotation of type "HTTP request" such as `@GET`, `@PUT`, `@POST`, `@DELETE`, or `@HEAD`. Such annotations define how the method can be accessed.

Another example is `@Path("/{username}")`, which defines the path to access this procedure (in addition to the resource-level path). As you can see, this path can include a variable part. This variable is then used as a parameter of the method, as defined `@PathParam("username") String name`.

> You can use many other annotations. See **Annotation Types Summary** here: https://jax-rs-spec.java.net/nonav/2.0-rev-a/apidocs/javax/ws/rs/package-summary.html (https://jax-rs-spec.java.net/nonav/2.0-rev-a/apidocs/javax/ws/rs/package-summary.html)

HTTP Session

Depending on your infrastructure and configuration, your MobileFirst server may be running with `SessionIndependent` set to true, where each request can reach a different node and HTTP sessions are not used. In such cases you should not rely on Java's HttpSession to persist data from one request to the next.

MobileFirst server-side API

Java adapters can use the MobileFirst server-side Java API to perform operations that are related to MobileFirst Server, such as calling other adapters, submitting push notifications, logging to the server log, getting values of configuration properties, reporting activities to Analytics and getting the identity of the request issuer.

To get the server API interface, use the following code:

```
WLServerAPI api = WLServerAPIProvider.getWLServerAPI();
```

The `WLServerAPI` interface is the parent of all the API categories: (Analytics, Configuration, Push, Adapters, Authentication).

If you want, for example, to use the push API, you can write:

```
PushAPI pushApi = serverApi.getPushAPI();
```

> For more information, review the Server-side API topics in the user documentation.

For examples of Java adapters communicating with an HTTP or SQL back end, see: