# Technical Note TN2322

# Building a Cocoa-AppleScript (AppleScriptObjC) Automator Action

This document explains how to use the Cocoa-AppleScript (AppleScriptObjC) Xcode template to create an Automator action.

Xcode includes a template that makes it easy to create Cocoa-AppleScript Automator actions. Building an action with this template consists of the following key steps:

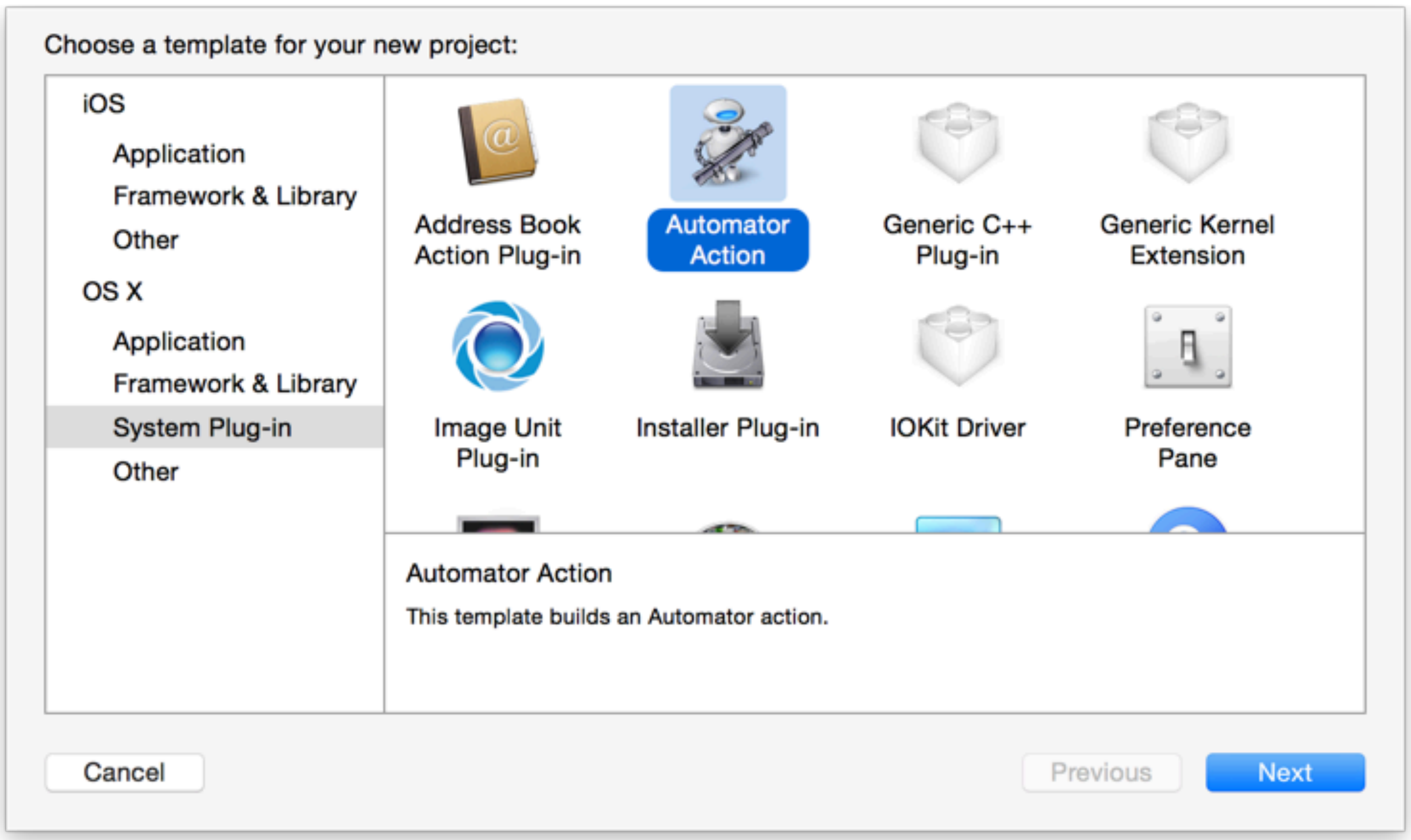
- Build an Xcode project
- Configure action attributes and behavior
- Create an interface for your action
- Add code to your action
- Test the action with Automator
- Build the action
- Install the action

**Note:** NOTE: Cocoa-AppleScript refers to AppleScriptObjC, which replaced AppleScript Studio in Xcode, effective OS X v10.6. For information on AppleScriptObjC, see the AppleScriptObjC Release Notes.

## To create a Cocoa-AppleScript Automator action Xcode project:

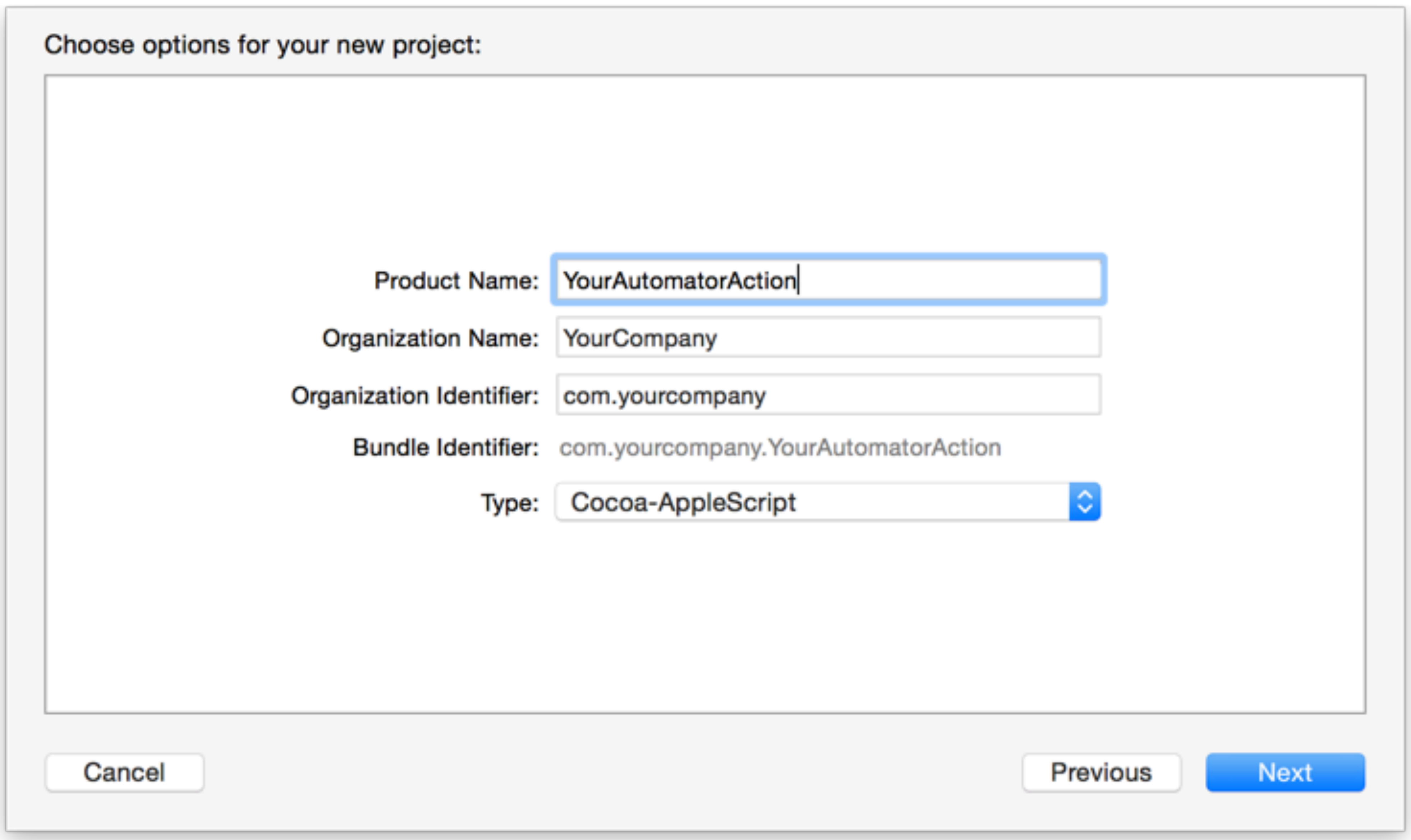
- Launch Xcode.
- Choose File > New > Project from the menu bar, or press Command-Shift-N.
- In the Xcode template selection dialog, click System Plug-in > Automator Action, and then, click Next.

**Figure 1:**



- Enter a name for your Automator action, your organization name, and identifier, and then select Cocoa-AppleScript from the Type pop-up menu.

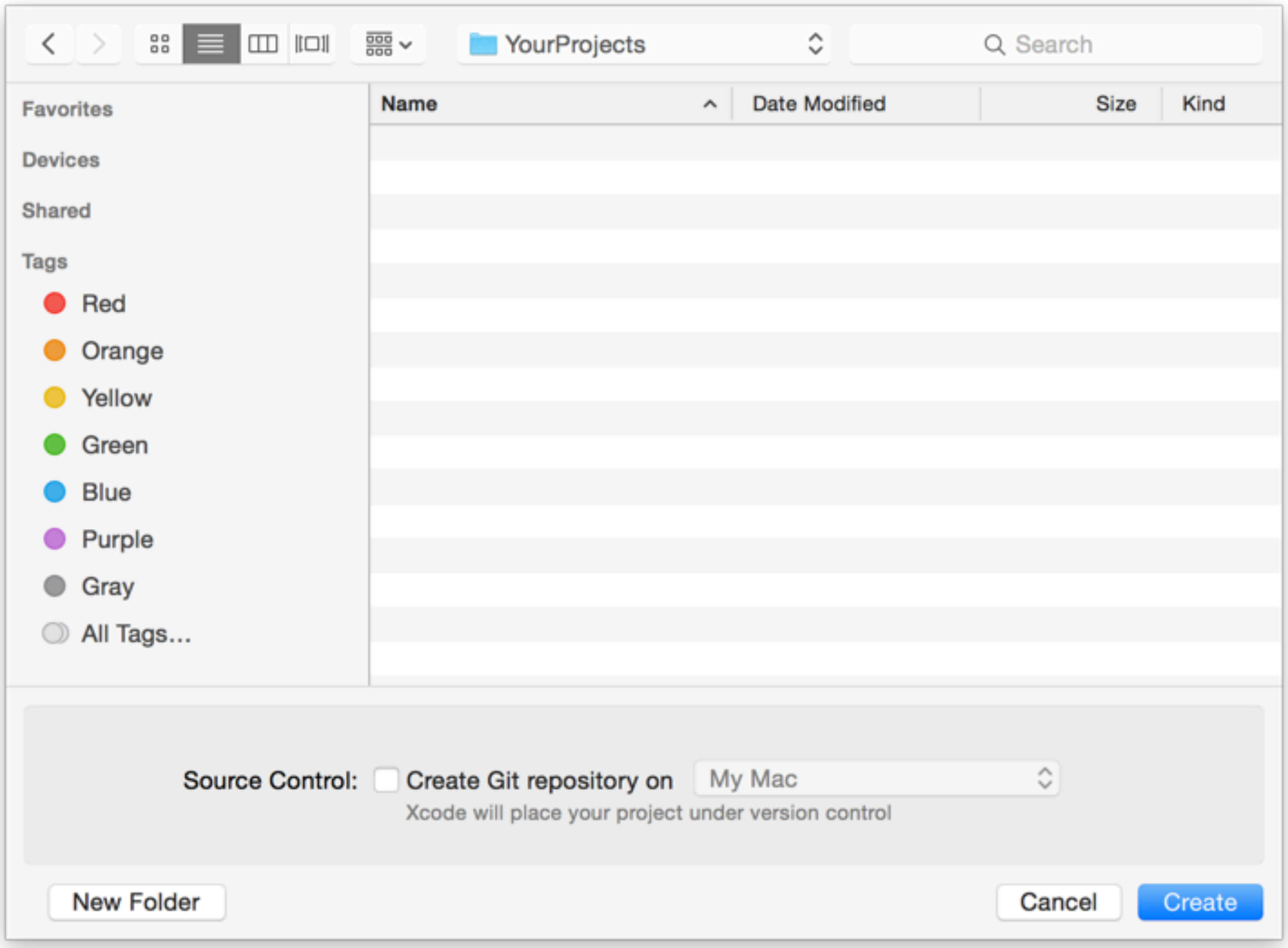
**Figure 2:**


Choose options for your new project:

Product Name: YourAutomatorAction
Organization Name: YourCompany
Organization Identifier: com.yourcompany
Bundle Identifier: com.yourcompany.YourAutomatorAction
Type: Cocoa-AppleScript

- Choose a location for the action project, and click Create.

**Figure 3:**



Source Control: Create Git repository on My Mac
Xcode will place your project under version control

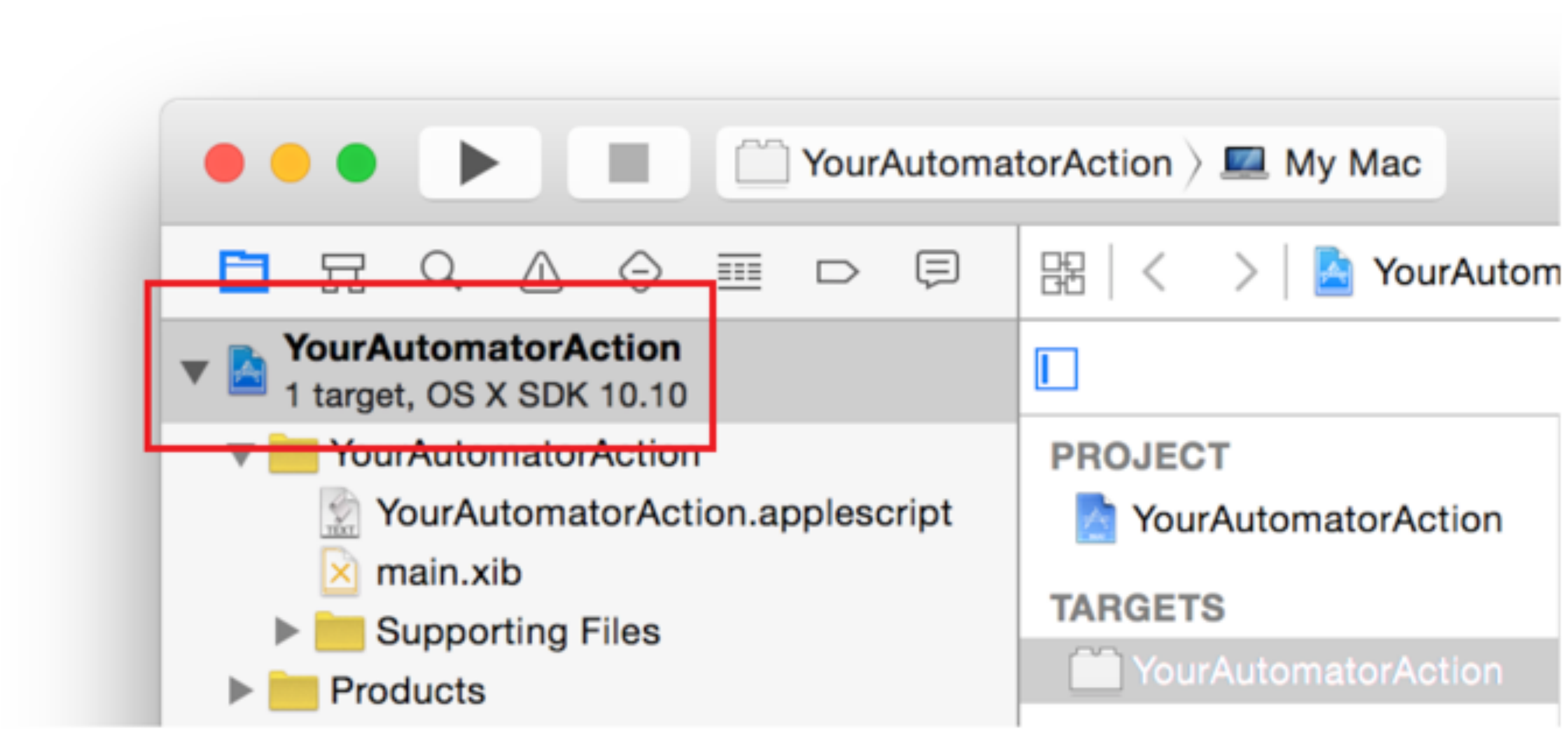
Your project is saved where you specified, and Xcode opens it in a new window. The next step is to configure attributes for your action and its behavior.

Back to Top

## To configure your action’s attributes and behavior:

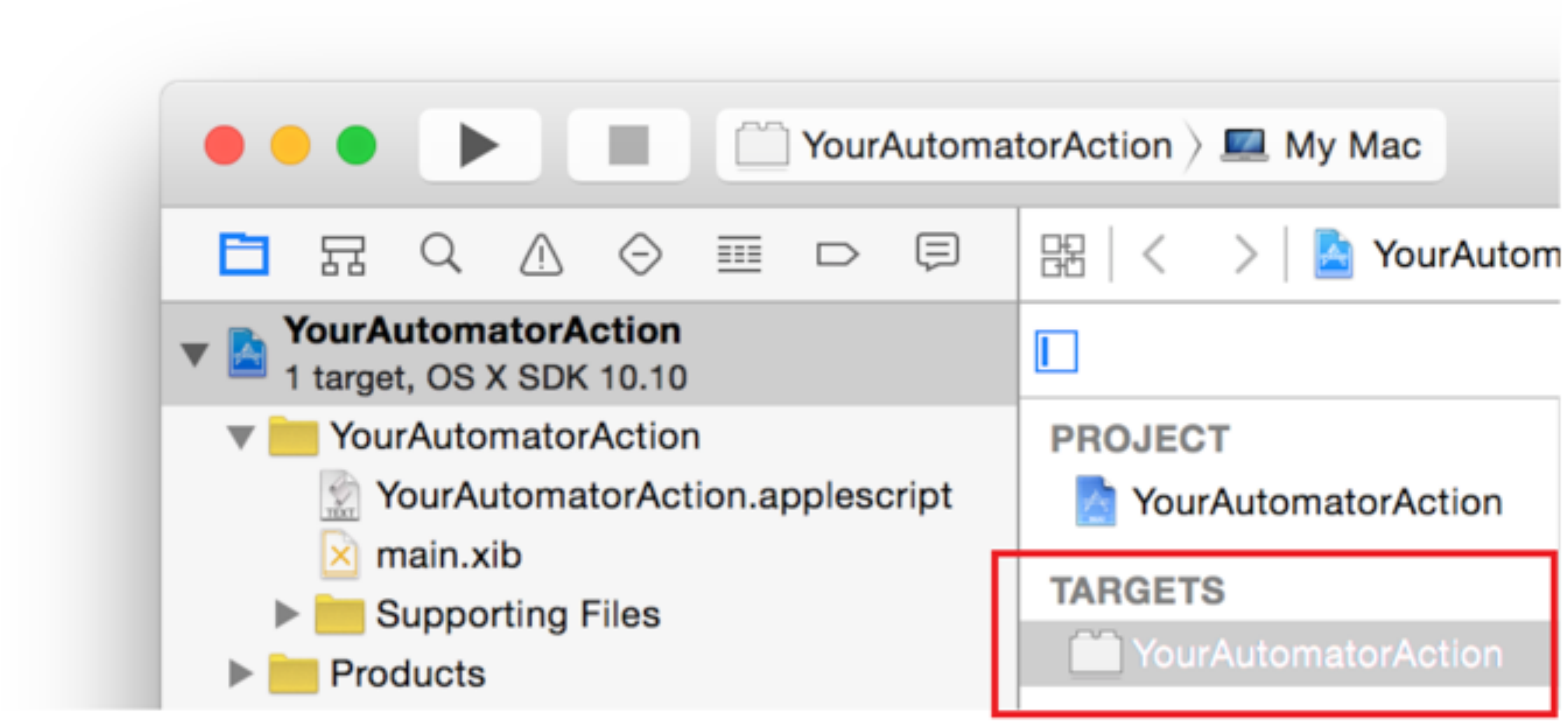
- Click on your action project in the Xcode project navigator.

**Figure 4:**



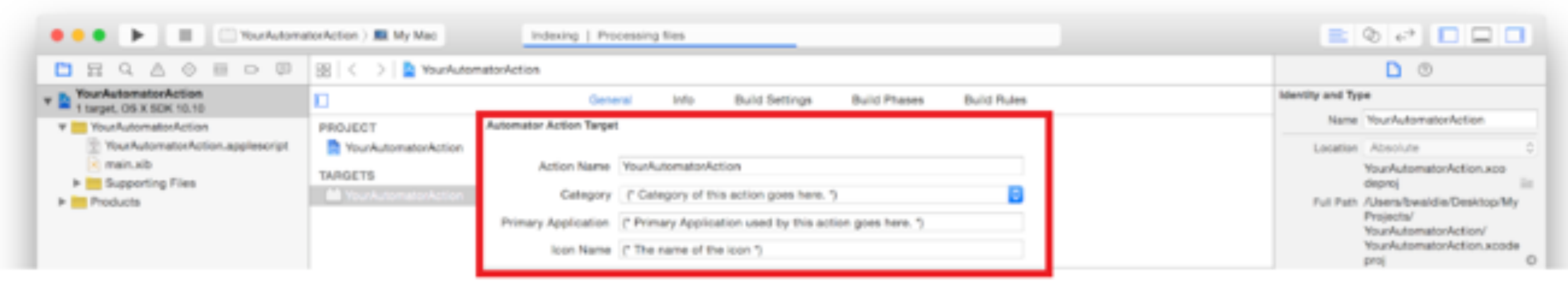
- In the editor area, select the target for your project.

**Figure 5:**



- Select a category, and enter the application your action will target. These settings control how your action is organized and found by users within the Automator action library.

**Figure 6:**

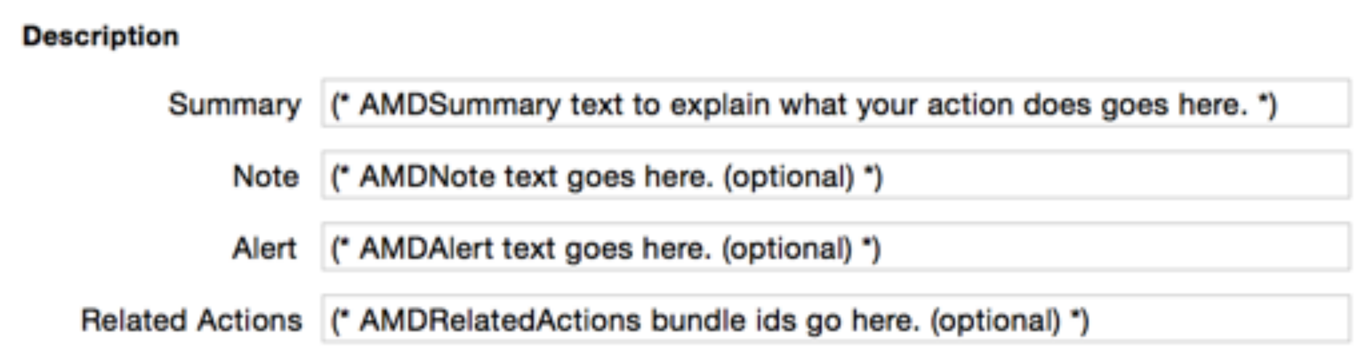


Automator Action Target

| | |
|---|---|
| Action Name | YourAutomatorAction |
| Category | (* Category of this action goes here. *) |
| Primary Application | (* Primary Application used by this action goes here. *) |
| Icon Name | (* The name of the icon *) |

- In the Description area, enter a summary (required) for your action, and fill in any other optional fields you may need. This content will appear in the description area when the action is selected in the Automator action library. Be detailed, because end users will refer to this information in order to determine what your action does and how to configure it.

**Figure 7:**

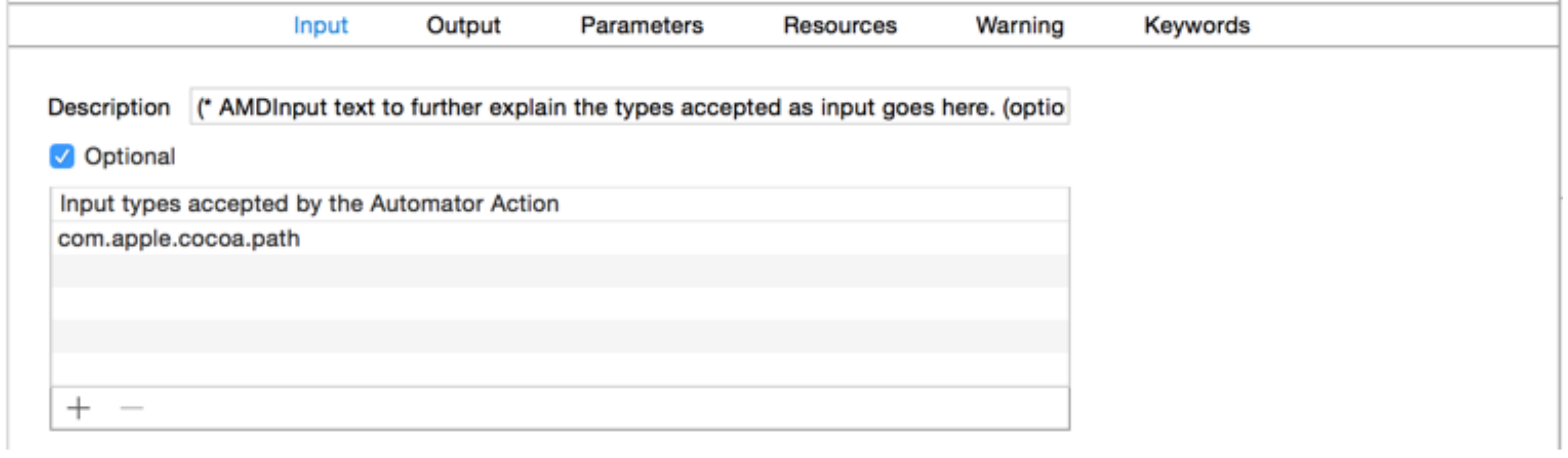


**Description**

| | |
|---|---|
| Summary | (* AMDSummary text to explain what your action does goes here. *) |
| Note | (* AMDNote text goes here. (optional) *) |
| Alert | (* AMDAlert text goes here. (optional) *) |
| Related Actions | (* AMDRelatedActions bundle ids go here. (optional) *) |

- Click Input, and specify the type of input your action will accept. If input is not required, select the Optional checkbox. If helpful to the user, consider expanding on the action’s input type by providing a detailed description.

**Figure 8:**


Input Output Parameters Resources Warning Keywords

Description (* AMDInput text to further explain the types accepted as input goes here. (optio

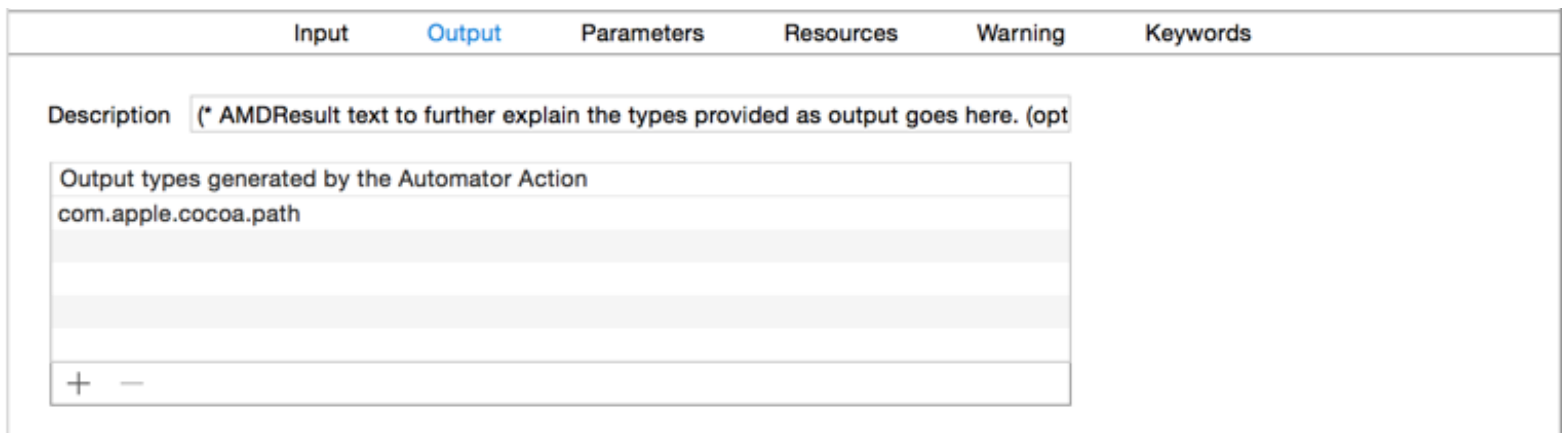
✓ Optional

| Input types accepted by the Automator Action |
|---|
| com.apple.cocoa.path |

**Note:** For information on the types of input an action can accept and the types of output an action can provide, see Type Identifiers in the Automator Programming Guide.

- Click Output, and specify the type of output your action will provide. Optionally, specify a more detailed description of the output.

**Figure 9:**

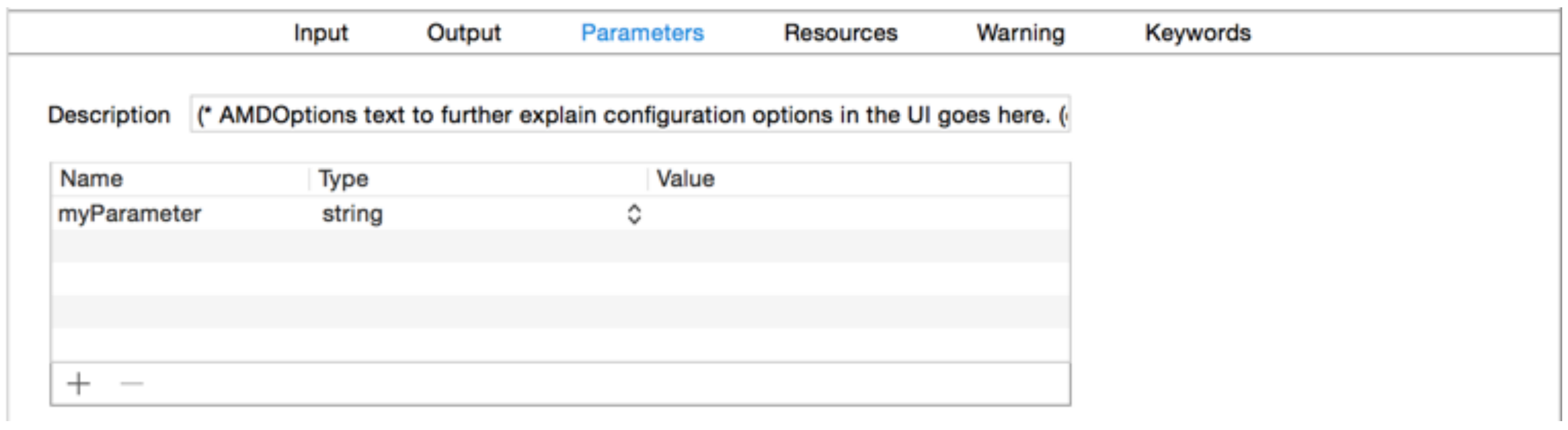

Input Output Parameters Resources Warning Keywords

Description (* AMDResult text to further explain the types provided as output goes here. (opt

| Output types generated by the Automator Action |
|---|
| com.apple.cocoa.path |

- Click Parameters, and add any parameters for your action. Parameters are configuration values that are typically bound to elements in your action's interface. For example, a parameter may be a value for a text field, the selected state of a checkbox, or the enabled state of a button. Parameters enable your action to display a certain state when added to a workflow, and provide information back to your action's code about any configuration changes the user may have made.

**Figure 10:**

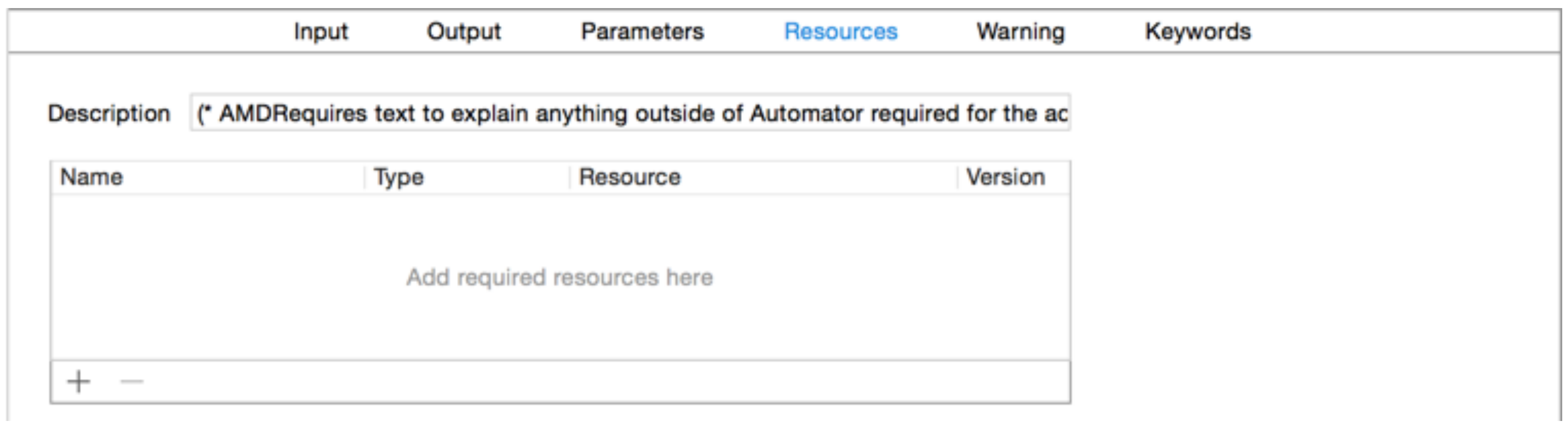

Input Output Parameters Resources Warning Keywords

Description (* AMDOptions text to further explain configuration options in the UI goes here. (

| Name | Type | Value |
|---|---|---|
| myParameter | string | |

- If your action requires any resources, such as specific app versions or files, click Resources and add them.

**Figure 11:**


Input Output Parameters Resources Warning Keywords

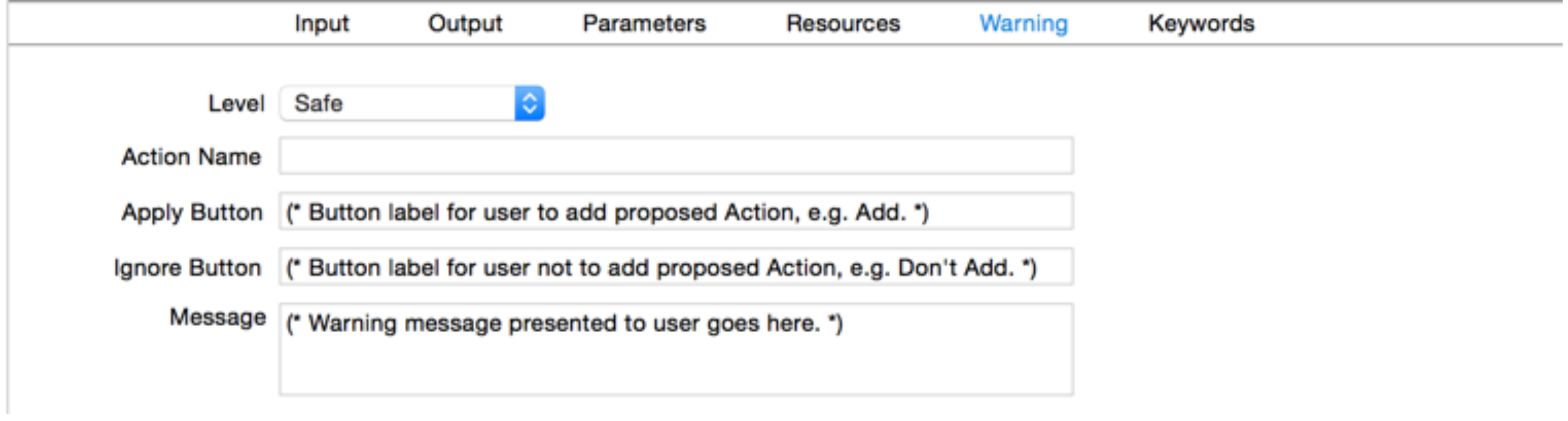
Description (* AMDRequires text to explain anything outside of Automator required for the ac

| Name | Type | Resource | Version |
|---|---|---|---|

Add required resources here

- If your action requires a warning message when added to the workflow, click Warning and provide the necessary details. Note that any action that will modify the user's data in a manner which cannot be undone should include a warning. For these types of actions, set the warning Level pop-up menu to Irreversible.

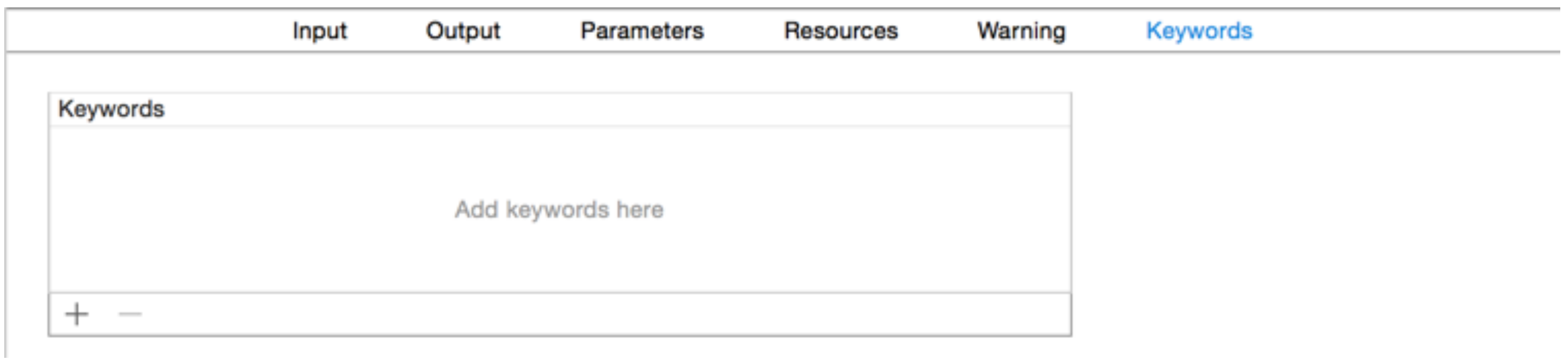
**Figure 12:**


Input Output Parameters Resources Warning Keywords

| | |
|---|---|
| Level | Safe |
| Action Name | |
| Apply Button | (* Button label for user to add proposed Action, e.g. Add. *) |
| Ignore Button | (* Button label for user not to add proposed Action, e.g. Don't Add. *) |
| Message | (* Warning message presented to user goes here. *) |

- Click Keywords, and add keywords that may pertain to your action. Keywords are queried when a user searches for an action, so it's important to provide variations that a user might enter. For example, if your action processes Address Book records, you might specify the following keywords: card, contacts, people, person, and vcard. Providing a range of good keywords will make your action more findable.

**Figure 13:**



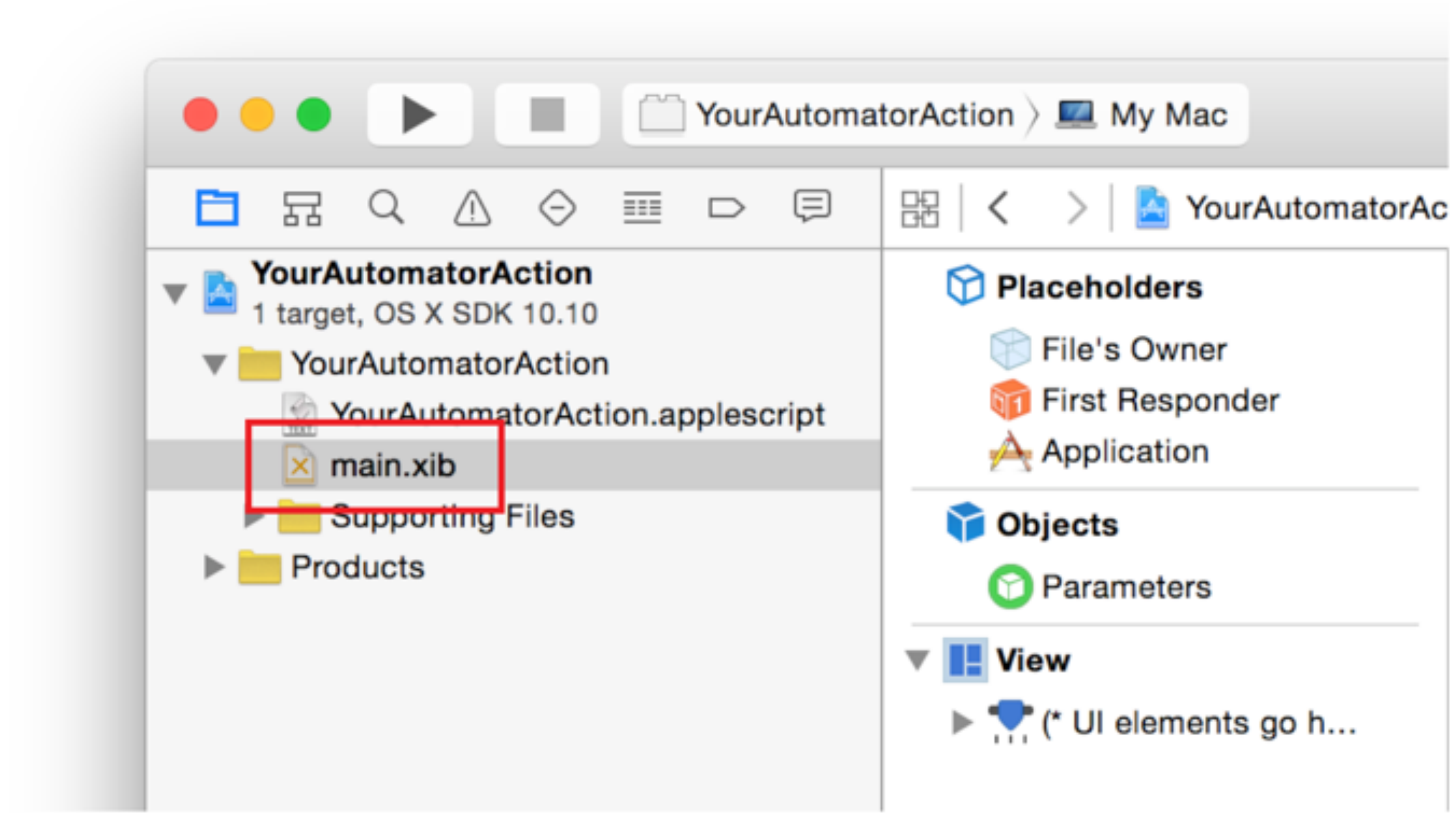
Not all Automator actions require an interface. Some actions simply receive input, process it, and produce a result. For example, the Eject Disk action that's included with Automator doesn't have any configurable settings. It just receives one or more disks as input, ejects them, and moves on to the next action in the workflow. Because there's nothing to configure, this action doesn't have an interface.

Back to Top

## To create an interface for your action:

- Click main.xib in the Xcode Project Navigator.

**Figure 14:**



- Click the default view, which is included in the Cocoa-AppleScript template.

**Figure 15:**

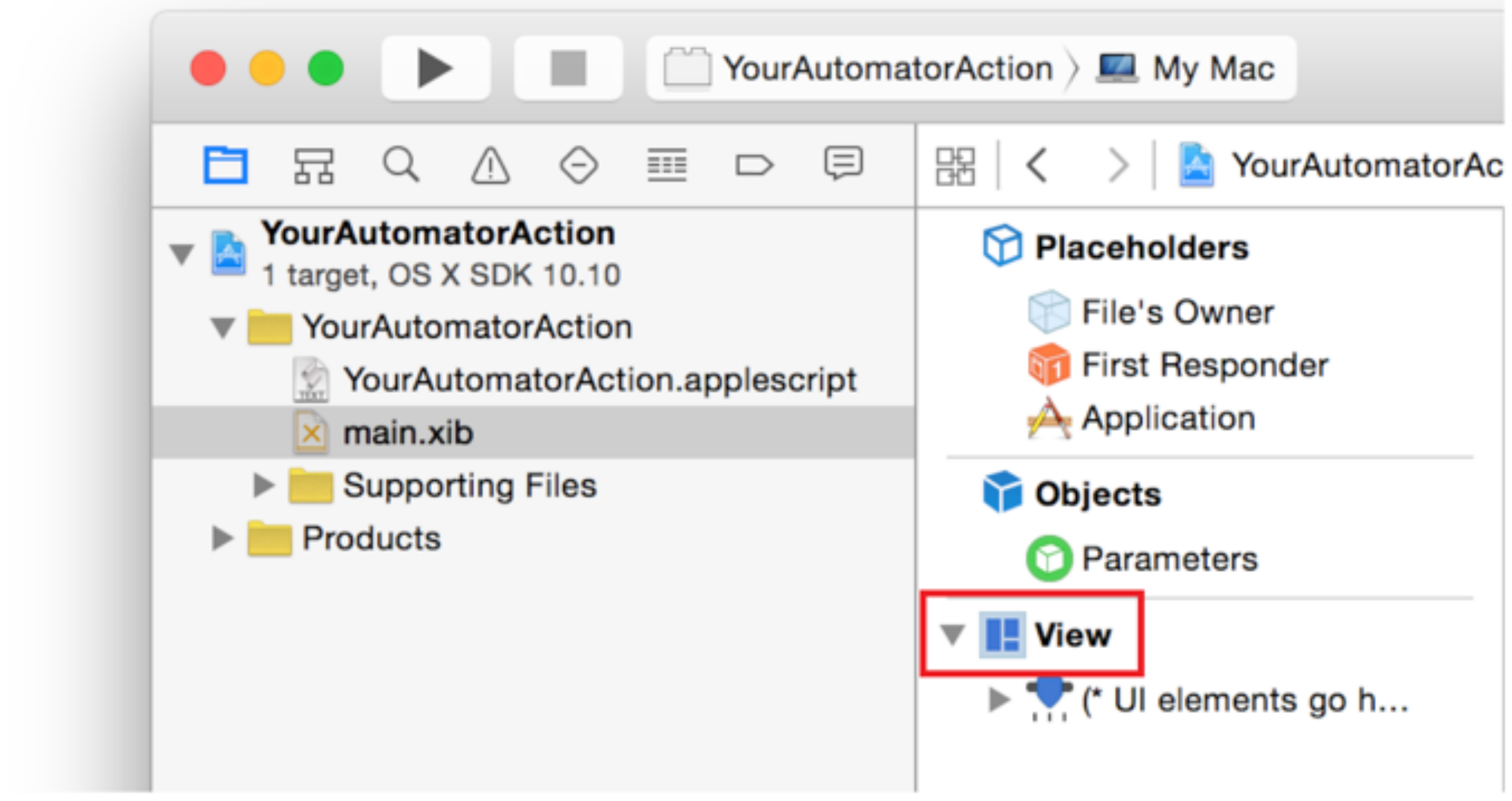
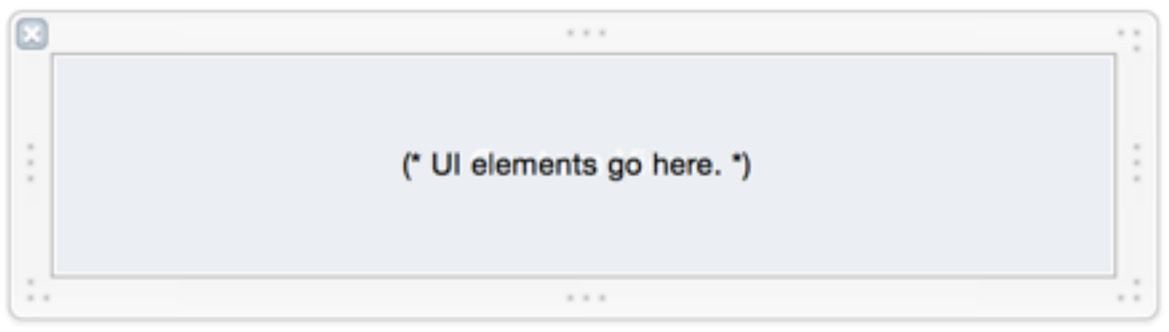
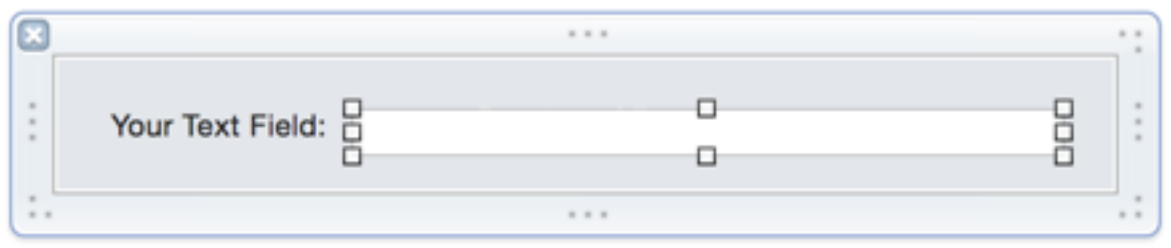


**Figure 16:**



- Add the desired interface elements to the view.

**Figure 17:**



When adding interface elements to an Automator action view, adhere to the following design guidelines:

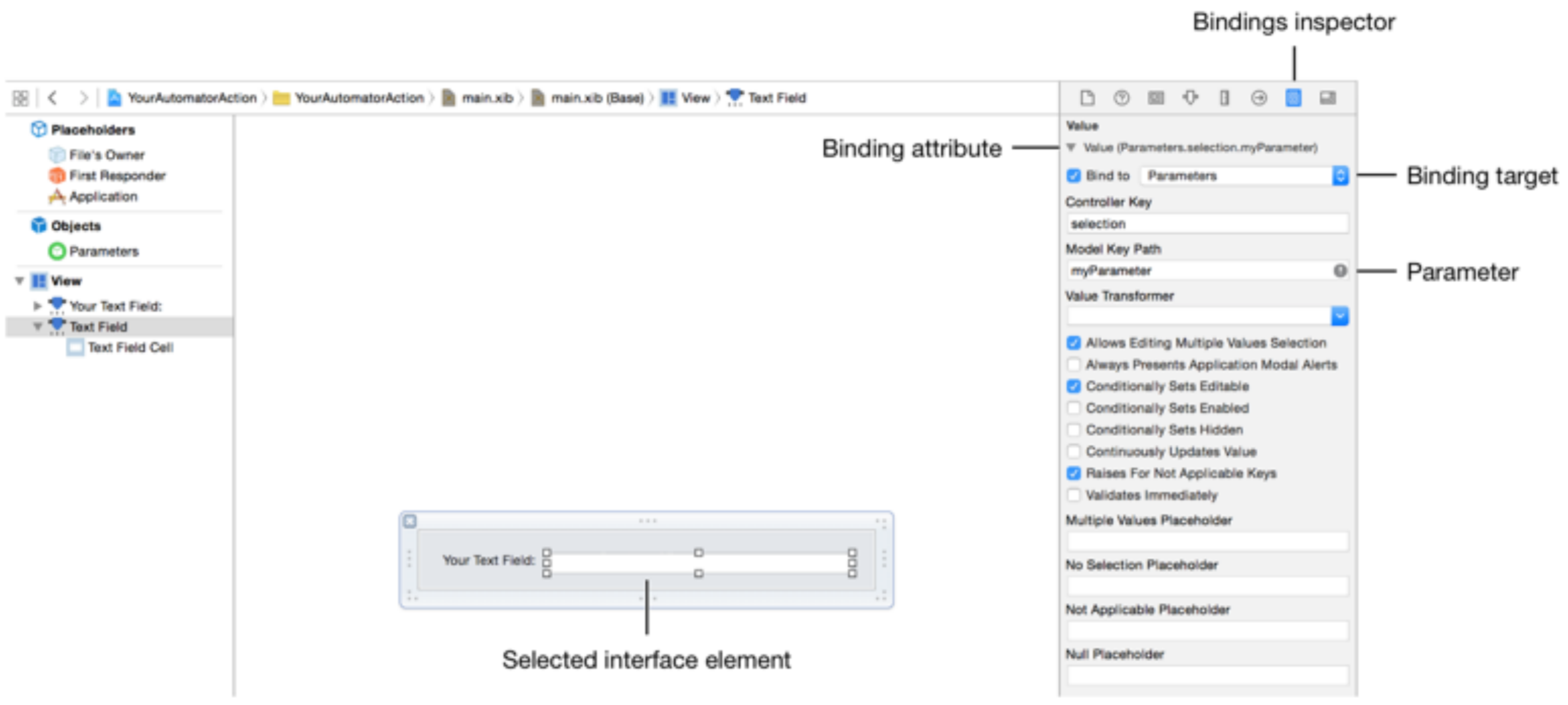
- Allow a 10-point margin between the edge of the action view and any interface elements.
- Don’t use labels to repeat information that already exists in the action’s name or description.
- Minimize the use of vertical space. For example, consider using pop-up menus instead of radio buttons.
- Use small-size interface elements.
- Use progress indicators to indicate when an interface element is busy loading its content.
- Follow the OS X Human Interface Guidelines.

Back to Top

## To map a parameter to an interface element attribute:

- Select the interface element in the main.xib view.

**Figure 18:**



Value
Value (Parameters.selection.myParameter)
[✓] Bind to: Parameters
Controller Key: selection
Model Key Path: myParameter
Value Transformer:
[✓] Allows Editing Multiple Values Selection
[ ] Always Presents Application Modal Alerts
[✓] Conditionally Sets Editable
[ ] Conditionally Sets Enabled
[ ] Conditionally Sets Hidden
[ ] Continuously Updates Value
[✓] Raises For Not Applicable Keys
[ ] Validates Immediately
Multiple Values Placeholder:
No Selection Placeholder:
Not Applicable Placeholder:
Null Placeholder:

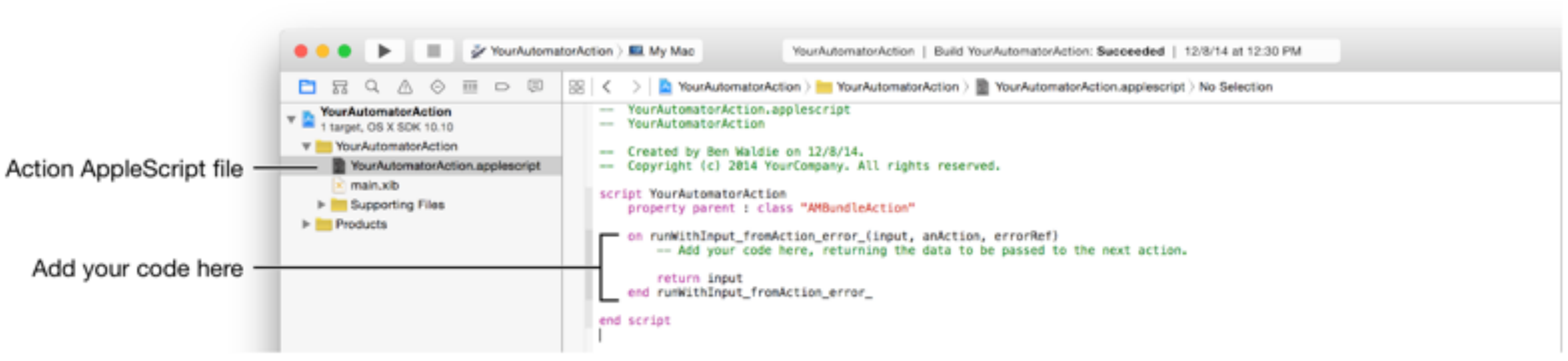
- Click the Bindings inspector in the Utilities pane, or press Command-Option-7.
- Locate the desired binding attribute, and click its disclosure triangle to expand it.
- Set the binding target to bind the attribute to Parameters.
- Enter the parameter name, which you specified when you configured your action's attributes and behavior, into the Model Key Path field.

Back to Top

## To add code to your action:

- Click yourprojectname.applescript in the Xcode project navigator.

**Figure 19:**


Action AppleScript file
Add your code here
YourAutomatorAction › My Mac
YourAutomatorAction | Build YourAutomatorAction: Succeeded | 12/8/14 at 12:30 PM
YourAutomatorAction › YourAutomatorAction › YourAutomatorAction.applescript › No Selection
YourAutomatorAction
1 target, OS X SDK 10.10
YourAutomatorAction
YourAutomatorAction.applescript
main.xib
Supporting Files
Products

```
--  YourAutomatorAction.applescript
--  YourAutomatorAction

--  Created by Ben Waldie on 12/8/14.
--  Copyright (c) 2014 YourCompany. All rights reserved.

script YourAutomatorAction
    property parent : class "AMBundleAction"

    on runWithInput_fromAction_error_(input, anAction, errorRef)
        -- Add your code here, returning the data to be passed to the next action.

        return input
    end runWithInput_fromAction_error_

end script
```

- Enter the desired processing code into the runWithInput_fromAction_error_ handler.

Back to Top

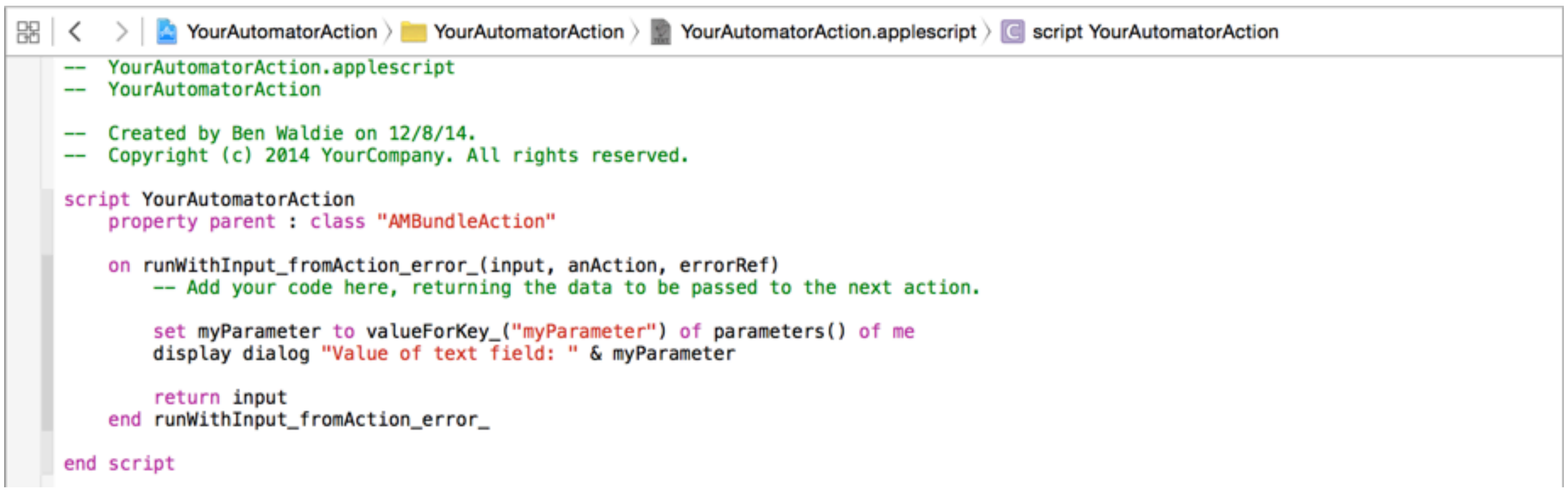
## To retrieve a parameter value in your AppleScript code:

Call valueForKey for the parameters method of the action, passing it the name of the parameter you wish to retrieve. For example:

**Listing 1:**

```
valueForKey_("myParameter") of parameters() of me
```

**Figure 20:**


YourAutomatorAction › YourAutomatorAction › YourAutomatorAction.applescript › script YourAutomatorAction

```
--  YourAutomatorAction.applescript
--  YourAutomatorAction

--  Created by Ben Waldie on 12/8/14.
--  Copyright (c) 2014 YourCompany. All rights reserved.

script YourAutomatorAction
    property parent : class "AMBundleAction"

    on runWithInput_fromAction_error_(input, anAction, errorRef)
        -- Add your code here, returning the data to be passed to the next action.

        set myParameter to valueForKey_("myParameter") of parameters() of me
        display dialog "Value of text field: " & myParameter

        return input
    end runWithInput_fromAction_error_

end script
```

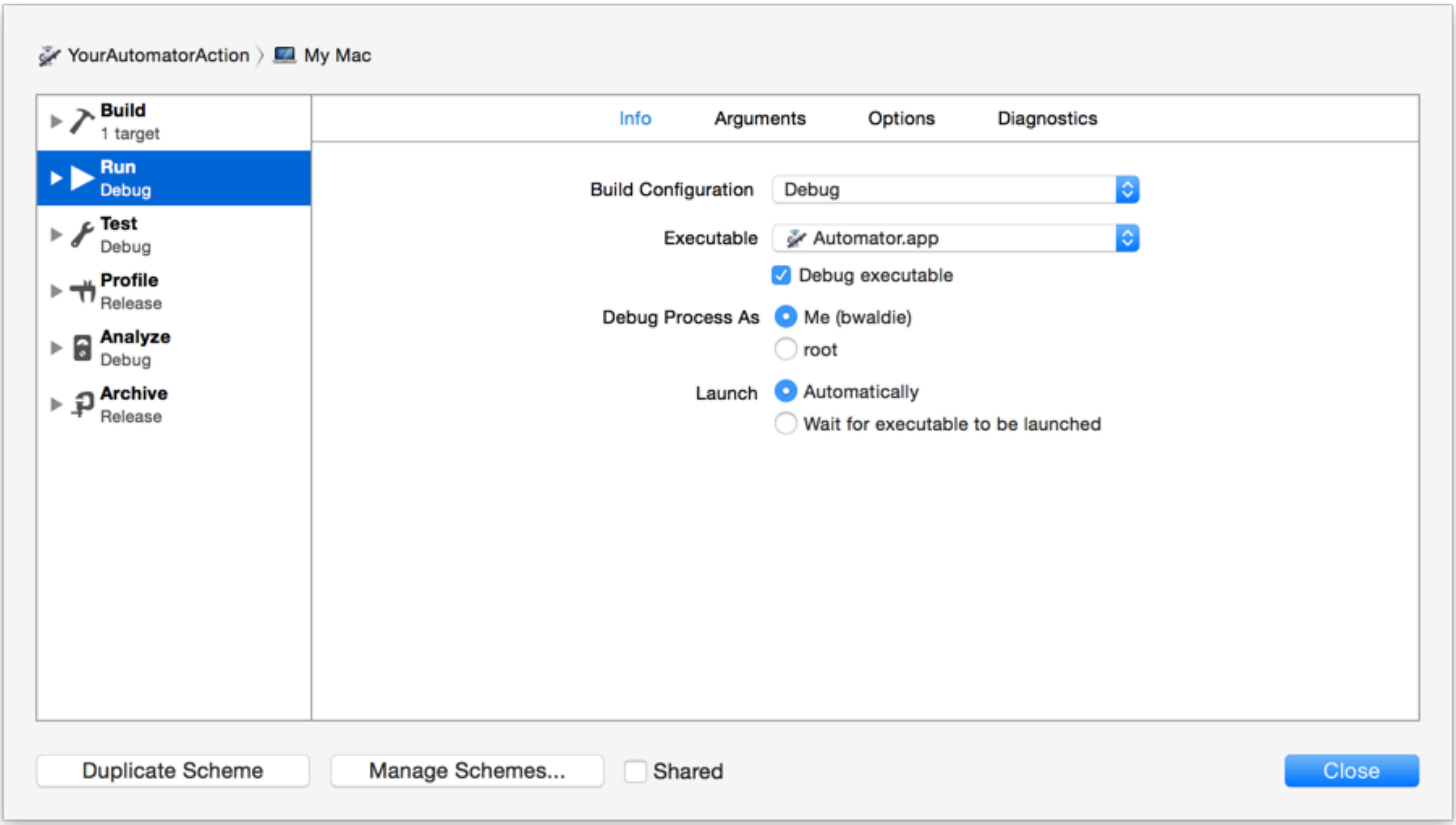
In order to test an Automator action Xcode project, you need to do two things. First, you need to set Automator as the run executable for the project. Second, you need to provide an argument that tells Automator where to find your action.

Back to Top

## To test your action with Automator:

- Choose Product > Scheme > Edit Scheme to open the scheme editor dialog.

**Figure 21:**


YourAutomatorAction 〉 My Mac

Build — 1 target
Run — Debug
Test — Debug
Profile — Release
Analyze — Debug
Archive — Release

Info | Arguments | Options | Diagnostics

Build Configuration: Debug
Executable: Automator.app
☑ Debug executable
Debug Process As: ● Me (bwaldie) ○ root
Launch: ● Automatically ○ Wait for executable to be launched

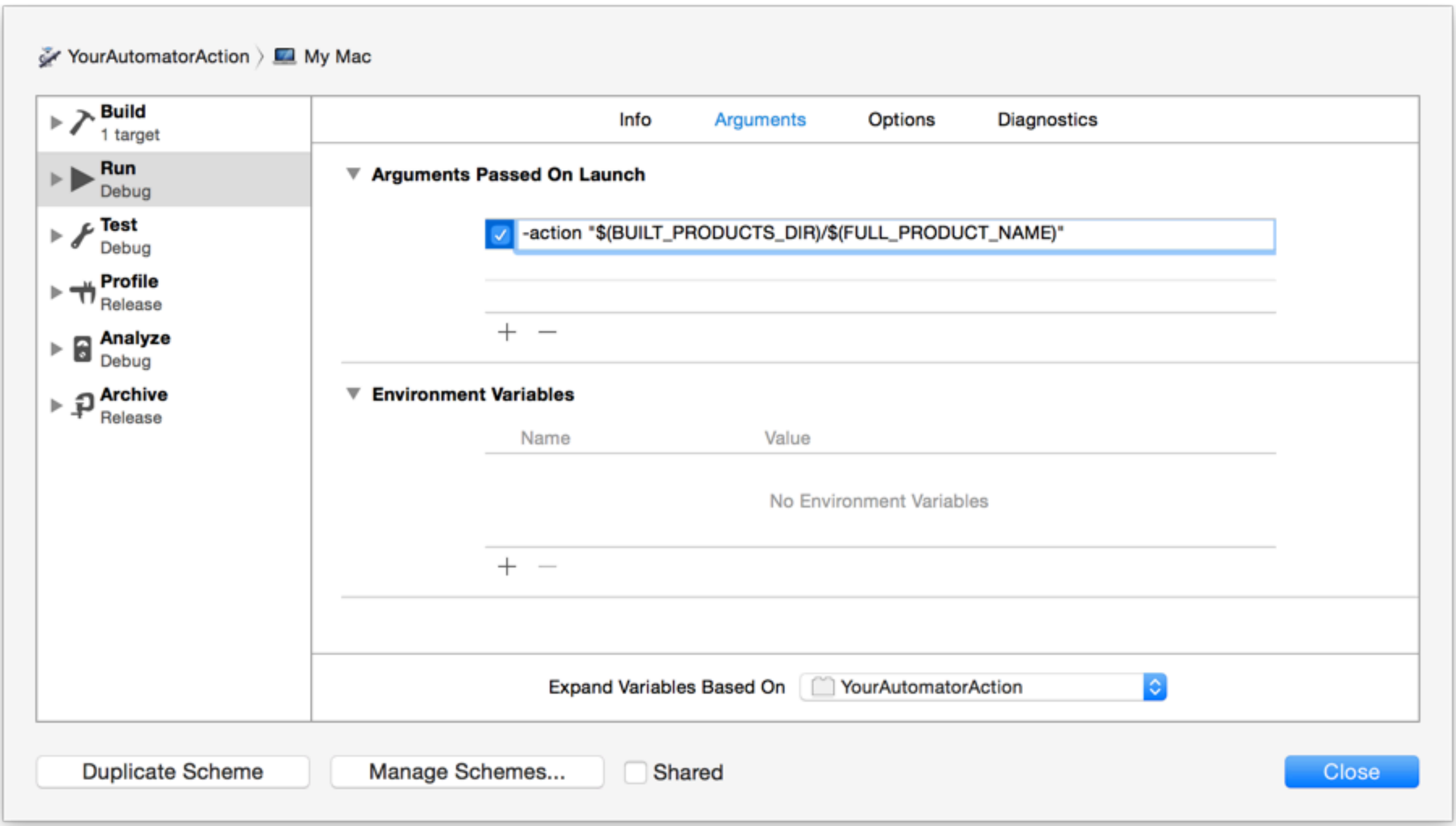
Duplicate Scheme | Manage Schemes... | ☐ Shared | Close

- In the list along the left side of scheme editor dialog, click Run.
- Set the Executable pop-up menu to Automator. Choose Other from the pop-up menu, and navigate to Automator in `/Applications`.
- Click Arguments.
- In the Arguments Passed On Launch area, click the Add button (+) to add a new argument. Set the argument to:

**Listing 2:**

```
-action "$(BUILT_PRODUCTS_DIR)/$(FULL_PRODUCT_NAME)"
```

**Figure 22:**


YourAutomatorAction 〉 My Mac

Build — 1 target
Run — Debug
Test — Debug
Profile — Release
Analyze — Debug
Archive — Release

Info | Arguments | Options | Diagnostics

Arguments Passed On Launch
☑ -action "$(BUILT_PRODUCTS_DIR)/$(FULL_PRODUCT_NAME)"

Environment Variables
Name | Value
No Environment Variables

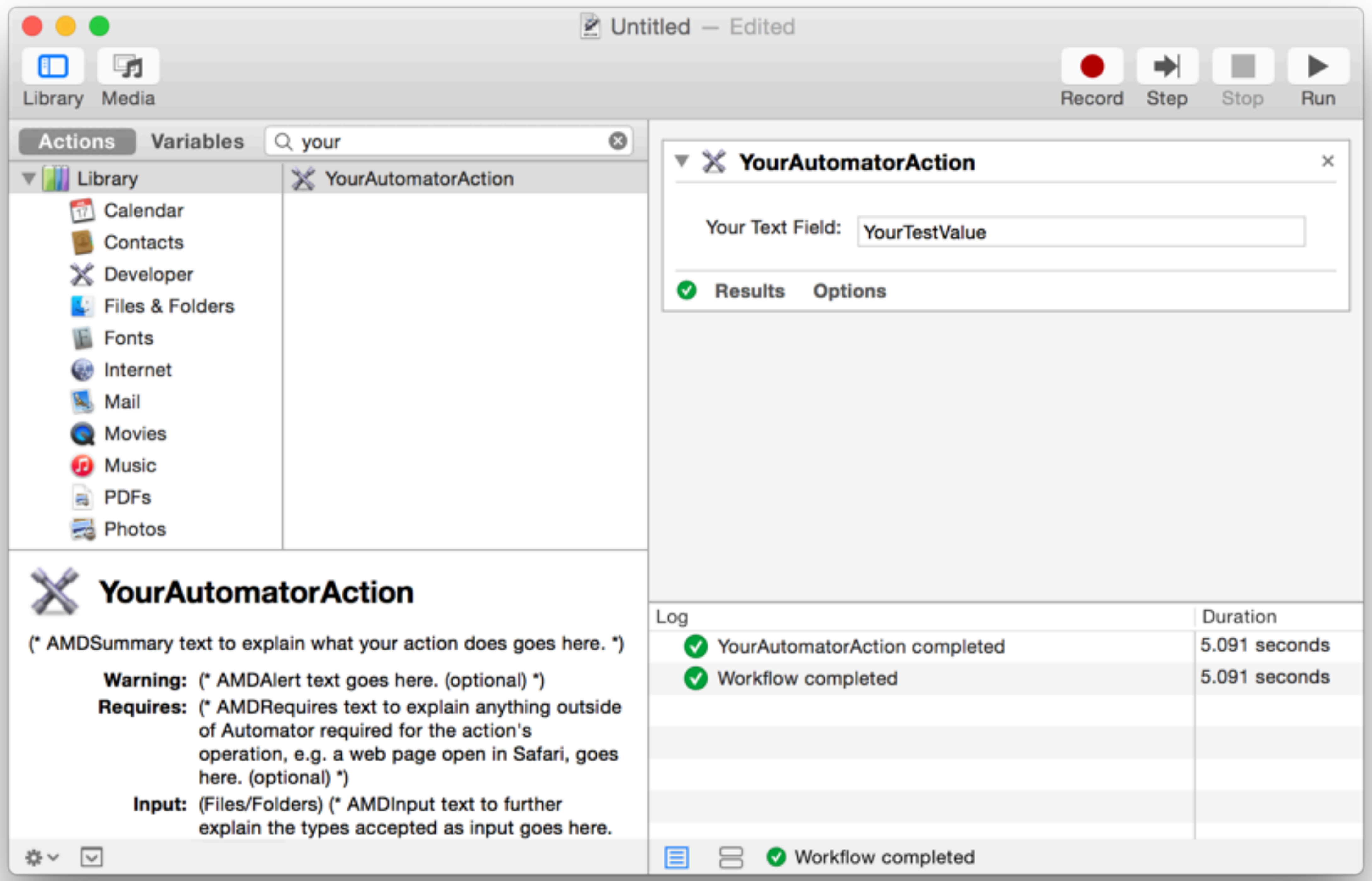
Expand Variables Based On: YourAutomatorAction

Duplicate Scheme | Manage Schemes... | ☐ Shared | Close

- Click Close to close the scheme editor dialog.

After configuring your Xcode project as described above, choose Product > Run or enter Command-R. An instance of Automator should launch and load your action. You can search the Automator action library to find it. Now, you can test your action and return to Xcode when you're done to stop testing and resume development.

**Figure 23:**



**YourAutomatorAction**

Your Text Field: YourTestValue

Results Options

**YourAutomatorAction**

(* AMDSummary text to explain what your action does goes here. *)

**Warning:** (* AMDAlert text goes here. (optional) *)

**Requires:** (* AMDRequires text to explain anything outside of Automator required for the action's operation, e.g. a web page open in Safari, goes here. (optional) *)

**Input:** (Files/Folders) (* AMDInput text to further explain the types accepted as input goes here.

| Log | Duration |
|---|---|
| YourAutomatorAction completed | 5.091 seconds |
| Workflow completed | 5.091 seconds |

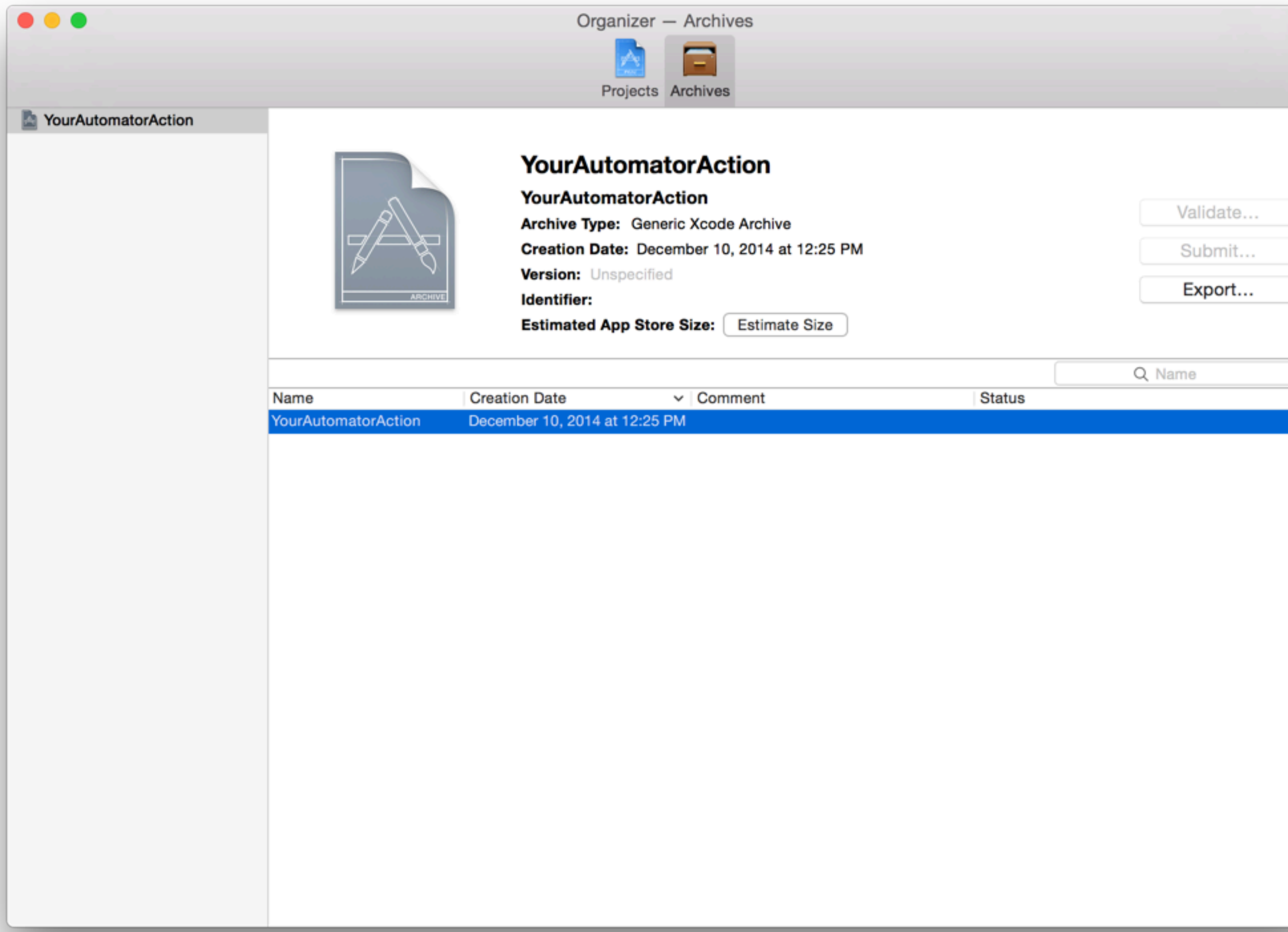
Workflow completed

Back to Top

## To build and install your action:

- Choose Product > Archive to display the archives pane of the organizer window.

**Figure 24:**



YourAutomatorAction

**YourAutomatorAction**

**YourAutomatorAction**

**Archive Type:** Generic Xcode Archive

**Creation Date:** December 10, 2014 at 12:25 PM

**Version:** Unspecified

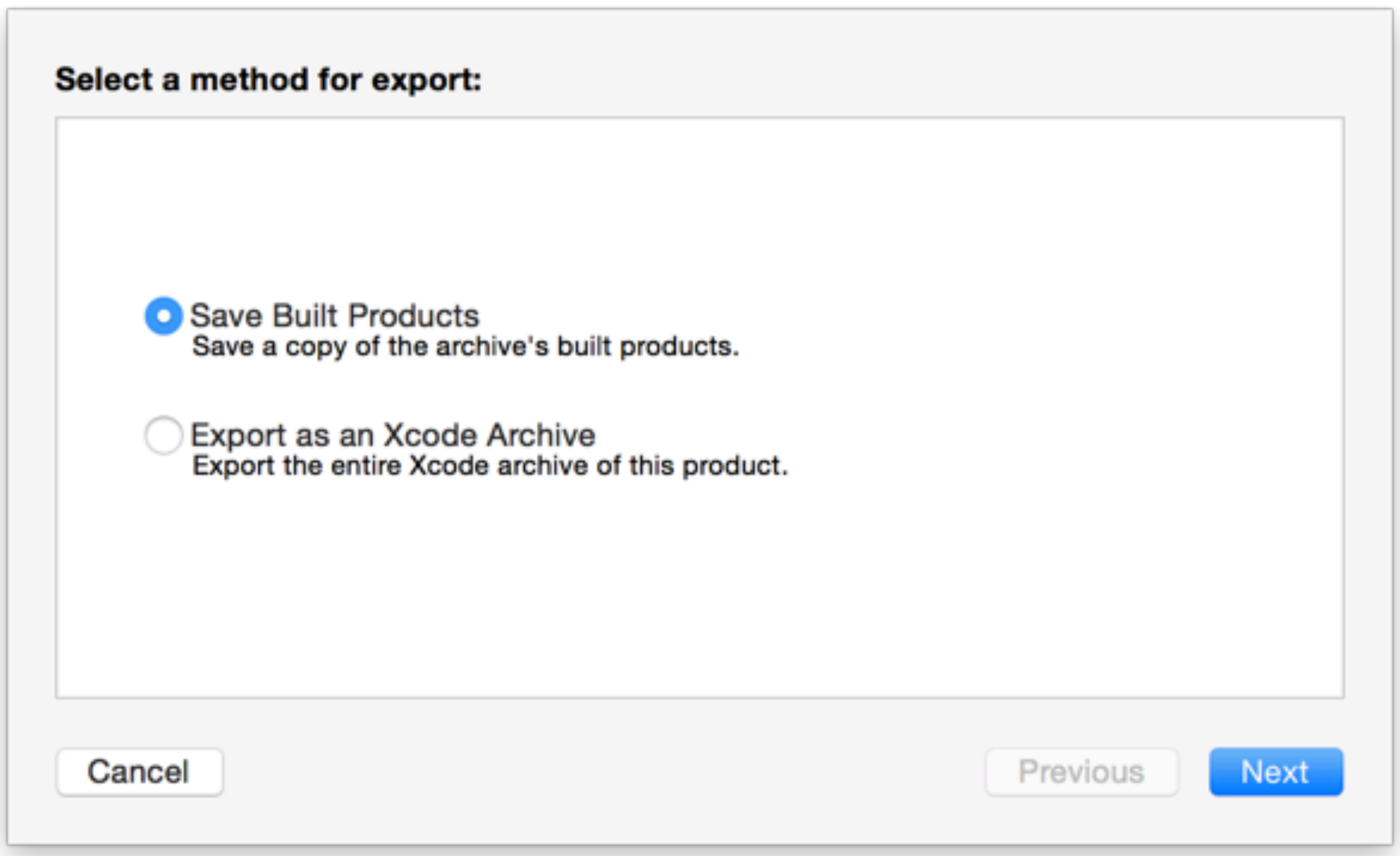
**Identifier:**

**Estimated App Store Size:** Estimate Size

| Name | Creation Date | Comment | Status |
|---|---|---|---|
| YourAutomatorAction | December 10, 2014 at 12:25 PM | | |

- Select the archive of your action, and click Export.
- Click Save Built Products, and click Next

**Figure 25:**



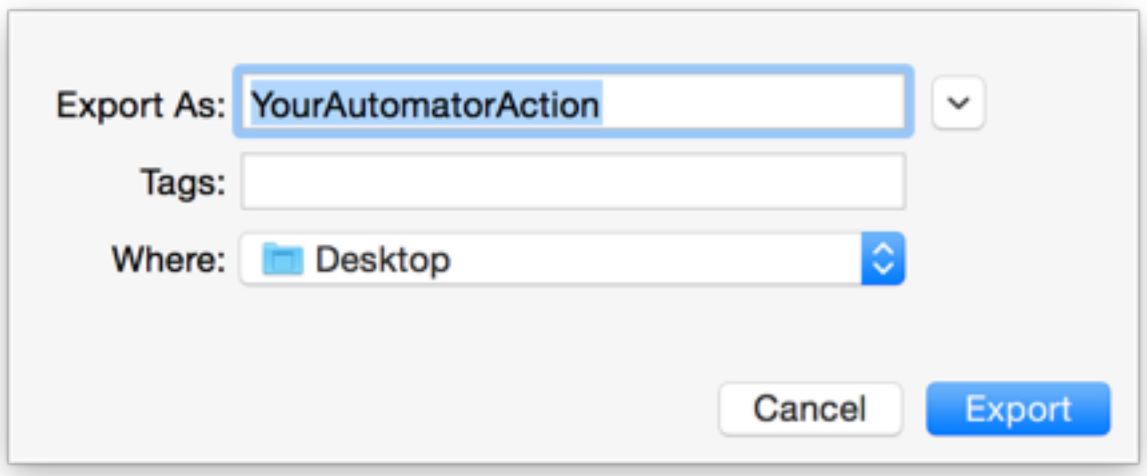
**Select a method for export:**

[✓ Save Built Products]
Save a copy of the archive's built products.

Export as an Xcode Archive
Export the entire Xcode archive of this product.

- Choose a location to save your action, and click Export.

**Figure 26:**


Export As: YourAutomatorAction

Tags:

Where: Desktop

After you’ve exported your action, it can be installed into one of the following locations:

`/Library/Automator/` – Install it here to make it available to all users.

`~/Library/Automator/` – Install it here to make it available to the current user.

When Automator launches, it scans these folders and loads any actions that it finds.

If you’re an app developer, you can install the action into a `/Contents/Library/Automator/` directory within your app bundle and Automator will find it there, too.

Back to Top

## Document Revision History

| Date | Notes |
|---|---|
| 2015-01-26 | New document that explains how to create a Cocoa-AppleScript (AppleScriptObjC) Automator action. |