Android shell development

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/6.3/advanced-topics/shell-development-concepts/android-shell-development.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

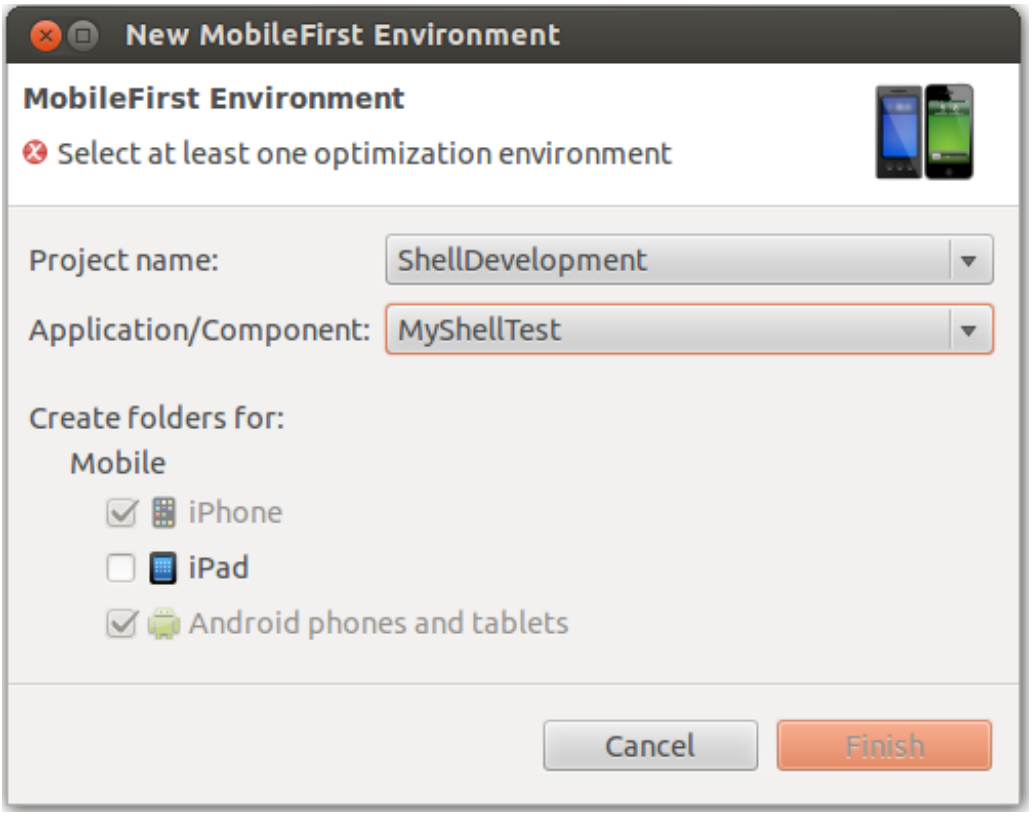
Overview

This module is a continuation of the tutorial Shell Development Concepts (../).

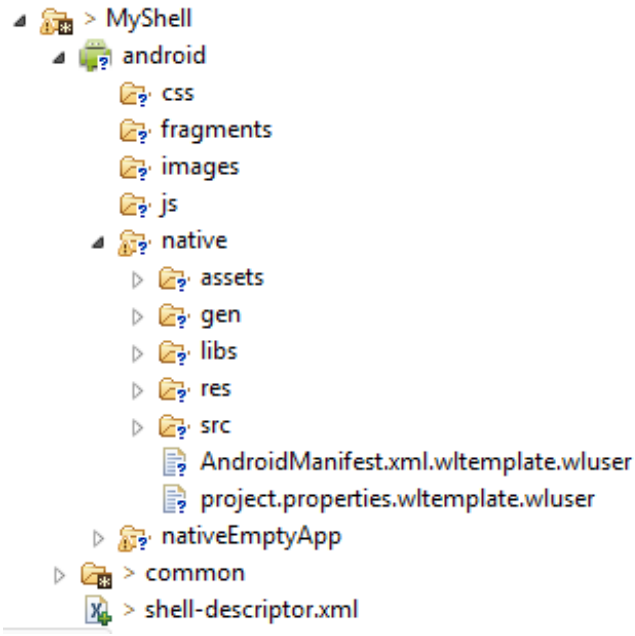
In this module, you learn how to add an Android environment to your shell component, test application, and inner application.

Adding an Android environment to a shell component

Start by adding a new Android environment to your shell component by following the same procedure as for a standard MobileFirst application.

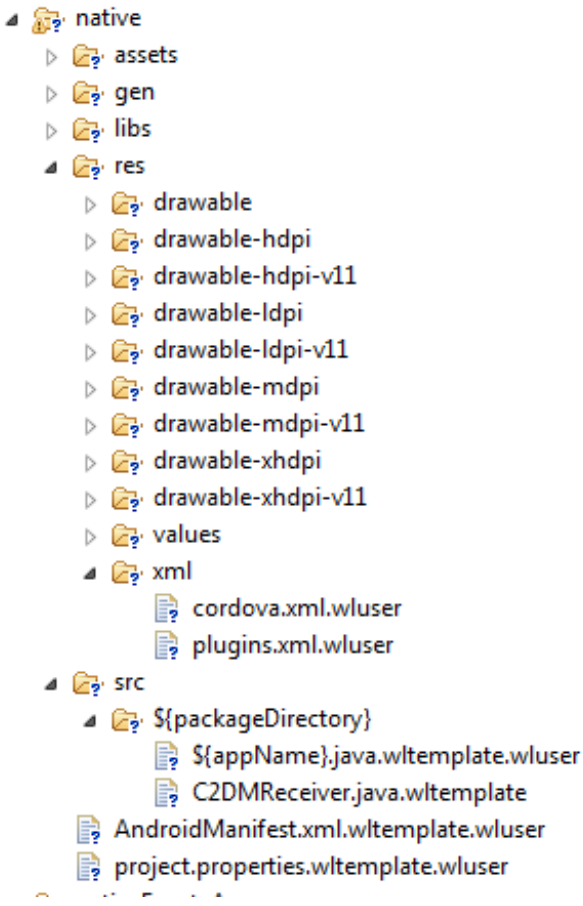


The following folder structure is created:



- **css, images, fragments** and **js** contain resources that override or extend resources from the shell component common folder.
- The **native** folder contains an application template to be used when you create an Android project from an inner application.
- The **nativeEmptyApp** folder contains application that is built from the shell component and an empty inner application as described in the Shell Development Concepts (../) tutorial.

The files in the **native** folder are templates that are used to create the inner application Android project. Some of the folder and file names contain placeholder elements that are populated during build.



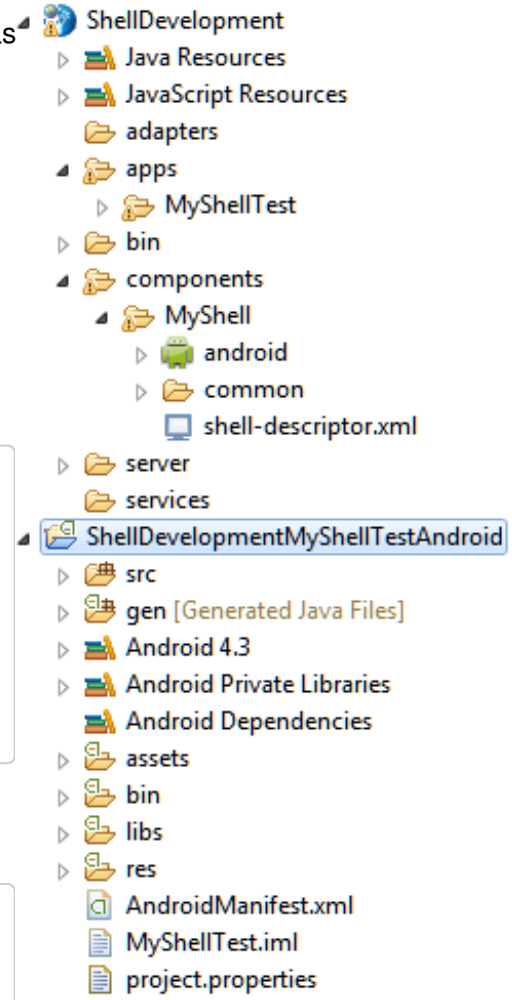
For example:

- The **${packageDirectory}** placeholder is populated with a package name used in the application.
- The **${appName}** placeholder is populated with the application name, thus creating the main application activity.

Files with the **.wluser** extensions are template files that shell developers can modify.

Adding custom Java code to a shell component

Because the *android\native* folder of a shell component is not an Android project, Eclipse does not provide advanced features such as auto-complete when you work on it directly.
The solution is to use the Android environment to create, modify, and debug Java code.
The generated Android project is added to your workspace.
Use it to work with your Java code.



Add the **com.mycustomcode** package to the generated Android project.
Add **MyCustomToaster.java** class under this package.
Add a static method to this class:

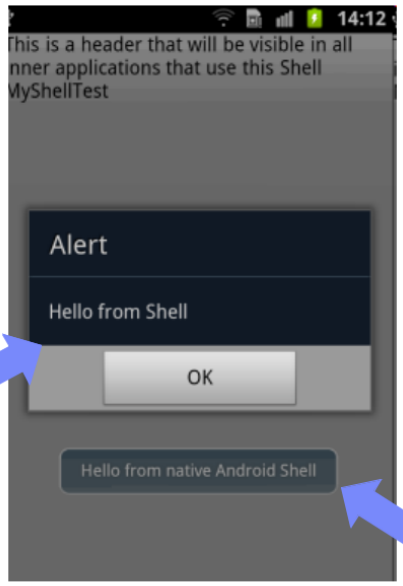
```
public class MyCustomToaster {
  public static void showToastAlert(Activity context, String te
xt) {
    Toast.makeText(context, text, 2000).show();
  }
}
```

Open your main application activity.
Start *MyCustomToast.showToastAlert()*:

```
public class MyShellTest extends WLDroidGap {
  @Override
  public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    //DeviceAuthManager.getInstance().setProvisioningDelegat
e(<replace with...>)
    super.loadUrl(getWebMainFilePath());
    MyCustomToaster.showToastAlert(this, "Hello from Androi
d Shell");
  }
}
```

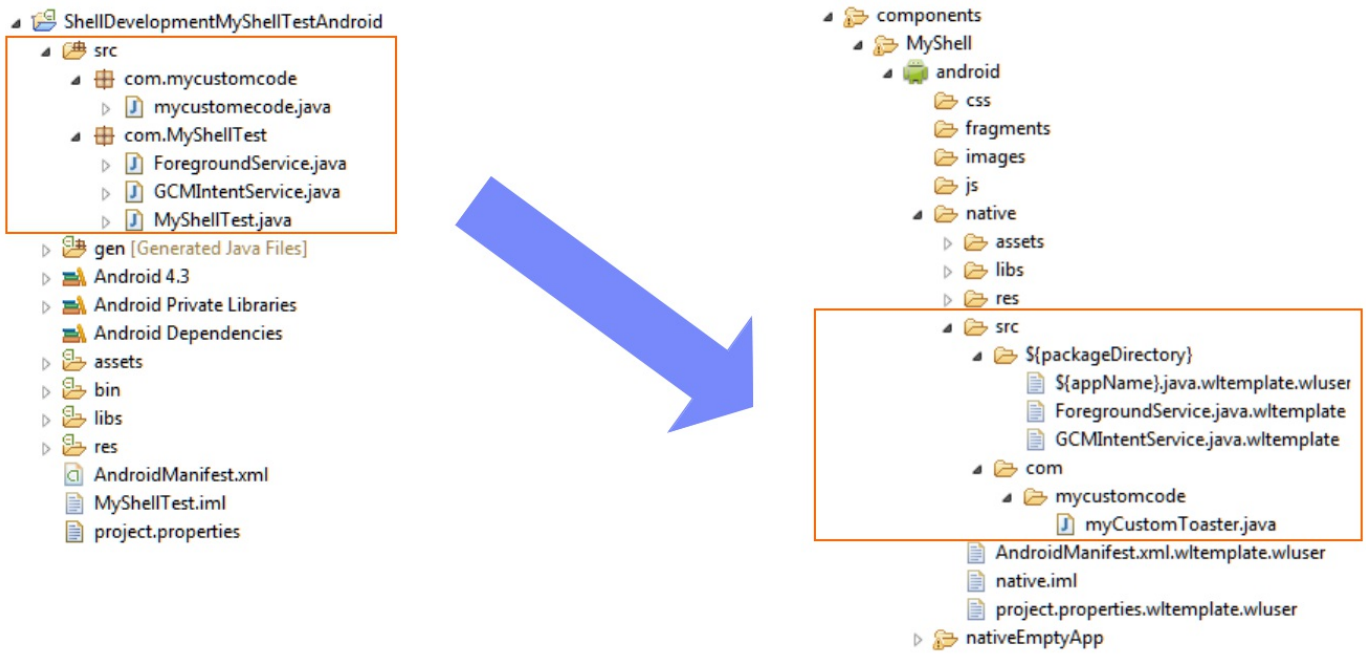
Run your application to see the implemented functions.

Here is the JavaScript alert that you implemented in the Shell Development Concepts module.



Here is a MyCustomToaster that you implemented as native Java code in previous slides.

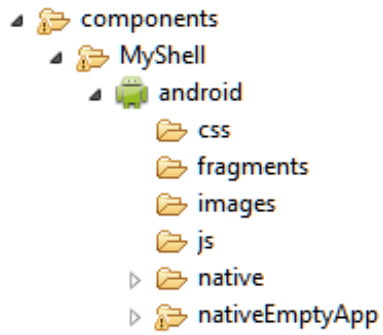
Finally, copy your Java code from the Android project that you used to develop it back to the shell component.



The custom Java classes that were added to the Android project can be copied to keep the package structure intact.
Modifications that are made to the predefined files created by MobileFirst Studio, however, must be copied manually to the corresponding template files. In this specific case, a highlighted line from the **MyShellTest.java** must be copied to **${appName}.java.wltemplate.wluser**.

The **native** folder of the test application is not being rebuilt from the shell component each time you build the Android application.
Doing so avoids overwriting the test application native code with the one in the shell component on each build, thus allowing shell developers to debug their code conveniently.
If you want your native folder to be fully re-created from a shell component, erase it in the test application, and then build and deploy the application.

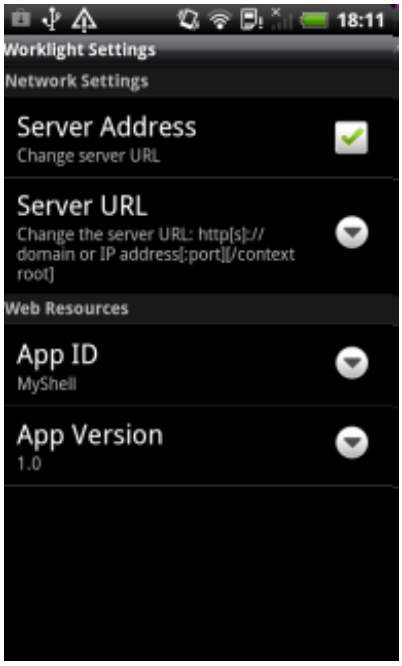
Using the NativeEmptyApp Project



NativeEmptyApp is a native application project that uses the Shell component, and that has an empty inner application.
This project can be built as an APK or IPA by a shell developer, and sent to inner application developers to use for debugging their applications.
After the NativeEmptyApp is installed on the device, an inner application developer can specify the URL of the MobileFirst Server from which to load the inner application.
Doing so helps inner application developers to test their code without the need to have native SDKs installed.
For example: to develop and test iPhone application without a Mac.
To use NativeEmptyApp, import its folder as a generic project in Eclipse.

The native empty application is automatically recognized as an Android project.

When the application is built and deployed to an Android device, click Menu and go to Settings to change the URL from which this inner application content is loaded.



Important:
NativeEmptyApp cannot load a remote inner application that has the device provisioning enabled.
NativeEmptyApp can be used only in the development environment.

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/ShellDevelopmentProject.zip) the Studio project.