How to create and invoke custom Java code in Mirth Connect

This guide will show you how to create a custom Java class, compile/package it into a JAR, include it in Mirth Connect, and invoke it in JavaScript. Your custom code can be invoked from any JavaScript context, including the global/channel scripts, filters/transformers, and JavaScript connectors.

Requirements

A working development environment, including the JDK (Java Development Kit). Generally you should use the same version of Java that your Mirth Connect server is using.

You can look here to see how to set up Eclipse to checkout and develop Mirth Connect, but it's an optional step. For this example we'll just use a single .java file and a command-line terminal.

Creating the Java code

First create the class(es) you want to invoke from Mirth Connect. You may import internal Mirth Connect packages if you need to extend some class or implement some interface (see here for an example of extending AcceptMessage for a custom Web Service Listener), but you certainly don't need to. This example will just use a simple POJO called JarTest:

JarTest.java

```
package com.example;

public class JarTest {

    public String returnSomething() {
        return "Hello, World!";
    }
}
```

Note that it must have the package name declared. You can implement your code in a single class or using many classes.

Using a package directory structure

Put your newly created JarTest.java file in the following structure:

- src
    - com
        - example
            - **JarTest.java**
- classes

Note how there are two top-level folders, src and classes. The src folder will contain your source files (.java), and the classes folder will contain your compiled class files (.class).

Compiling your code

Now you need to convert your source code into compiled class files, which will be included in the JAR. In a command-line terminal, navigate to the top-level folder that contains your src and classes folders. Then do this:

```
javac -d classes src/com/example/*
```

If you need to include other libraries in the classpath, use the -cp option. Type "javac -help" for the full set of options.

Creating the JAR file

Now you should have JarTest.class in the classes/com/example folder, so the next step is to archive it into a JAR file. In the same top-level directory as before, do this:

```
jar -cf JarTest.jar -C classes com
```

The -c option means you're creating a new archive, and -f specifies the filename to output with. The -C option changes the working directory before adding files. Finally, the "com" at the end indicates what files/folders to include in the archive.

You can also compile source files and create a JAR in one step in Eclipse, with the export feature.

Installing and testing

You will be able to use the custom class anywhere JavaScript is used. For example, you can create a new channel that uses a JavaScript Writer destination, and use the following code:

```
var obj = new Packages.com.example.JarTest();
logger.info(obj.returnSomething());
```

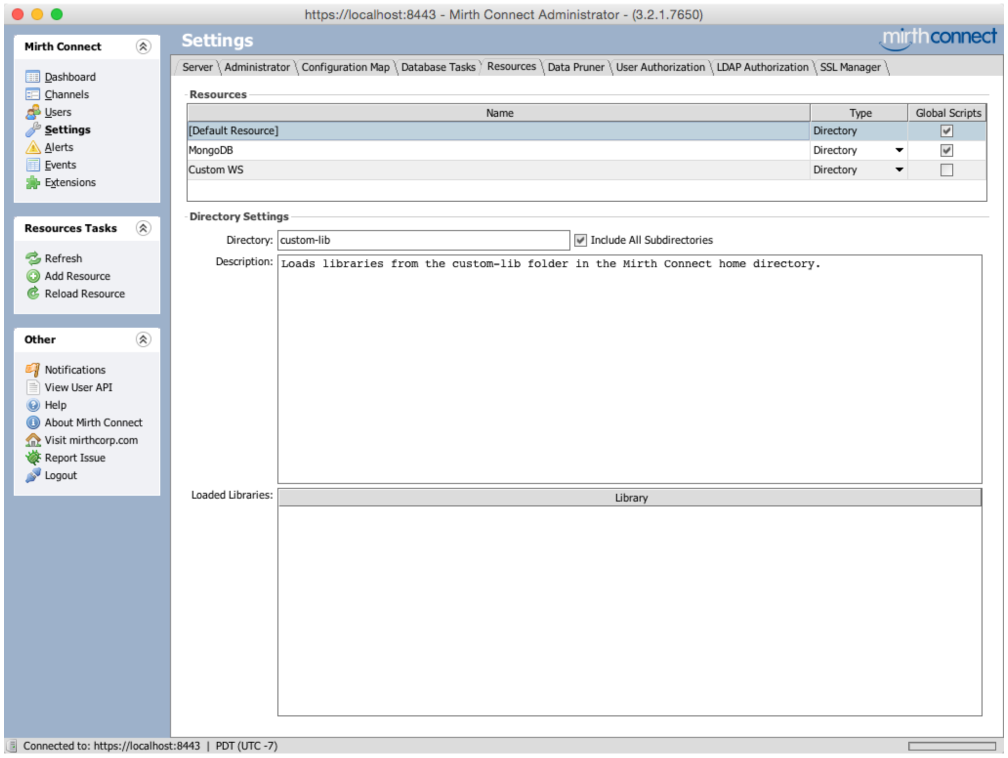
At first this channel won't yet work; we need to include the JAR that we just created.

Mirth Connect 3.1.1 or earlier

- Drop the JAR file into the lib/custom folder (for version 1.8.2 or earlier) or custom-lib (for versions 2.0.0 through 3.1.1) under your Mirth Connect home directory. This way, the classloader should be able to locate it.
- Restart the Mirth Connect service, and login to the Administrator.

Mirth Connect 3.2.0 or later

- In 3.2 or later you can define custom resources to point to any directory you want. So you can drop the JAR file into the custom-lib folder, or create a new folder anywhere on the server filesystem.
- Restarting Mirth Connect is ***not*** required. Instead, in the Administrator navigate to the Settings -> Resources tab.



Resources

Name	Type	Global Scripts
[Default Resource]	Directory	✓
MongoDB	Directory	✓
Custom WS	Directory	

Directory Settings

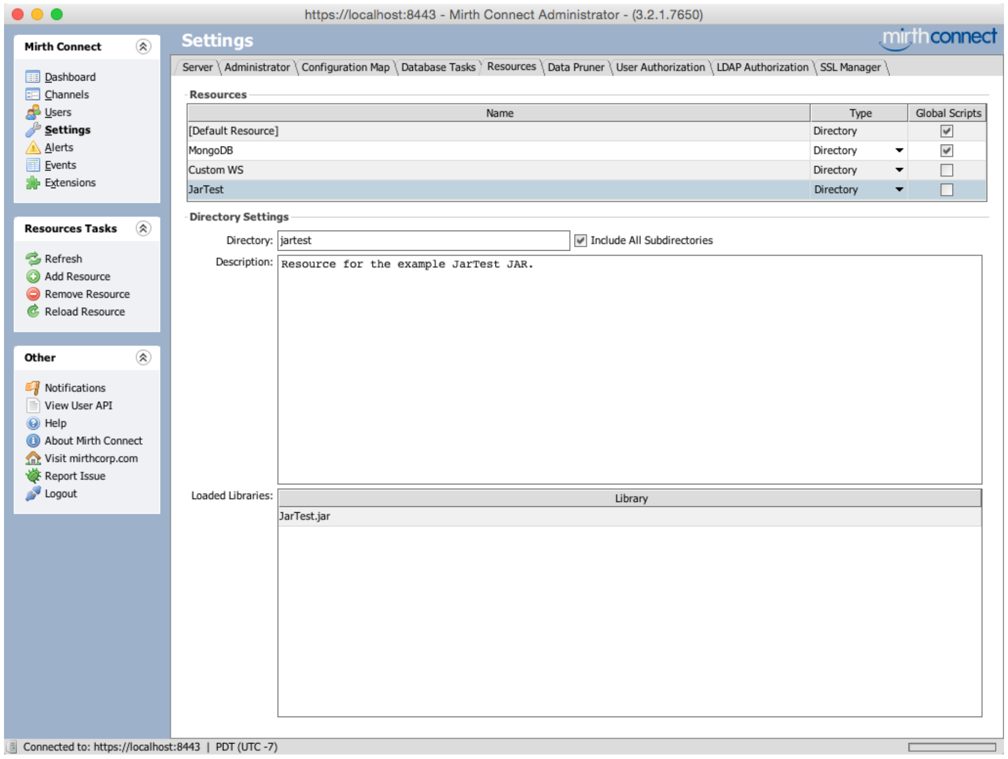
Directory: custom-lib ✓ Include All Subdirectories

Description: Loads libraries from the custom-lib folder in the Mirth Connect home directory.

Loaded Libraries:

Library

- Create a new Directory resource if needed. If you're just using the custom-lib folder, and the default resource is pointing to custom-lib (it will by default), then you only need to hit Reload Resource and the JAR will automatically be picked up. Hit the refresh button if the JAR doesn't show up initially. You should now see it in the Loaded Libraries table at the bottom.



Resources

Name	Type	Global Scripts
[Default Resource]	Directory	✓
MongoDB	Directory	✓
Custom WS	Directory	
JarTest	Directory	

Directory Settings

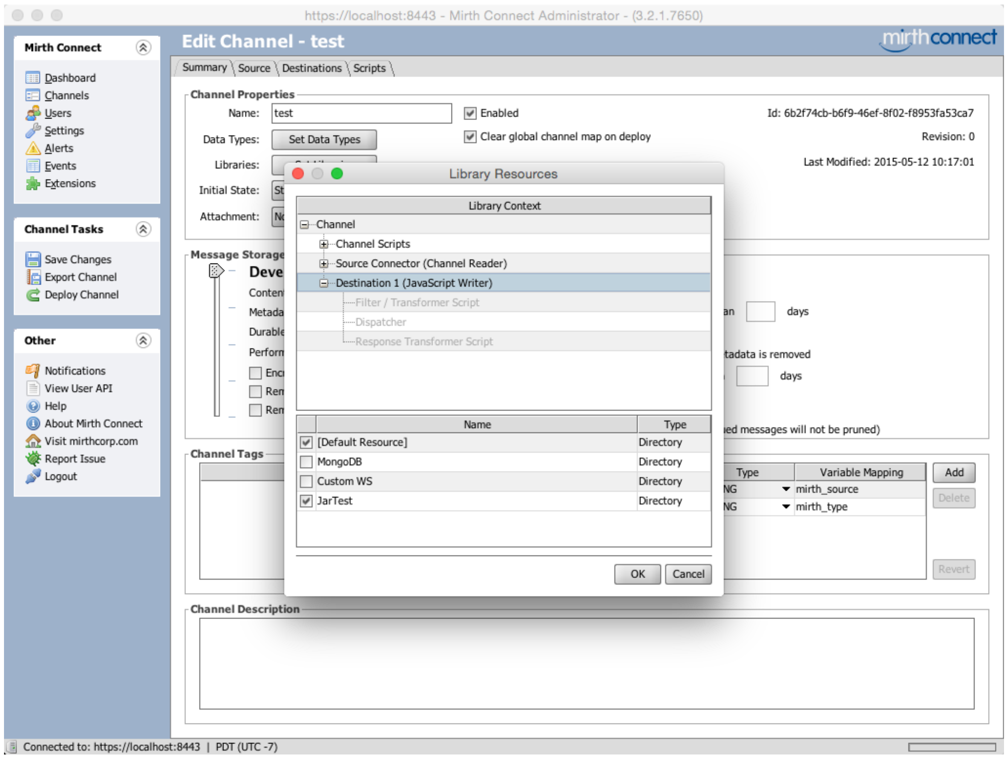
Directory: jartest ✓ Include All Subdirectories

Description: Resource for the example JarTest JAR.

Loaded Libraries:

Library
JarTest.jar

- Edit the channel you want to use the resource with, go to the Summary tab, and click the Set Libraries button. Make sure the resource is checked for the context you want to use it in. If you're using the default resource, it should already be checked by default.



Channel Properties

Name: test

[✓] Enabled

[✓] Clear global channel map on deploy

Id: 6b2f74cb-b6f9-46ef-8f02-f8953fa53ca7

Revision: 0

Last Modified: 2015-05-12 10:17:01

Library Context
- Channel
  - Channel Scripts
  - Source Connector (Channel Reader)
  - Destination 1 (JavaScript Writer)
    - Filter / Transformer Script
    - Dispatcher
    - Response Transformer Script

	Name	Type
✓	[Default Resource]	Directory
	MongoDB	Directory
	Custom WS	Directory
✓	JarTest	Directory

Connected to: https://localhost:8443 | PDT (UTC -7)

- Redeploy the channel if you made any changes to it.

Now that you've included the JAR, send a message through the channel. You should see something like this in the server log:

```
[2015-05-12 10:45:54,666]  INFO  (js-connector:?): Hello, World!
```