Android shell development

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.1/advanced-client-side-development/shell-development-concepts/android-shell-development.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

This tutorial complements Shell Development Concepts (../).

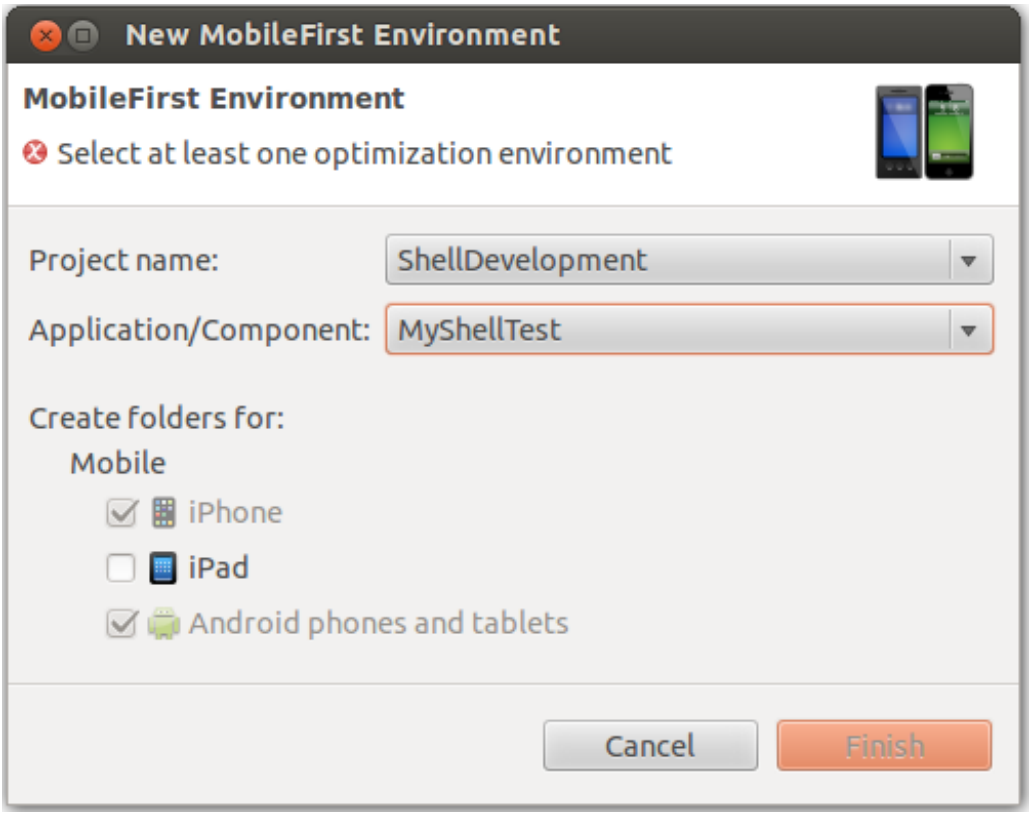
In this tutorial, you learn how to add an Android environment to your shell component, test application, and inner application.

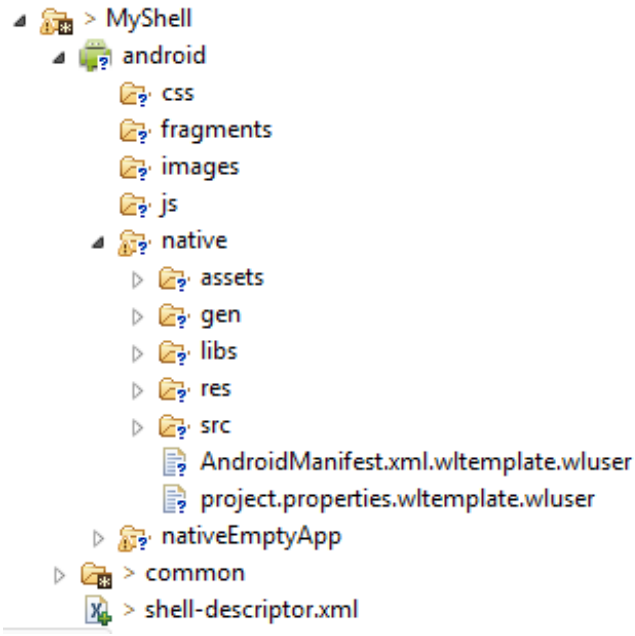
This tutorial covers the following topics.

- Adding an Android environment to a shell component
- Adding custom Java code to a shell component
- Using the NativeEmptyApp project
- Sample application

Adding an Android environment to a shell component

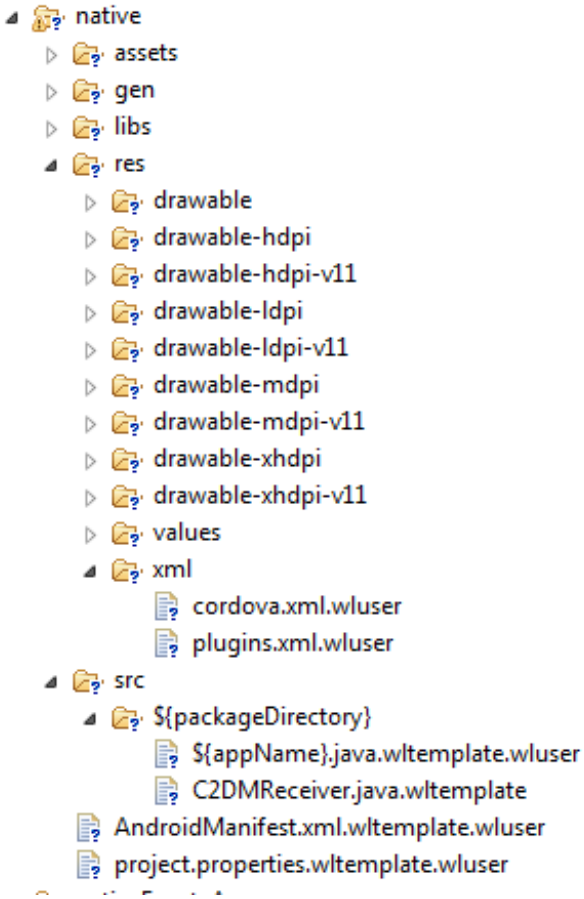
Start by adding a new Android environment to your shell component. Follow the same procedure as for a standard MobileFirst application.





The following folder structure is created:

- The `css`, `images`, `fragments`, and `js` folders 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 the application that is built from the shell component and an empty inner application as described in the Shell Development Concepts (../) tutorial.

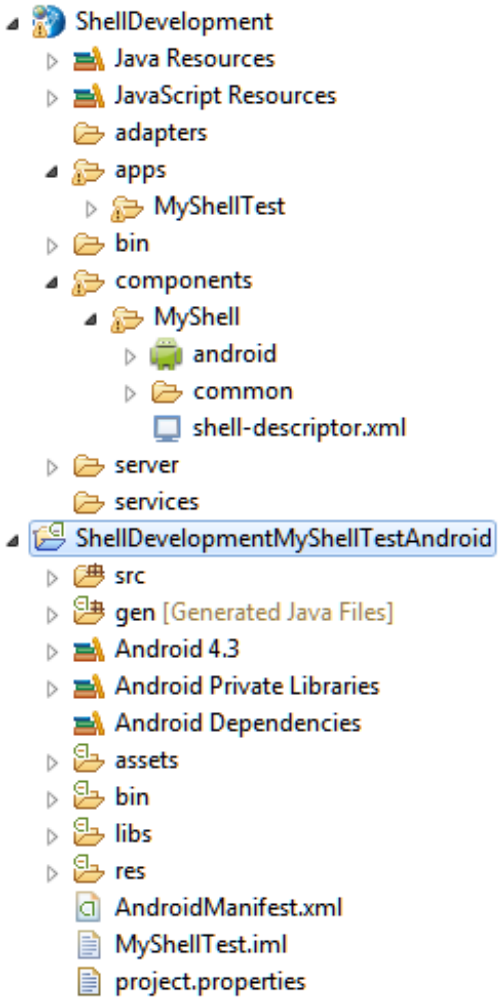


The files in the `native` folder are templates that are used to create the inner application Android project. Some of the folder and file names contain placeholder elements that are populated during the build proces. For example:

- 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` name extension 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 autocomplete 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.

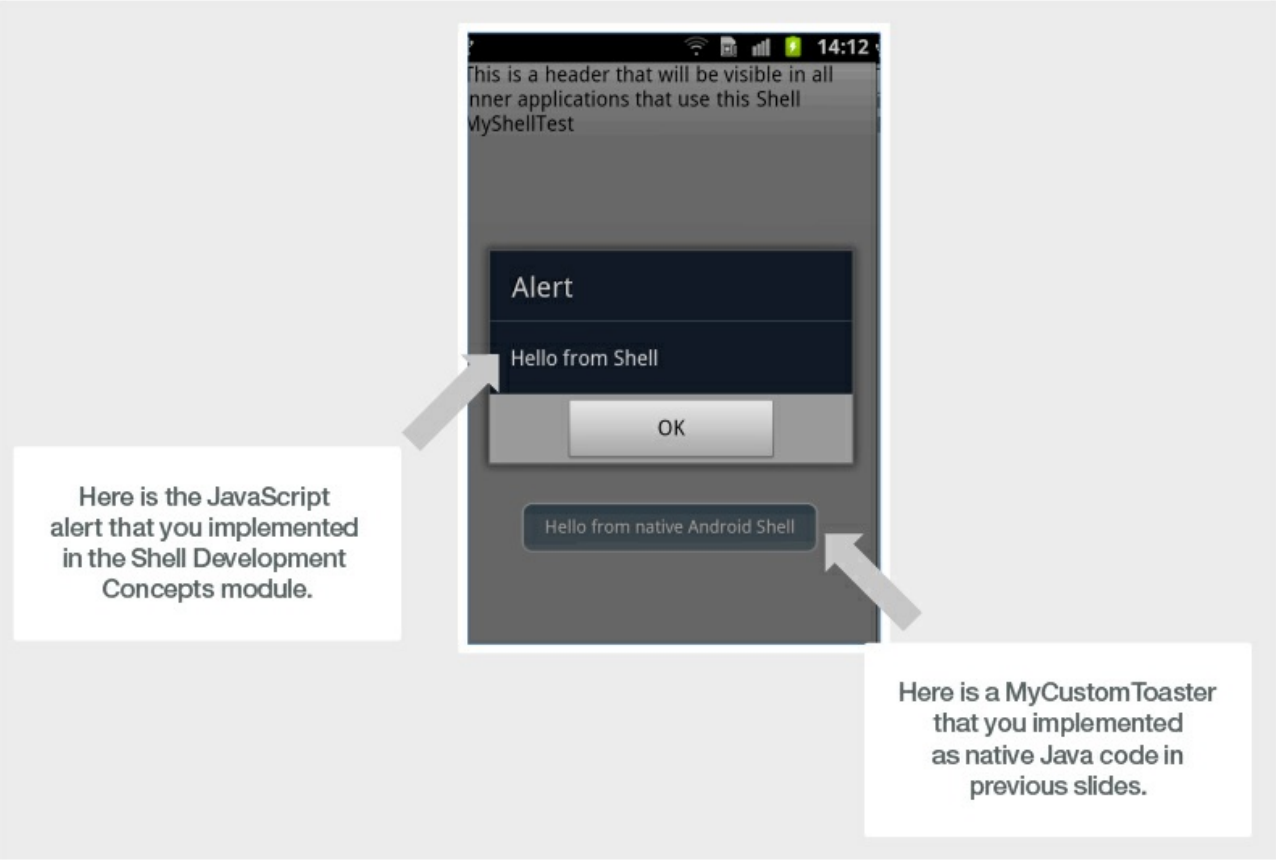
1. Add the `com.mycustomcode` package to the generated Android project.
2. Add the `MyCustomToaster.java` class under this package.
3. Add a static method to this class:

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

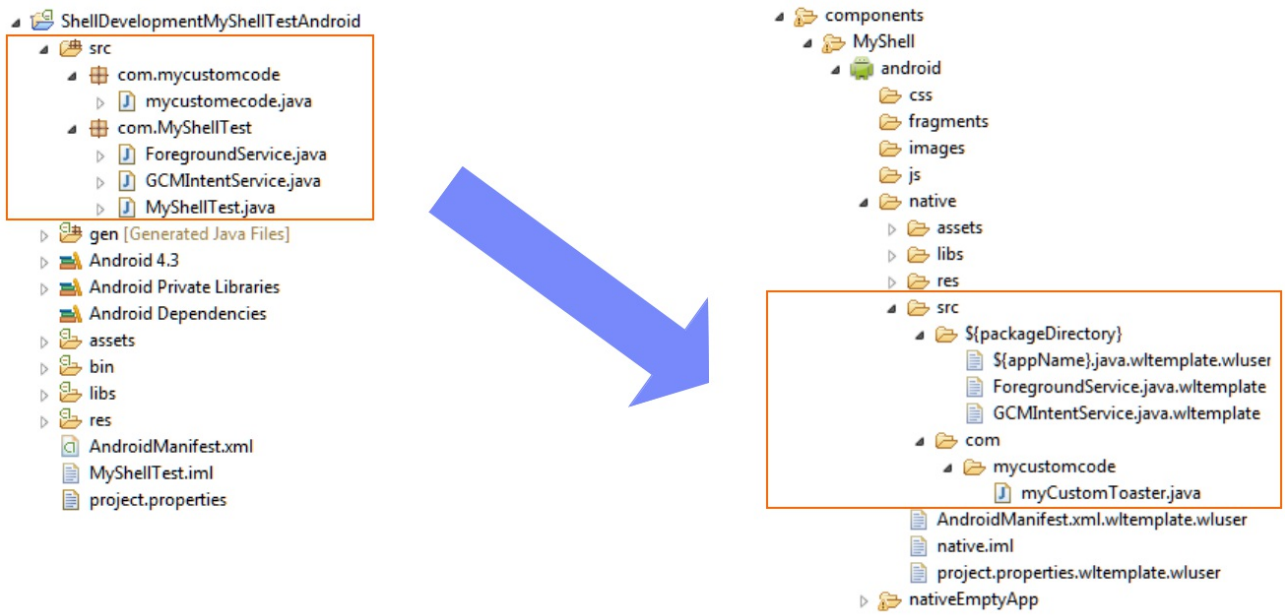
4. Open your main application activity.
5. Start `MyCustomToast.showToastAlert()`:

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

6. Run your application to see the implemented functions.



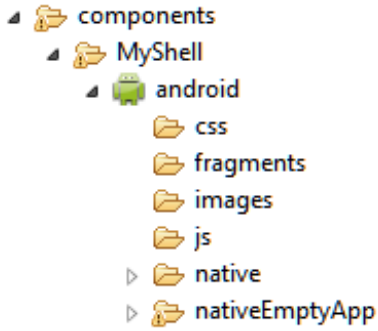
7. Finally, copy your Java code from the Android project that you used to develop it back to the shell component.



You can copy the custom Java classes that were added to the Android project 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` file must be copied to the `${appName}.java.wltemplate.wluser` file.

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 enabling 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



The `NativeEmptyApp` project 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 for debugging their applications.
After the `NativeEmptyApp` project is installed on the device, an inner application developer can specify the URL of the MobileFirst Server instance from which to load the inner application.
Doing so helps inner application developers to test their code without having native SDKs installed. For example: to develop and test iPhone application without a Mac.
To use the `NativeEmptyApp` project, 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**, then **Settings** to change the URL from which this inner application content is loaded.

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

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/ShellDevelopment/tree/release71) the MobileFirst project.