iOS shell development

Overview

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

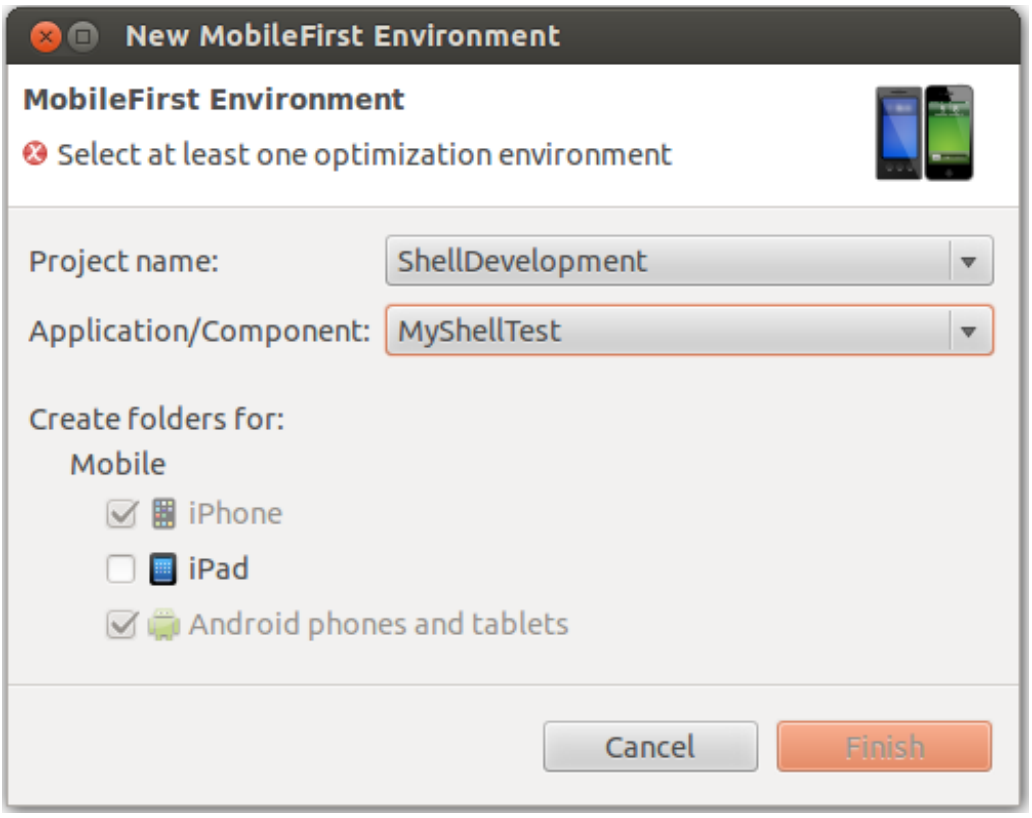
In this tutorial, you learn how to add an iOS environment to your shell component, test application, and inner application.

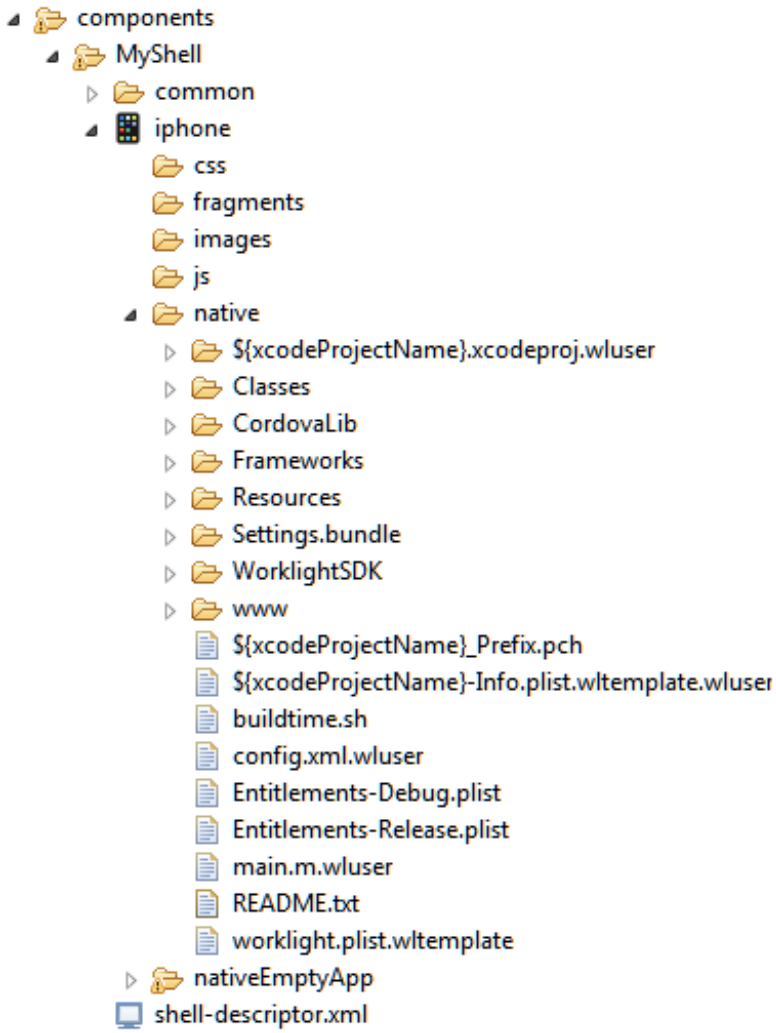
This tutorial covers the following topics.

- Adding an iOS environment to a shell component
- Adding custom Objective-C code to a shell component
- Using the NativeEmptyApp project
- Sample application

Adding an iOS environment to a shell component

Start by adding an iPhone environment to your shell component. Follow the same procedure as for a standard IBM MobileFirst Platform Foundation 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 iOS project from an inner application.
- The **`nativeEmptyApp`** folder contains an 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 iOS project.
Some of the folder and file names contain placeholder elements that are populated during the build.
For example:

- The placeholder *${xcodeProjectName}.xcodeproj.wluser* is populated with a package name used in the application.
- The *${xcodeProjectName}-Info.plist.wltemplate.wluser* is populated with the application name, thus creating the main application `plist` file.

Files with the `.wluser` name extension are template files that shell developers can modify.

Adding custom Objective-C code to a shell component

Because the `iphone\native` folder of a shell component is not an iOS project, advanced features such as autocomplete are not provided when you work on it directly.
The solution is to use the iPhone environment of the test application to create, modify, and debug the

Objective-C code.
The generated iOS project is created under the test application `native\` folder.
Use it to work with your Objective-C code.

1. Open the generated iOS project in Xcode.
2. Add an Objective-C `MyCustomAlert` class in the `Classes` folder.
3. Add a method signature to the `MyCustomAlert.h` file and a method implementation to the `MyCustomAlert.m` file:

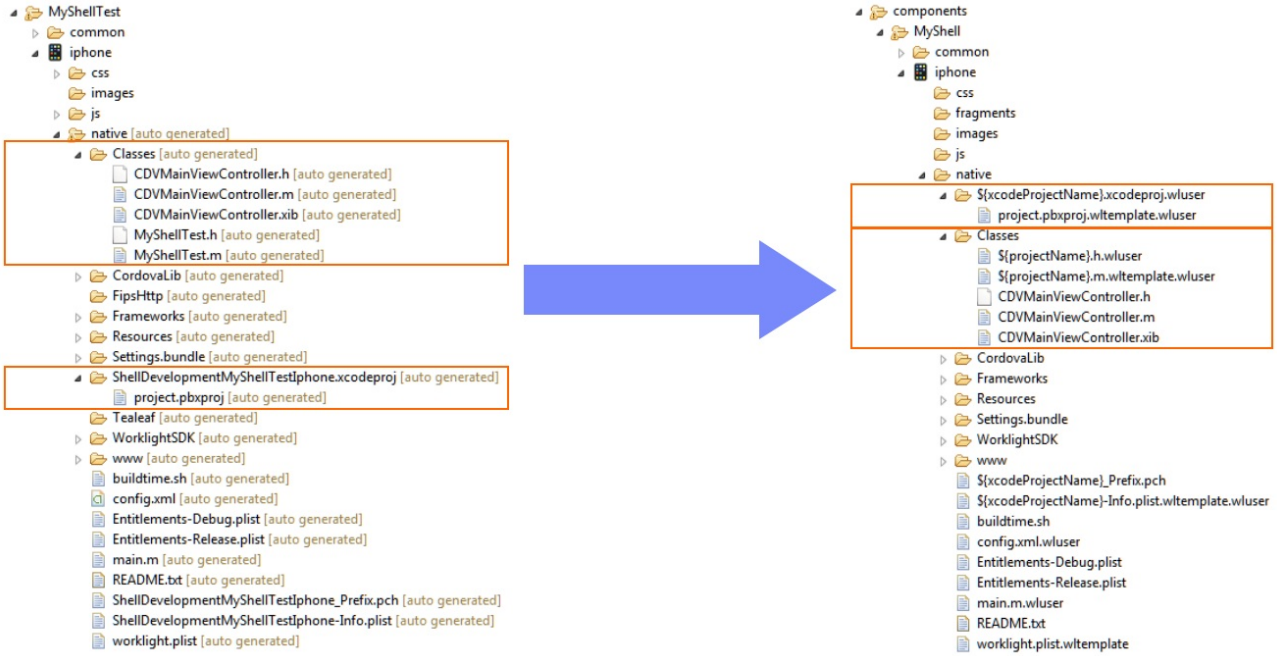
```
#import "MyCustomAlert.h"

@implementation MyCustomAlert
+(void)showUIAlert:(NSString *)text{
  UIAlertView *alert = [[UIAlert alloc] initWithTitle:@"Native Alert
"
        message:text
        delegate:nil
        cancelButtonTitle:@"Close"
        otherButtonTitles:nil];
  [alert show];
  [alert release];
}
@end
```

4. Import `MyCustomAlert.h` and call this method from the `viewDidLoad` method of the application `ViewController` instance:

```
- (void)viewDidLoaad {
  [super viewDidLoad];
  [MyCustomAlert showUIAlert:@"Hello from native iOS Shell"];
}
```

5. Run your application to see the implemented functions.

6. Finally, copy your Objective-C code from the iPhone project that you used to develop it back to the shell component.



You can copy the custom Objective-C classes added to the iPhone project to keep the folder structure intact. Xcode stores its own project structure in a `project.pbxproj` file. Therefore, the content of this file must also be copied from the test application to the shell component.

The `native` folder of the test application is not being rebuilt from the shell component each time you build the iOS 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 recreated 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 an iPhone application without a Mac.
To use the `NativeEmptyApp` project, open it as an Xcode project.

When the application is built and deployed to an iOS device, 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 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.