Windows Module Project

Introduction

This guide covers how to manage your module project as well as how to add third-party frameworks and bundle assets with your module.

Prerequisite

> Support for Windows 8.1 and Windows Phone SDKs has been deprecated as of SDK 6.3.0.GA and has be removed in SDK 7.0.0.GA.

To develop an Windows-based Module, you'll need all of the software required to build a Titanium application for Windows:

- Titanium SDK
- Supported versions of Visual Studio and the Windows Phone SDK, as described in Installing the Windows Phone SDK
- Studio or the Appcelerator Command-Line Interface (CLI) for creating modules, and building and running test applications

Like Windows application development, Windows module development is only supported on Windows.

Project Structure

When you create a new project, it generates the following directories and files. Titanium expects to find files in certain directories with a specific naming convention.

Filename / Directory	Description / Purpose
LICENSE	The module's full license text; this will be distributed with the module to let other developers know how they can use and redistribute it.
README	The file that gives a short explanation of the module project. This file is not distributed with your module.
assets	The directory where you should place module assets, such as images.
documentation	The directory where you should place your module documentation for end-users. The file index.md is a Markdown-formatted template file that you should use when writing your module documentation. You may also write your documentation using the TDoc Specification. This is only required for module distributed in the Appcelerator Marketplace. You can safely ignore this directory if you do not intend to distribute your module.
example	The directory where your module example(s) should go. The file app.js will be generated to include a sample loading of your module in a test window. This file can be used for quickly testing your module as well as give an example to end-users on how to use your module. This directory is distributed with your module.
windows/include	The directory where your C++/CX header classes should go, used by the Visual Studio compiler. By default, when you create a new project, you are given a boiler plate module class (`ModuleId.hpp`). These files are not distributed with your module. For information about these files, see Windows Module Architecture.
windows/src	The directory where your C++/CX implementation classes should go, used by the Visual Studio compiler. By default, when you create a new project, you are given a boiler plate module class (`ModuleId.cpp`). These files are not distributed with your module. For information about these files, see Windows Module Architecture.
windows/cmake	The directory which contains configuration files used by cmake. These files are not distributed with your module.

windows/CMakeLists.txt	A special file that describes configuration used about your module used by cmake. These files are not distributed with your module.
windows/manifest	A special file that describes metadata about your module and used by the Titanium compiler. This file is required and is distributed with your module.
windows/Windows10.ARM	The directory which contains your Windows 10 project for ARM. Open this directory to launch your module project in Visual Studio. This directory is not distributed with your module.
windows/Windows10.Win32	The directory which contains your Windows 10 project for Win32. Open this directory to launch your module project in Visual Studio. This directory is not distributed with your module.
windows/platform	Files in this folder are copied directory into the Windows build directory (`<project-dir>/build/windows`) when the Windows app is compiled. For example you can place res strings under `windows/platform/Strings` folder, and place drawable files under `windows/platform/Assets` folder. Files in this directory are copied directly on top of whatever files are already in the build directory, so please be careful that your files don't clobber essential project files or files from other modules.
windows/timodule.xml	Titanium module configuration file. The format is described in tiapp.xml and timodule.xml Reference. This file is not currently supported by Windows modules.

Since Release 3.3.0, the CLI creates a module project that contains multiple platforms. Each platform contains its own folder with platform-specific resources and common folders for assets, documentation and example.

Prior to Release 3.3.0, none of the previous listed folder are contained in an `windows` folder.

Manifest File

Titanium module metadata is described in a special text file named `manifest`. This file is a simple key/value property format.

Before you distribute your module, you must edit this manifest and change a few values. Some of the values are pre-generated and should not be edited. These are noted with the comment before them. In the manifest file, any line starting with a hash character (#) is ignored. The following are the descriptions of each entry in the manifest:

Key	Description/Purpose
version	This is the version of your module. You should change this value each time you make major changes and distribute them. Version should be in the dotted notation (X.Y.Z) and must not con-tain any spaces or non-number characters.
architectures	The binary architectures the module supports as a delimited list. Example: ARM x86
description	This is a human-readable description of your module. It should be short and suitable for display next to your module name.
author	This is a human-readable author name you want to display next to your module. It can simply be your personal name, such as "Jeff Haynie" or an organizational name such as "Appcelerator".
apversion	The is a generated value for the required module API version that was used when creating your module. The current API version for new Windows modules is 7.
license	This is a human-readable name of your license. You should use a short description such as "Apache Public License" or "Commercial".
copyright	This is a human-readable copyright string for your module. For example, "Copyright (c) 2010 by Appcelerator, Inc."
name	This is a read-only name of your module that is generated when you created your project. You must not edit this value.
moduleid	This is a read-only module id of your module that is generated when you created your project. You should not edit this value. NOTE: you must generate a unique id. We recommend using your reverse-DNS company name + module_name as a pattern to guarantee uniqueness. The Titanium Marketplace will only allow unique module ids when distributing modules. If you must edit this value, you must also edit the value in your module implementation file.
guid	This is a read-only unique module id for your module that is generated when you created your project. You must not edit this value.
platform	This is a read-only platform target of your module that is generated when you created your project. You must not edit this value.
minsdk	The is a generated value for the minimum Titanium SDK version that was used when creating your module. The current minimum version for new Windows modules is 8.0.0.

CLI Tasks

Create a New Module Project

To create a new module project, run the following Titanium CLI command:

```
appc new -d /PATH/TO/WORKSPACE -n <MODULE_NAME> --id <MODULE_ID>
### when prompted for the project type, select "Titanium Module"

### Example
$ ti create -d . -n test --id com.example.test
Appcelerator Command-Line Interface, version 0.2.230
Copyright (c) 2014-2015, Appcelerator, Inc.  All Rights Reserved.
? What type of project are you creating?
  Native App
  Arrow App
 Titanium Module
```

If you omit any of the options, the CLI will prompt you to enter them.

Build and Package the Module

Next, build the module and package it. This process produces a ZIP file containing a binary library with unprocessed module assets, example code and documentation.

CLI Instructions
From a terminal, go to the module's windows directory and run the `appc ti build -p windows -T ws-local`:

```
cd test/windows
ti build -p windows -T ws-local
## When you build module for Windows 10, use "ti build -p windows -T ws-local
--wp-sdk 10.0"
```

After the build completes, unzip the built module in the Titanium SDK home path: (`C:\ProgramData\Titanium`).

After the build completes, you should have a ZIP file in the `windows` directory and see the following message in the console:

```
** BUILD SUCCEEDED **
```

With the ZIP file, you can either:

- Uncompress it in the Titanium SDK home path to install the module globally for all your Titanium applications
- Uncompress it in a Titanium project's parent directory to install the module locally for that one Titanium application
- Distribute the ZIP file

Studio Tasks

Create a New Module Project

1. From the menu, select **File** > **New** > **Mobile Module Project** to open the **New Mobile Module Project** dialog.
2. In the **Project name** field, enter a name for the module.
3. In the **Module Id** field, enter a module ID for the module.
4. In **Deployment Targets**, select **Windows**.

5. Click **Next.**
6. In the **Module Manifest File** page, enter information about your module, such as the license information, version number, etc. You can also edit this information in the `manifest` file later.
7. Click **Finish**.

Build and Package the Module

1. Select your module folder in the **Project Explorer** view.
2. Verify **Package** and **Windows Module** are displayed in **Launch Mode** and **Launch Target**, respectively.
3. Click the Package icon to open the **Package Windows Module** dialog.
4. In **Output Location**, select either
    a. **Titanium SDK** to install the module in the Titanium SDK home path to be accessed by any Titanium application
    b. **Mobile App Project** and choose an application to install the module locally that can be accessed by one that Titanium application
    c. **Location** and enter a path to copy the ZIP file to for distribution
5. Click **Finish**.

Test the Module

To test a module:

1. Create a new Titanium Classic or Alloy project.
2. Install the module to either the Titanium SDK home directory or in the project.
3. Add the module as a dependency to the project.
4. Load the module and make module API calls.

Next Steps

- For information about Windows Runtime Background Service API, see Windows.ApplicationModel.Background namespace
- For a general overview of background tasks in Windows Store apps, see the Introduction to Background Tasks whitepaper
- For example code that shows how to implement background tasks, see the Background Task Sample.