Tutorial #1 for Textual 6

A “plugin” (also known as an “addon” or “extension”) is a bundle that is loaded when the app launches. Plugins, which are written in in Objective-C or Swift, have virtually unlimited access to Textual's codebase. They can add new features, modify the behavior of features that already exist, and much, much more.

Introduction

This tutorial will guide you through the process of configuring, building, and testing a plugin.

The plugin created by this tutorial adds a new menu item to the context menu available by control clicking a user.

The plugin, including source code, created by this tutorial is available for download here.

Prerequisite

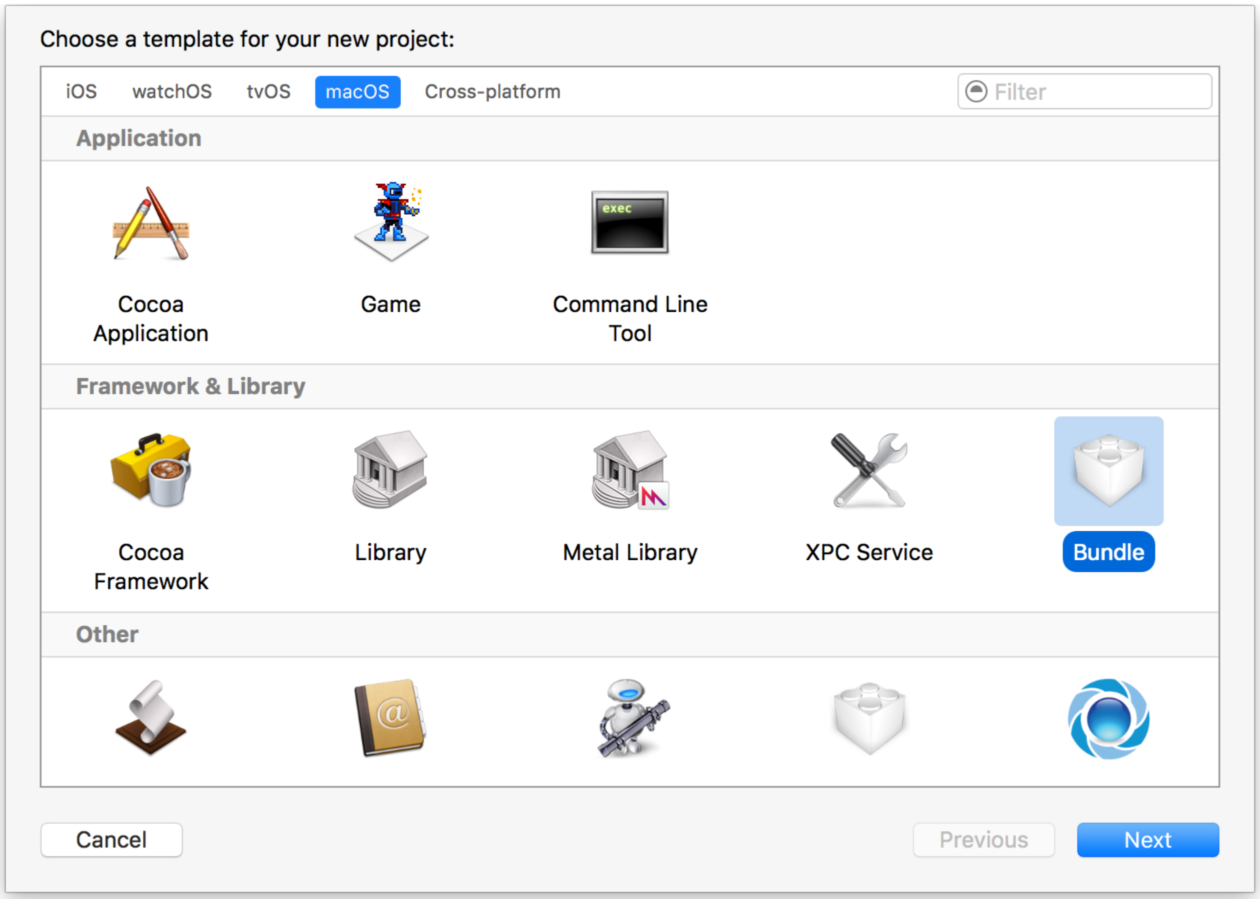
This document assumes that you, the reader, have experience with Objective-C or Swift, and Xcode.
This document uses technical jargon that may be unfamiliar to those that are not.

Copyright

Step #1: Configuring the Project

With **Xcode** open, begin a new project using the keyboard combination **Shift Command N** (⇧⌘N)

In the new project dialog, select the template named **Bundle** available under the **Framework & Library** section.



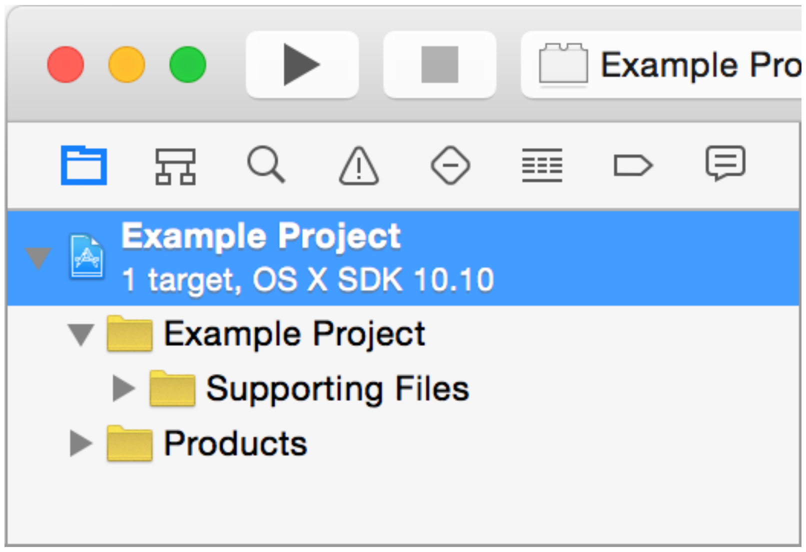
Continue by clicking the **Next** button.

Fill in any remaining information that Xcode may ask for such as the project name.

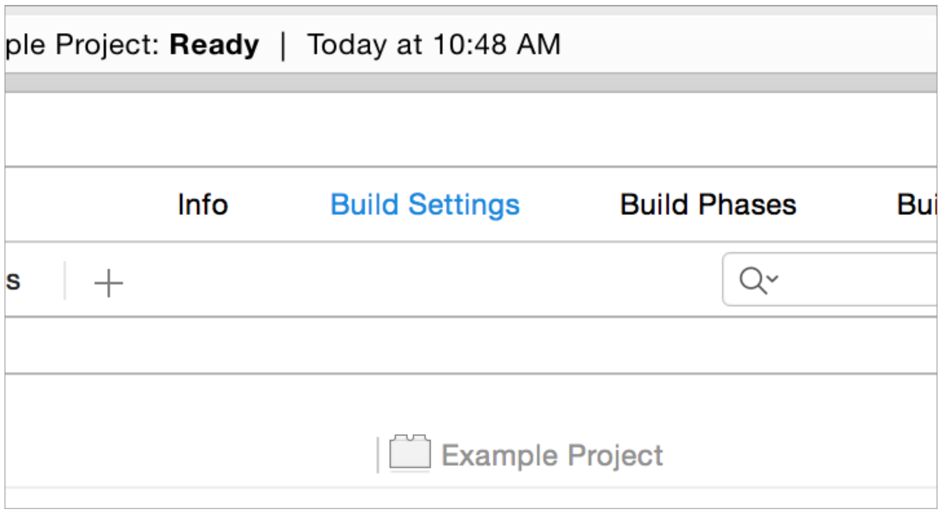
Double check all information then click the **Finish** button to complete the setup process.

Step #1: Configuring the Project cont.

When setup completes, select the project in Xcode.



With the project selected, move focus to the **Build Settings** tab of the project.



The **Build Settings** tab contains dozens of configuration options for the project.

Move focus to the following configuration options and modify each to match the values presented below.

Configuration Option	Configuration Value
Bundle Loader	/Applications/Textual.app/Contents/MacOS/Textual
Framework Search Paths	"/Applications/Textual.app/Contents/Frameworks/**"
Header Search Paths	"/Applications/Textual.app/Contents/Headers/**"

The configuration options **Framework Search Paths** and **Header Search Paths** may already contain a value. If these options already contain a value, then add the values seen above alongside those that already exist.

Step #2: Importing Headers

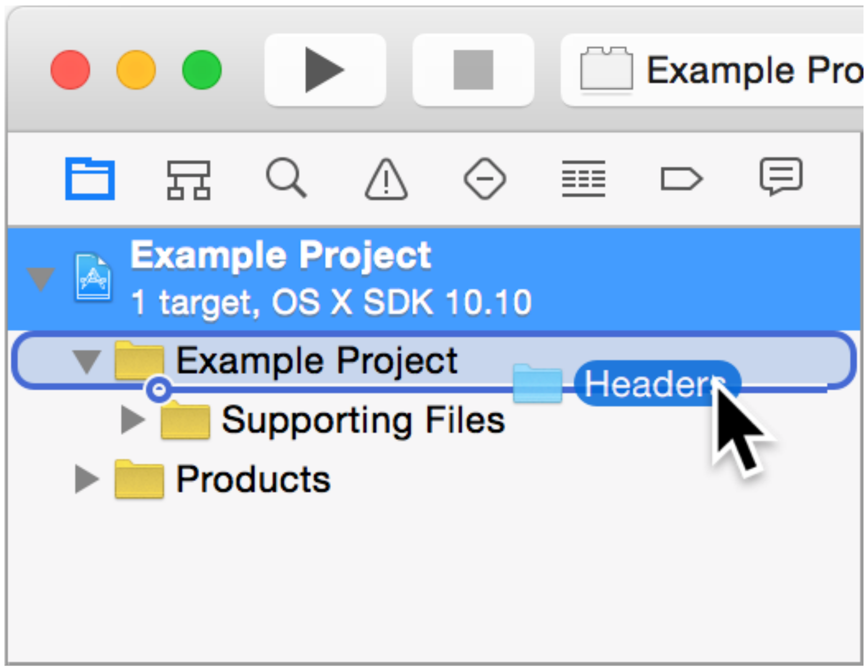
It is recommended that you manually add Textual's header files to the project in Xcode. Adding these files to the project will make them much easier to access during the development process.

With **Finder** open, move to the **Applications** folder of OS X and find **Textual.**

Control click (right click) Textual and select the menu item labelled **Show Package Contents**.

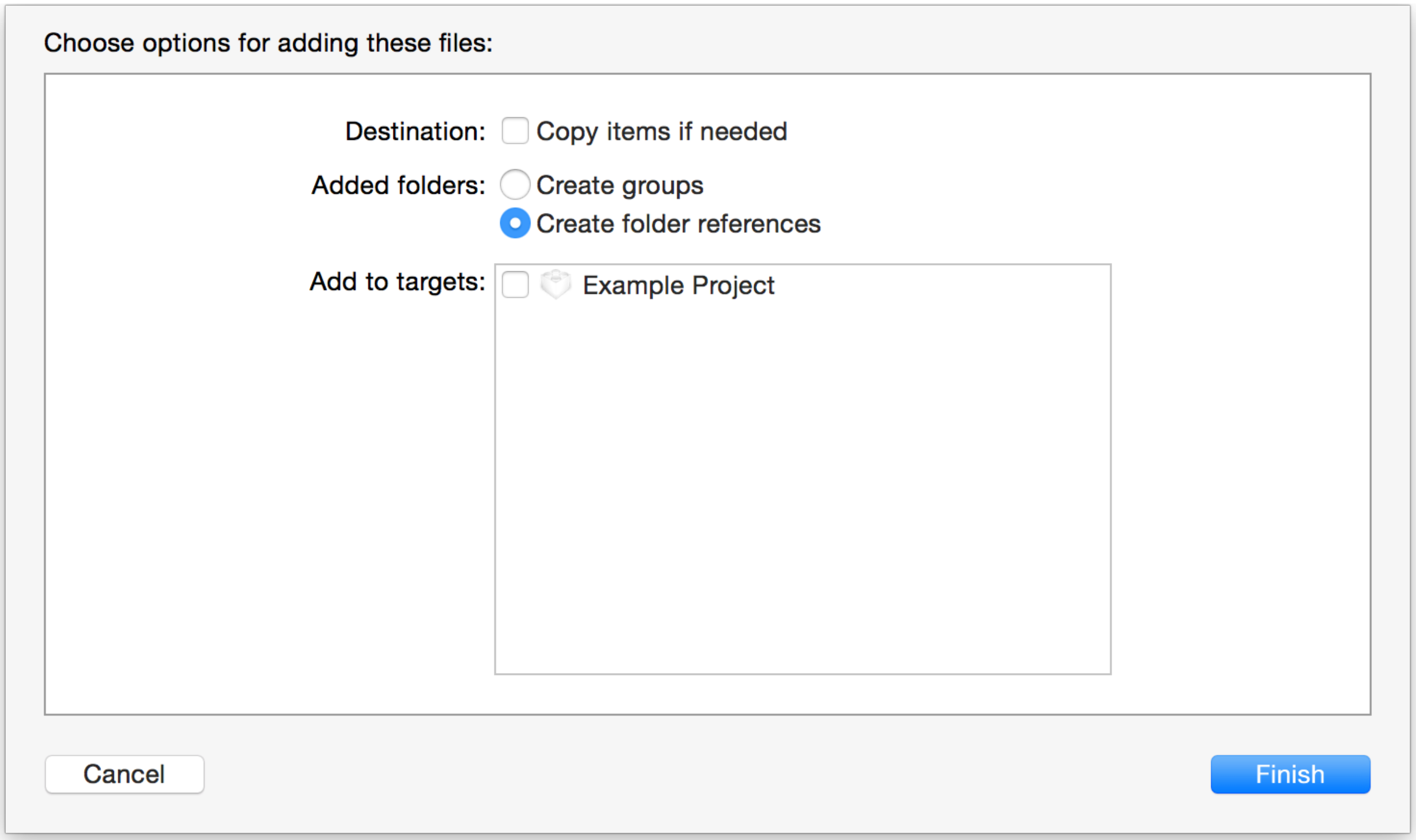
Finder will present a new window that contains a single folder named **Contents**. Double click this folder to open it.

Inside the **Contents** folder is a folder named **Headers**. Drag the **Headers** folder into Xcode and drop it onto the file list on the left side of the window in order to add it to the project.



Step #2: Importing Headers cont.

After dropping the folder into Xcode, a dialog will appear with several options for adding the files.


Choose options for adding these files:

Destination: ☐ Copy items if needed

Added folders: ○ Create groups [✓ ● Create folder references]

Add to targets: ☐ Example Project

In the dialog, uncheck the option to **Copy items if needed**.

Next, select the option to **Create folder references**.

Lastly, **uncheck all targets**. By unchecking all targets, Xcode will not copy the header files during the build process. The header files do not need to be packaged with the built product for the plugin to function correctly.

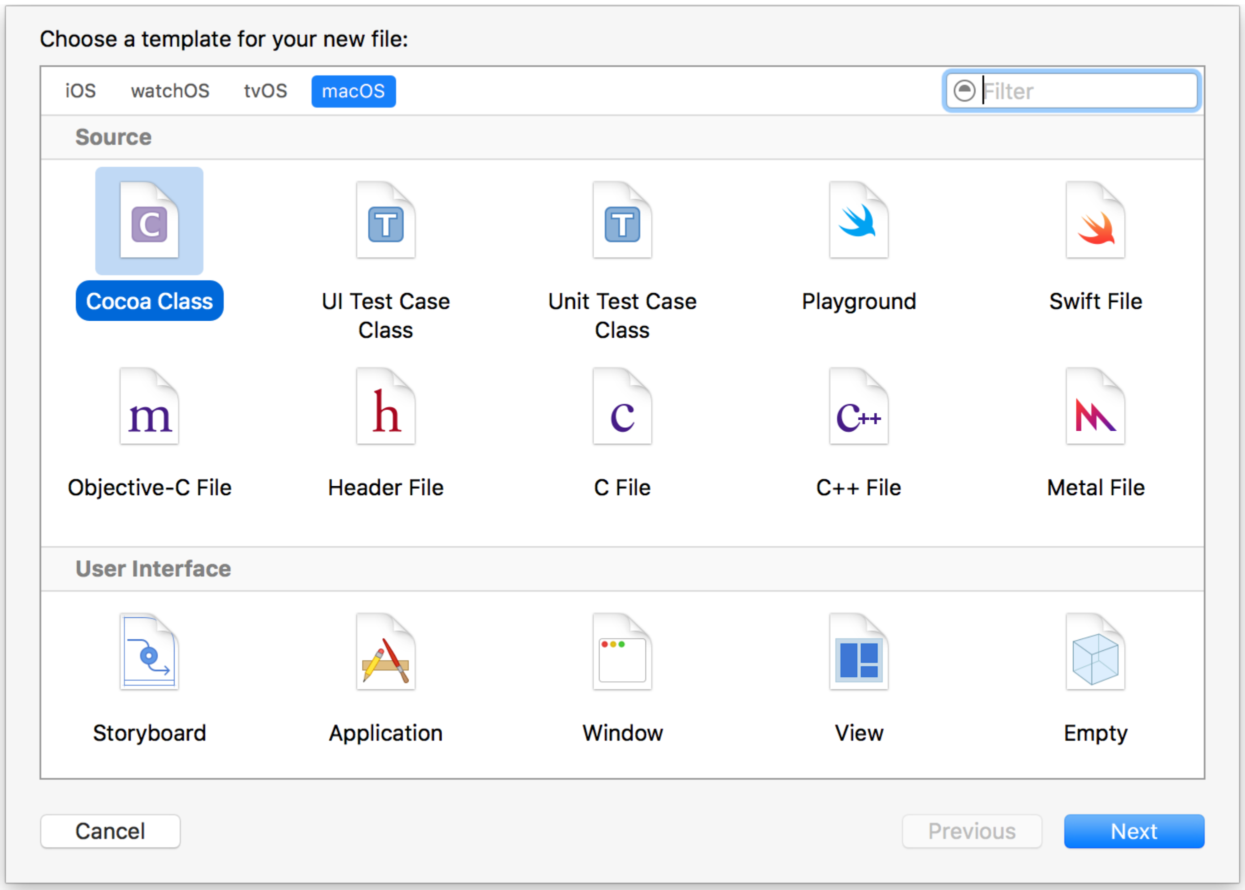
Finish adding the files to the project by clicking the **Finish** button.

Step #3: Creating the Principal Class

When Textual loads a plugin, it allocates a new instance of the plugin's "principal class" and retains a reference to it. The principal class is the only class in your plugin that Textual will communicate with.

Begin by creating a new file in Xcode using the keyboard shortcut **Command N** (⌘N).

In the new file dialog, select the template named **Cocoa Class** available under the **Source** section. Continue by clicking the **Next** button.



When prompted for the name of the class (new file), enter a name that begins with “**TPI_**” - “TPI_” is a class prefix reserved for plugins. Using this prefix is not required, but it will allow your plugin to avoid collisions with other classes.

Step #4: Prepare the Principal Class

Move focus to the header file of the principal class.

Near the top of the header file, `#import` Textual's main header file which is named **TextualApplication.h**.
This is the only header file that needs to be imported.

Next, declare that the principal class conforms to the <`THOPluginProtocol`> protocol.

```
#import <Foundation/Foundation.h>

#import "TextualApplication.h"

@interface TPIExamplePrincipalClass : NSObject <THOPluginProtocol>

@end
```

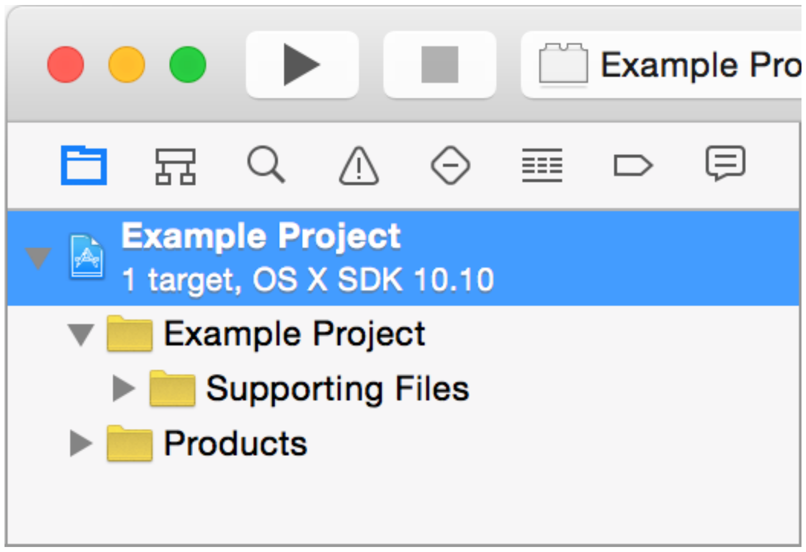
The <`THOPluginProtocol`> protocol, which is documented here, defines a set of features that Textual offers directly to plugin authors to make the process of adding certain features much more convenient.

However, a plugin is not limited to this protocol. With enough effort, a plugin can do almost anything.

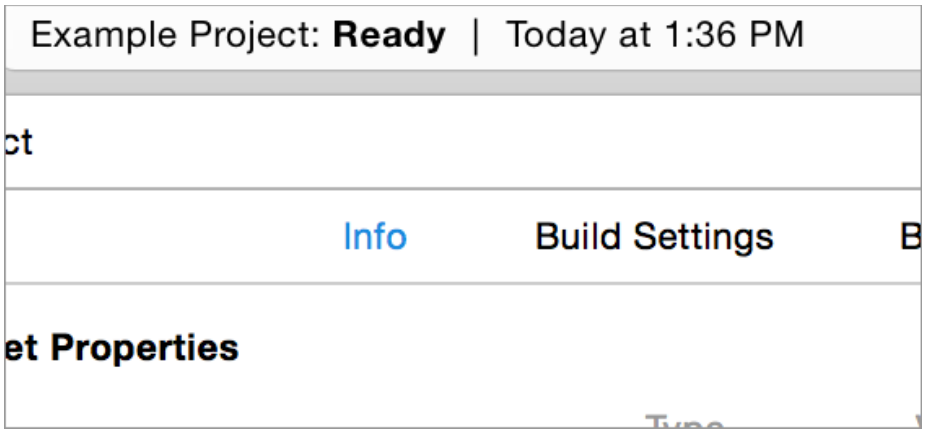
The plugin created as part of this tutorial is a fine example of this. It modifies one of the context menus of Textual even though the <`THOPluginProtocol`> protocol does not provide a way to do this.

Step #4: Prepare the Principal Class cont.

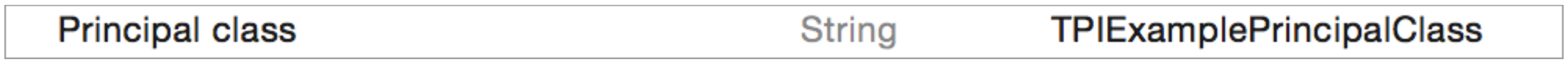
Move focus back to the project by selecting it in Xcode.



With the project selected, move focus to the **Info** tab of the project.


Example Project: **Ready** | Today at 1:36 PM

The **Info** tab will present a set of properties related to the active target. In the list of properties, find the property named **Principal class**. Set its value to the name of the principal class created earlier on in the tutorial.



Principal class	String	TPIExamplePrincipalClass

Step #5: Declaring the Target Version

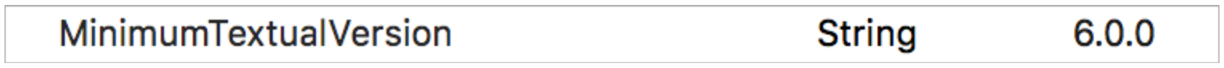
A plugin must declare the minimum version of Textual that it is compatible with.

Textual declares the constant named `THOPluginProtocolCompatibilityMinimumVersion`. This constant is compared against the minimum version that a plugin specifies. If the plugin's value is equal to or greater than this constant, then the plugin is considered safe to load.

Unlike the version information that visible to the end user, this constant does not change often. It only changes when modifications have been made to Textual’s codebase that may result in crashes when loading existing plugins.

For example, even though Textual’s visible version number is “5.0.4”, the value of this constant is “5.0.0”

After declaring the plugin’s principal class in the **Info** tab of Xcode, add a new entry with the key named: **MinimumTextualVersion** - Set the value of this entry to “6.0.0” as a **String**.



MinimumTextualVersion	String	6.0.0

Step #6: Writing Code

To keep this tutorial short, the code that will power the plugin being created will not be walked through line by line. However, it has been documented with comments which explain the logic behind what is happening.

Move focus to the implementation file of the principal class.

Paste the following source code below the `@implementation` declaration of the principal class.

```
- (void)pluginLoadedIntoMemory
{
      /* Ask the menu controller for the current user control menu. */
      /* menuController() is a macro which is designed to provide easier access to
       some of the most frequently accessed sections of Textual's codebase. Other
       defines include masterController(), worldController(), and mainWindow() */
      NSMenu *userControlMenu = menuController().userControlMenu;

      /* Create the new menu item. */
      NSMenuItem *newItem = [NSMenuItem new];

      [newItem setTitle:@"Post Link to Download Page"];
      [newItem setKeyEquivalent:@""]; // No keyboard shortcut
      [newItem setTarget:self]; // The class that hosts the method to invoke
      [newItem setAction:@selector(menuItemClicked:)]; // Invoke this method

      /* Add new item to the existing menu. */
      [userControlMenu addItem:[NSMenuItem separatorItem]];
      [userControlMenu addItem:[newItem copy]];
}
```

The method named `-pluginLoadedIntoMemory`, which encapsulates the source code seen above, is defined by the `<THOPluginProtocol>` protocol. This particular method is the first method invoked by Textual when a plugin is loaded. The plugin being created takes advantage of this to setup the new menu item.

Step #6: Writing Code cont.

Below the method `-pluginLoadedIntoMemory`, paste the following source code.

```
- (void)menuItemClicked:(id)sender
{
    /* The menu controller records which connection and channel were
     selected when the menu opened. If you do not want to target those,
     you can ask mainWindow() for the most up-to-date selection info. */
    IRCClient *client = menuController().selectedClient;

    IRCChannel *channel = menuController().selectedChannel;

    /* Only continue if both values are non-nil */
    if (client && channel) {
     /* Ask the menu controller for the list of users selected during the
      click event. It is important to send the "sender" parameter to the
      menu controller when asking for this information so that it can
      understand the context of the event and provide the correct return. */

     for (IRCChannelUser *user in [menuController() selectedMembers:sender]) {
         /* Build the message that will be posted. */
         NSString *message = [NSString stringWithFormat:@"%@, the Textual IRC Client
     can be downloaded from: www.textualapp.com", user.user.nickname];

         /* Send the message to the channel that was selected. */
         [client sendPrivmsg:message toChannel:channel];
     }

     /* Instruct the menu controller to forget about the event and deselect
      related users when it is completed. */
     [menuController() deselectMembers:sender]; // Always send "sender"
    }
}
```

The method named `-menuItemClicked:` is invoked when the new menu item is clicked.

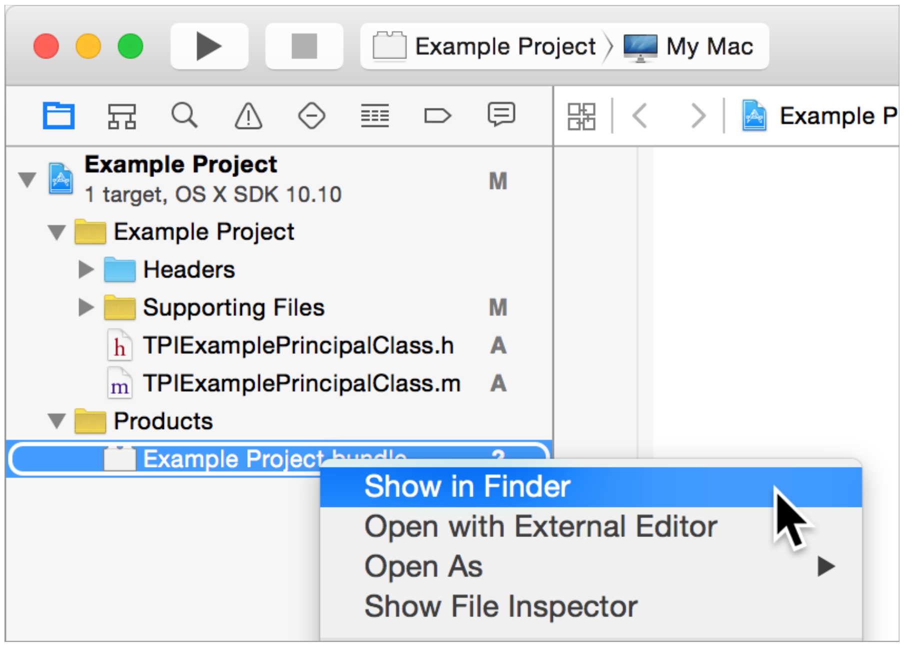
Step #7: Building & Testing

With all source code in place, attempt to build the project using the keyboard combination **Command B** (⌘B)

If assembled correctly, the project will build with **zero warnings and zero errors**.

In the file list on the left side of Xcode, expand the folder named **Products**.

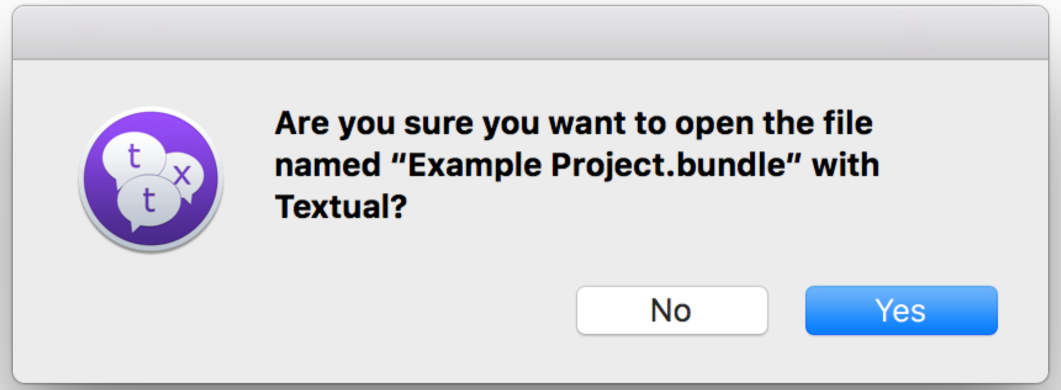
Inside the **Products** folder is the built plugin. Control click it and select the option to **Show In Finder**.



Step #7: Building & Testing cont.

Control click the file opened in Finder and select **Textual** as the application to open it with.

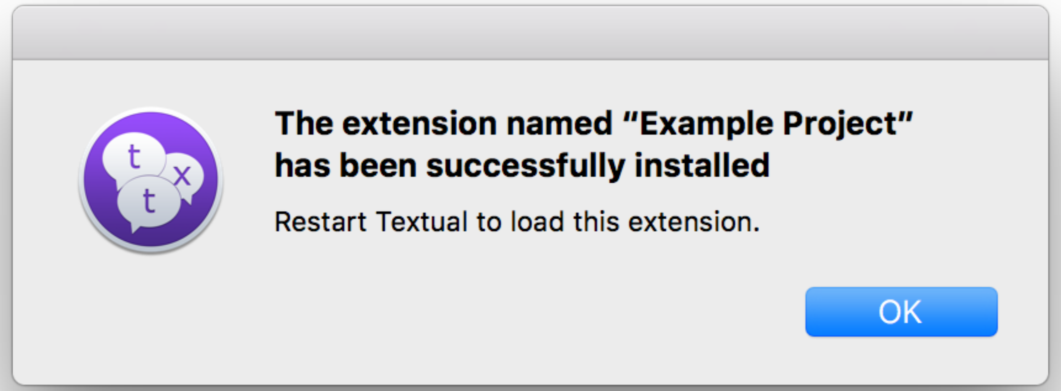
When opened, Textual will present a dialog confirming that the file was not opened accidentally.



Are you sure you want to open the file named "Example Project.bundle" with Textual?

Click the **Yes** button to continue.

After a second, Textual will present another dialog confirming a successful installation of the plugin.



The extension named "Example Project" has been successfully installed

Restart Textual to load this extension.

Close Textual using the keyboard combination **Command Q** (⌘Q), then relaunch it using Finder or Spotlight.

Step #7: Building & Testing cont.

Repeat these steps each time a copy of the plugin is finished building. That, or you can create a shell script that moves the plugin to the correct location when the build completes.

Mac App Store version:

```
~/Library/Group Containers/8482Q6EPL6.com.codeux.irc.textual/Library/Application Support/Textual/Extensions
```

Standalone version:

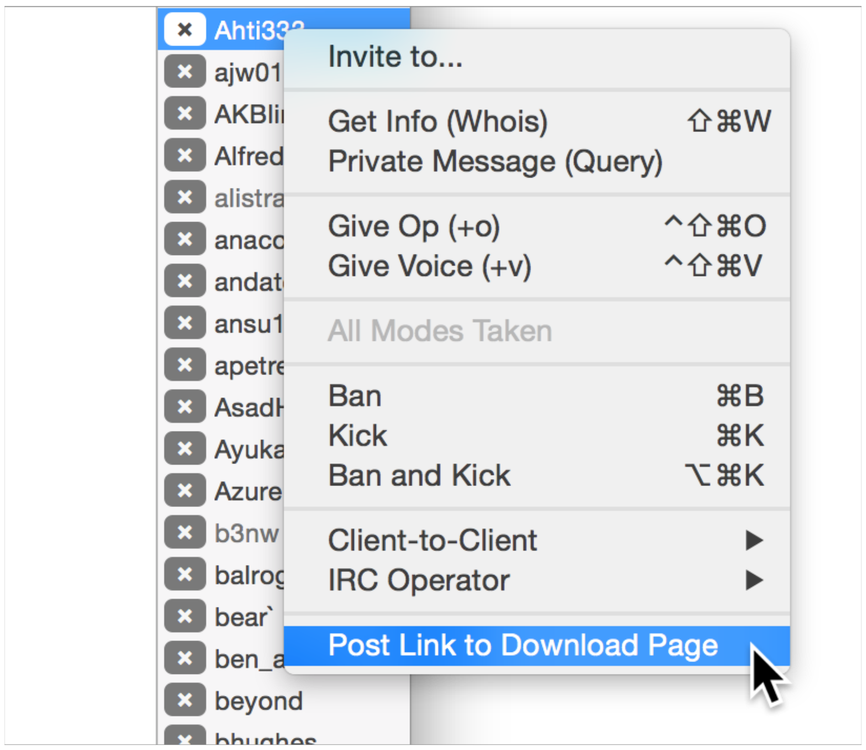
```
~/Library/Group Containers/com.codeux.apps.textual/Library/Application Support/Textual/Extensions
```

Step #7: Building & Testing cont.

With the plugin installed, join an IRC channel.

Once joined, select one or more users, then control click them to open the user control context menu.

The new menu item will appear at the bottom of the menu.


Ahti333
ajw01
AKBli
Alfred
alistra
anaco
andat
ansu1
apetre
AsadH
Ayuka
Azure
b3nw
balrog
bear`
ben_a
beyond
bhughes
Invite to...
Get Info (Whois) ⇧⌘W
Private Message (Query)
Give Op (+o) ^⇧⌘O
Give Voice (+v) ^⇧⌘V
All Modes Taken
Ban ⌘B
Kick ⌘K
Ban and Kick ⌥⌘K
Client-to-Client ▶
IRC Operator ▶
Post Link to Download Page