Automator Framework Reference

2006-05-23

Contents

Figures

Automator Objective-C Reference

Framework	/System/Library/Frameworks/Automator.framework
Header file directories	/System/Library/Frameworks/Automator.framework/Headers

The Automator framework supports the development of actions for the Automator application. An action is a bundle loaded by the Automator application that performs a specific task, such as copying a file or cropping an image. Using the application, users can construct and execute workflows consisting of a sequence of actions. As a workflow executes, Automator passes the output of an action to the next action in the workflow. Automator loads action bundles from standard locations in the file system: `/System/Library/Automator`, `/Library/Automator`, and `~/Library/Automator`.

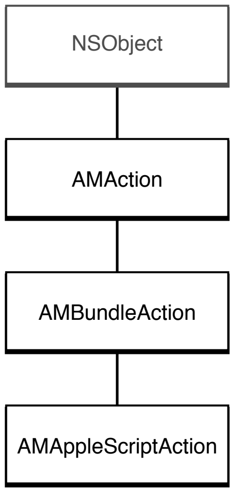
The Classes of the Automator Framework

The Automator framework has four public classes:

- AMAction (page 11) is the abstract superclass for all actions. It defines the essential behavior and characteristics of all actions.
- AMBundleAction (page 19), a concrete subclass of `AMAction`, implements actions as loaded bundles. If your action is Cocoa-based, you typically implement a custom subclass of `AMBundleAction`.
- AMAppleScriptAction (page 17), a subclass of `AMBundleAction`, implements actions that are driven by AppleScript scripts.
- AMShellScriptAction (page 25), as subclass of `AMBundleAction`, implements actions that are driven by shell scripts, Perl scripts, or Python scripts.

The following diagram depicts the hierarchical relationships of these classes.

Figure I-1 Class hierarchy



P A R T   I

Classes

PART I

Classes

AMAction Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	Automator.framework
Declared in:	Automator/AMAction.h
Companion guide:	Automator Programming Guide

Class Description

`AMAction` is an abstract class that defines the interface and general characteristics of Automator actions. Automator is an Apple-provided application that allows users to construct and execute workflows consisting of a sequence of discrete modules called actions. An action performs a specific task, such as copying a file or cropping an image, and passes its output to Automator to give to the next action in the workflow. Actions are currently implemented as loadable bundles owned by objects of the `AMBundleAction` class, a subclass of `AMAction`.

The critically important method declared by `AMAction` is `runWithInput:fromAction:error:` (page 15). When Automator executes a workflow, it sends this message to each action object in the workflow (in workflow sequence), in most cases passing in the output of the previous action as input. The action object performs its task in this method and ends by returning an output object for the next action in the workflow.

Subclassing Notes

Subclassing `AMAction` is not recommended. For most situations requiring an enhancement to the Automator framework, it is sufficient to subclass `AMBundleAction` or one of its public subclasses, `AMAppleScriptAction` or `AMShellScriptAction`.

Methods by Task

Initialization and encoding

- `definition` (page 13)
  Returns the definition of the receiver.

- `initWithDefinition:fromArchive:` (page 13)
  Initializes the receiver with the definition *`dict`*.

- `writeToDictionary:` (page 16)
  Implemented by the receiver to examine the parameters and other configuration information in *`dictionary`* and add its own information to it if appropriate.

Controlling the action

- `reset` (page 14)
  Resets the receiver to its initial state.

- `runWithInput:fromAction:error:` (page 15)
  Requests the receiver to perform its task using the input object *`input`* passed it from the action *`anAction`*.

- `stop` (page 15)
  Stops the receiver from running.

Initializing and synchronizing the action user interface

- `activated` (page 13)
  Invoked when the window of the Automator workflow to which the receiver belongs becomes the main window. This allows the receiver to synchronize its information with settings in another application.

- `opened` (page 14)
  Invoked when the receiver is first added to a workflow, allowing it to initialize its user interface.

Updating action parameters

- `parametersUpdated` (page 14)
  Requests the receiver to update its user interface from its stored parameters, which have changed.

- `updateParameters` (page 15)
  Requests the receiver to update its stored set of parameters from the settings in the action's user interface.

Instance Methods

activated

Invoked when the window of the Automator workflow to which the receiver belongs becomes the main window. This allows the receiver to synchronize its information with settings in another application.

```
- (void)activated
```

Discussion
Be sure to invoke the superclass implementation of this method as the last thing in your implementation.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- opened` (page 14)

definition

Returns the definition of the receiver.

```
- (NSMutableDictionary *)definition
```

Discussion
The definition is a mutable dictionary containing the current parameters of the action as well as other information. If your action has non-persistent data, it may override this method to append that data to the superclass definition and return it.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- initWithDefinition:fromArchive:` (page 13)

initWithDefinition:fromArchive:

Initializes the receiver with the definition *`dict`*.

```
- (id)initWithDefinition:(NSDictionary *)dict fromArchive:(BOOL)archived
```

Discussion
If the receiver is being unarchived, *`archived`* is `YES`. The `AMAction` object being instantiated should perform whatever initializations are necessary after invoking `super`'s implementation of this method. It can then examine the values in *`dict`*, particularly if the receiver had been archived with custom definition properties. This is the primary initializer for all Automator classes. The Automator application sends this message to `AMAction` objects when it both loads actions bundles and unarchives them.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `definition` (page 13)
- `writeToDictionary:` (page 16)

opened

Invoked when the receiver is first added to a workflow, allowing it to initialize its user interface.

```
- (void)opened
```

Discussion
You should perform all initializations of an action's user interface in this method and not in `awakeFromNib`. Be sure to invoke the superclass implementation of this method as the final step of your implementation.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `activated` (page 13)

parametersUpdated

Requests the receiver to update its user interface from its stored parameters, which have changed.

```
- (void)parametersUpdated
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `updateParameters` (page 15)

reset

Resets the receiver to its initial state.

```
- (void)reset
```

Discussion
Resetting causes the action to release its output generated from the current execution of the workflow.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `stop` (page 15)

runWithInput:fromAction:error:

Requests the receiver to perform its task using the input object *input* passed it from the action *anAction*.

```
- (id)runWithInput:(id)input fromAction:(AMAction *)anAction error:(NSDictionary
    **)errorInfo
```

Discussion

The *input* object contains one or more objects compatible with one of the types specified in the action's `AMAccepts` property. The action receiving this message must return an object for its output; this output must contain one or more objects of a data type compatible with a type specified in the action's `AMProvides` property. (Input and output objects are usually instances of `NSArray` or, in AppleScript, lists containing one or more objects of an accepted or provided type.) If the receiver does not modify the input passed it, it should return it unchanged. If the receiver encounters problems, it should return by indirection dictionary *errorInfo* to describe the error. The keys and values for this dictionary are:

- `OSAScriptErrorNumber` — An instance of `NSNumber` whose integer value indicates an error code. See the header file `MacErrors.h` in the Carbon Core framework for a list of valid error codes, particularly the section on OSA errors.
- `OSAScriptErrorMessage` —An instance of `NSString` describing the error.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- reset` (page 14)
`- stop` (page 15)

stop

Stops the receiver from running.

```
- (void)stop
```

Discussion

The output acquired by the action during execution of the current workflow is still accessible to Automator.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- reset` (page 14)

updateParameters

Requests the receiver to update its stored set of parameters from the settings in the action's user interface.

```
- (void)updateParameters
```

Discussion
This message is sent just before an action is saved, copied, or run. Preferably, an action's settings should not solely reside in the controls of its view, but if they do, the action can fetch and save them in this method.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- parametersUpdated` (page 14)

writeToDictionary:

Implemented by the receiver to examine the parameters and other configuration information in *`dictionary`* and add its own information to it if appropriate.

```
- (void)writeToDictionary:(NSMutableDictionary *)dictionary
```

Discussion
Automator sends this message to an action object prior to archiving it. In its implementation of this method the action object should first invoke the superclass implementation.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- initWithDefinition:fromArchive:` (page 13)

AMAppleScriptAction Class Reference

Inherits from:	AMBundleAction : NSObject
Conforms to:	NSObject (NSObject) NSCoding NSCopying
Framework	Automator.framework
Declared in:	Automator/AMApppleScriptAction.h
Companion guide:	Automator Programming Guide

Class Description

Instances of the `AMAppleScriptAction` class own Automator actions whose runtime behavior is driven by an AppleScript script. An `AMAppleScriptAction` object holds the compiled script as an instance of the `OSAScript` class. By default, the `OSAScript` object is instantiated from the script in the Xcode project file `main.applescript`.

When you create a Automator Applescript Action project in XCode, the project template supplies an `AMAppleScriptAction` instance as File's Owner of the action bundle. This ready-made instance provides a default implementation of the `AMAction` `runWithInput:fromAction:error:` (page 15) method that uses the logic defined in the script. You can substitute your own subclass of `AMAppleScriptAction` for File's Owner if you need to.

Methods by Task

Accessing the script

`- script` (page 18)
Returns the `OSAScript` object representing the receiver's script containing the `on run` command handler.

`- setScript:` (page 18)
: Set's the receiver's script to *`newScript`*.

Instance Methods

script

Returns the `OSAScript` object representing the receiver's script containing the `on run` command handler.

```
- (OSAScript *)script
```

Discussion
By default, this script is `main.applescript`, which is stored in the action bundle.

Availability
Available in Mac OS X v10.4 and later.

setScript:

Set's the receiver's script to *`newScript`*.

```
- (void)setScript:(OSAScript *)newScript
```

Discussion
`newScript` must be an `OSAScript` object that could be instantiated from a script in the action bundle. The script must contain the `on run` command handler.

Availability
Available in Mac OS X v10.4 and later.

AMBundleAction Class Reference

Inherits from:	`AMAction` : NSObject
Conforms to:	NSObject (NSObject) NSCoding NSCopying
Framework	Automator.framework
Declared in:	Automator/AMBundleAction.h
Companion guide:	Automator Programming Guide

Class Description

Instances of the `AMBundleAction` class (or its public subclasses) manage Automator actions that are loadable bundles. Automator loads action bundles from standard locations in the file system: `/System/Library/Automator`, `/Library/Automator`, and `~/Library/Automator`.

`AMBundleAction` objects have several important properties:

- The `NSBundle` object associated with the action's physical bundle
- The action's view, which holds its user interface
- A parameters dictionary that reflects the settings in the user interface

When you create a Cocoa Automator Action project in Xcode, the project template includes a custom subclass of `AMBundleAction`. (This custom class is given the name of the project.) The sole requirement for this custom class is to provide an implementation of `runWithInput:fromAction:error:` (page 15), which is declared by the superclass `AMAction`.

Subclassing Notes

As noted in "Class Description" (page 19), the project template for Cocoa Automator actions includes partially completed header and source files for a custom subclass of `AMBundleAction`. The name of this custom class is the name of the XCode project. To complete the implementation of this subclass, you must override `runWithInput:fromAction:error:` (page 15) (declared by `AMAction`).

Another reason for subclassing `AMBundleAction` is to obtain a class that is a peer to `AMAppleScriptAction`, itself a subclass of `AMBundleAction`. For example, the `AMShellScriptAction` class is a subclass of `AMBundleAction` whose instances can control the behavior of an action through shell, Perl, and Python scripts.

Methods to Override

To subclass `AMBundleAction`, you must override the `runWithInput:fromAction:error:` (page 15) to implement the task performed by the action. If you have added any instance variables, you must override the `initWithDefinition:fromArchive:` (page 21) method and the `writeToDictionary:` method of `AMAction` to work with them.

Methods by Task

Initializing the action

`– awakeFromBundle` (page 21)
Sent to the receiver when all objects in its bundle have been unarchived.

`– initWithDefinition:fromArchive:` (page 21)
Initializes and returns an allocated `AMBundleAction` object.

Setting and getting action properties

`– bundle` (page 21)
Returns the receiver's bundle object.

`– hasView` (page 21)
Returns whether the receiver has a view associated with it.

`– view` (page 23)
Returns the receiver's view object.

`– parameters` (page 22)
Returns the receiver's parameters.

`– setParameters:` (page 22)
Sets the parameters of the receiver to *`newParameters`*.

Instance Methods

awakeFromBundle

Sent to the receiver when all objects in its bundle have been unarchived.

```
- (void)awakeFromBundle
```

Discussion
This method allows an action object to perform set-up tasks requiring the presence of all bundle objects, such as adding itself as an observer of notifications, dynamically establishing bindings, and dynamically setting targets and actions.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- initWithDefinition:fromArchive:` (page 21)

bundle

Returns the receiver's bundle object.

```
- (NSBundle *)bundle
```

Discussion
Returns `nil` if no bundle has been set.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- view` (page 23)

hasView

Returns whether the receiver has a view associated with it.

```
- (BOOL)hasView
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- view` (page 23)

initWithDefinition:fromArchive:

Initializes and returns an allocated `AMBundleAction` object.

```
- (id)initWithDefinition:(NSDictionary *)dict  fromArchive:(BOOL)archived
```

Discussion
The definitions dictionary *dict* contains configuration information specific to the receiver. If *archived* is `YES`, the definitions are coming from an archive. You may examine the definitions dictionary to learn about specific properties and settings of the action, but some of the keys are private to Automator. You should not attempt to change the values in *dict*, but you may add custom key-value pairs to the definition dictionary by overriding the `writeToDictionary:` (page 16) method declared by the superclass, `AMAction`. If at runtime you need to learn about or change the action's properties in its information property list (`Info.plist`), send the appropriate `NSDictionary` messages to the action bundle's `infoDictionary`; for example:

```
[NSDictionary *infoDict = [[self bundle] infoDictionary];
NSString *theApp = [infoDict objectForKey:@"AMApplication"];
if ([theApp isEqualToString:@"Finder"]) {
    // do something appropriate
}
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- awakeFromBundle` (page 21)
`- definition` (page 13) (AMAction)

parameters

Returns the receiver's parameters.

```
- (NSMutableDictionary *)parameters
```

Discussion
The parameters of an action reflect he choices made and values entered in the action's user interface.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setParameters:` (page 22)

setParameters:

Sets the parameters of the receiver to *newParameters*.

```
- (void)setParameters:(NSMutableDictionary *)newParameters
```

Discussion
The parameters of an action reflect the choices made and values entered in the action's user interface. Keys in the parameters dictionary identify specific user-interface objects. If an action uses the Cocoa bindings mechanism, the parameters of an `AMBundleAction` object are automatically set. You can change the parameters wholesale with this method. Or you can get the current parameters dictionary with the `parameters` (page 22) and update individual parameters.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- parameters` (page 22)

view

Returns the receiver's view object.

```
- (NSView *)view
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- hasView` (page 21)
`- bundle` (page 21)

AMShellScriptAction Class Reference

Inherits from:	AMBundleAction : NSObject
Conforms to:	NSObject (NSObject) NSCoding NSCopying
Framework	Automator.framework
Declared in:	Automator/AMShellScriptAction.h
Companion guide:	Automator Programming Guide

Class Description

Instances of the `AMShellScriptAction` class own Automator actions whose runtime behavior is driven by a shell script or by a Perl or Python script. When you create a Shell Script Automator Action project in Xcode, the project template supplies an `AMShellScriptAction` instance as File's Owner of the action bundle. This ready-made instance provides a default implementation of the `AMAction` `runWithInput:fromAction:error:` (page 15) method that uses the logic defined in the script. You can substitute your own subclass of `AMShellScriptAction` for File's Owner if you need to.

Methods by Task

Handling the I/O separator character

`– inputFieldSeparator` (page 26)
Returns the character that is used as the delimiter between items in the string passed in through standard input.

`– outputFieldSeparator` (page 26)
Returns the character used as a delimiter in the string output of the receiver.

`- remapLineEndings` (page 26)
Returns whether you want automatic remapping of carriage return (`\r`) to newline (`\n`) characters in the input string.

Instance Methods

inputFieldSeparator

Returns the character that is used as the delimiter between items in the string passed in through standard input.

```
- (NSString *)inputFieldSeparator
```

Discussion
The Automator framework converts the output from the previous action (which is usually in the form of a list or array) into a single string in which the array elements are concatenated by the input field separator. By default this separator is the newline character (@"\n"). You could, for example, override this method to return a null character (@"\0") to provide null-terminated strings for `xargs -0`.

Availability
Available in Mac OS X v10.4 and later, Xcode 2.1 and later.

outputFieldSeparator

Returns the character used as a delimiter in the string output of the receiver.

```
- (NSString *)outputFieldSeparator
```

Discussion
The Automator framework converts this string into an array (or list) whose elements are derived from the fields delimited by the separator character, and then passes the array (or list) to the next action in the workflow. The default value is the separator character returned by `inputFieldSeparator` (page 26). Override this method if you want a different delimiter for output.

Availability
Available in Mac OS X v10.4 and later, Xcode 2.1 and later.

remapLineEndings

Returns whether you want automatic remapping of carriage return (`\r`) to newline (`\n`) characters in the input string.

```
- (BOOL)remapLineEndings
```

Discussion
The default is `NO`. Override to return `YES` if you want the remapping to occur.

Availability
Available in Mac OS X v10.4 and later, Xcode 2.1 and later.

Document Revision History

This table describes the changes to *Automator Framework Reference*.

Date	Notes
2006-05-23	First publication of this content as a collection of separate documents.

Index