Automator Framework Reference

Interapplication Communication

2006-10-26

Contents

Introduction

Framework	/System/Library/Frameworks/Automator.framework
Header file directories	/System/Library/Frameworks/Automator.framework/Headers
Declared in	AMAction.h AMAppleScriptAction.h AMBundleAction.h AMShellScriptAction.h AMWorkflow.h AMWorkflowController.h AMWorkflowView.h AutomatorErrors.h

The Automator framework supports the development of actions for the Automator application, as well as the ability to run a workflow in developer applications. An action is a bundle that, when loaded and run, performs a specific task, such as copying a file or cropping an image. Using Automator, users can construct and execute workflows consisting of a sequence of actions. Developers can also load and execute workflows in their applications. As a workflow executes, the output of one action is typically passed as the input to the next action. Automator loads action bundles from standard locations in the file system: `/System/Library/Automator`, `/Library/Automator`, and `~/Library/Automator`.

Classes

PART I

Classes

AMAction Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Automator/AMAction.h
Companion guide	Automator Programming Guide
Related sample code	Apply Firmware Password CoreRecipes UnsharpMask

Overview

`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 17). 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`.

Tasks

Initialization and Encoding

- `initWithDefinition:fromArchive:` (page 14)
  Initializes the receiver with the specified definition.
- `initWithContentsOfURL:error:` (page 13)
  Loads an Automator action from a file URL.
- `writeToDictionary:` (page 19)
  Examines the parameters and other configuration information specified in the passed dictionary and add its own information to it if appropriate.
- `definition` (page 12) Deprecated in Mac OS X v10.4
  Returns the definition of the receiver. (Deprecated. Removed for performance reasons. There is no replacement.)

Controlling the Action

- `reset` (page 16)
  Resets the receiver to its initial state.
- `runAsynchronouslyWithInput:` (page 16)
  Causes Automator to wait for notification that the receiver has completed execution, which allows the receiver to perform an asynchronous operation.
- `runWithInput:fromAction:error:` (page 17)
  Requests the receiver to perform its task using the specified input from the specified action.
- `stop` (page 18)
  Stops the receiver from running.

Initializing and Synchronizing the Action User Interface

- `activated` (page 11)
  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 15)
  Invoked when the receiver is first added to a workflow, allowing it to initialize its user interface.

Updating Action Parameters

- `parametersUpdated` (page 15)
  Requests the receiver to update its user interface from its stored parameters, which have changed.
- `updateParameters` (page 18)
  Requests the receiver to update its stored set of parameters from the settings in the action's user interface.

Getting Information About the Action

- `closed` (page 11)
  Invoked by Automator when the receiving action is removed from a workflow, allowing it to perform cleanup operations.
- `didFinishRunningWithError:` (page 12)
  Sent by the receiver to itself when it has finished running asynchronously.
- `ignoresInput` (page 13)
  Returns a Boolean value that indicates whether the action acts upon its input or the input is ignored.
- `name` (page 14)
  Returns the name of the action.
- `output` (page 15)
  Returns the receiver's output.
- `willFinishRunning` (page 18)
  Invoked by Automator when the receiver has essentially completed its run phase.

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 15)

Declared In
`AMAction.h`

closed

Invoked by Automator when the receiving action is removed from a workflow, allowing it to perform cleanup operations.

```
- (void)closed
```

Discussion
This method is intended to be overridden, so that your action can perform its specific cleanup operations.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMAction.h`

definition

Returns the definition of the receiver. (Deprecated in Mac OS X v10.4. Removed for performance reasons. There is no replacement.)

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

Return Value
A mutable dictionary containing the current parameters of the action as well as other information.

Discussion
If your action has non-persistent data, it may override this method to append that data to the definition supplied by the superclass and return it.

Special Considerations

This method was removed to allow for performance improvements in the underlying implementation. There is no replacement.

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

See Also
`- initWithDefinition:fromArchive:` (page 14)

Declared In
`AMAction.h`

didFinishRunningWithError:

Sent by the receiver to itself when it has finished running asynchronously.

```
- (void)didFinishRunningWithError:(NSDictionary *)errorInfo
```

Parameters

`errorInfo`
If an error occurred during asynchronous running of the action, upon return contains an instance of `NSError` that describes the problem.

Discussion
An action that overrides `runAsynchronouslyWithInput:` (page 16) should invoke `didFinishRunningWithError:` on completion, so that Automator can resume running the workflow that the action is part of.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- willFinishRunning` (page 18)

Related Sample Code
AutomatorHandsOn

Declared In
`AMAction.h`

ignoresInput

Returns a Boolean value that indicates whether the action acts upon its input or the input is ignored.

```
- (BOOL)ignoresInput
```

Return Value
`YES` if the action acts upon its input, otherwise `NO`.

Discussion
Many actions act upon their input, but an action may merely pass on its input or, rarely, ignore it.

Special Considerations

Although this method was documented in Mac OS X version 10.4, and an action would respond to this message, the method was not made public, and using it would generate a warning in Xcode.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMAction.h`

initWithContentsOfURL:error:

Loads an Automator action from a file URL.

```
- (id)initWithContentsOfURL:(NSURL *)fileURLerror:(NSError **)outError
```

Parameters

`fileURL`
URL that specifies the location of an action file.

`outError`
If no action is found or if an error occurs in initializing or running it, upon return contains an instance of `NSError` that describes the problem. For keys and error constants used with action errors, see *Automator Constants Reference*.

Return Value
The initialized action.

Discussion
This method is typically invoked by applications that use the `AMWorkflow` class to embed Automator workflows. It is used to allow creation of actions for a workflow.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMAction.h`

initWithDefinition:fromArchive:

Initializes the receiver with the specified definition.

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

Parameters

`dict`
Describes the action, including any custom definition properties.

`archived`
If the receiver is being unarchived, `YES`, otherwise `NO`.

Return Value
The initialized action.

Discussion
This is the primary initializer for all Automator classes. The Automator application sends this message to instances of `AMAction` both when it loads actions bundles and when it unarchives them.

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.

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

See Also
`– definition` (page 12)
`– writeToDictionary:` (page 19)

Declared In
`AMAction.h`

name

Returns the name of the action.

```
- (NSString *)name
```

Return Value
The name of the receiving action.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMAction.h`

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 11)

Declared In
`AMAction.h`

output

Returns the receiver's output.

```
- (id)output
```

Return Value
The receiving action's output, or `nil` if called before the action is run.

Discussion
This method is used in conjunction with the `AMWorkflow` class, which allows access to the actions in a workflow. Within a workflow, for example, you might iteratively inspect the output of each action. Or, on completion of a workflow, you might examine the output of the last action, to determine the output of the workflow.

Availability
Available in Mac OS X v10.5 and later.

Related Sample Code
AutomatorHandsOn

CoreRecipes

UnsharpMask

Declared In
`AMAction.h`

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 18)

Declared In
`AMAction.h`

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 18)

Declared In
`AMAction.h`

runAsynchronouslyWithInput:

Causes Automator to wait for notification that the receiver has completed execution, which allows the receiver to perform an asynchronous operation.

```
- (void)runAsynchronouslyWithInput:(id)input
```

Parameters

`input`
The input for the action. Should contain one or more objects compatible with one of the types specified in the action's `AMAccepts` property.

Discussion
This method should be overridden only by actions that need to make asynchronous calls. After `runAsynchronouslyWithInput:` is invoked, Automator does not continue until the action invokes `didFinishRunningWithError:` (page 12). So in your override of this method, you can make an asynchronous call, wait to be notified of its completion, then invoke `didFinishRunningWithError:` to signal to Automator that the action has completed.

Warning: Failure to invoke `didFinishRunningWithError:` can cause a workflow to stall indefinitely.

For actions that do not need to make asynchronous calls, the preferred method is `runWithInput:fromAction:error:` (page 17).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMAction.h`

runWithInput:fromAction:error:

Requests the receiver to perform its task using the specified input from the specified action.

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

Parameters

`input`

The input for the receiving action. Should contain one or more objects compatible with one of the types specified in the action's `AMAccepts` property.

By default, actions can only accept and provide the following types. However, by overriding this method, you can change the types your action can use:

- Objective-C actions: Accepts and provides types must inherit from com.apple.cocoa.string, com.apple.cocoa.path, com.apple.cocoa.url, or, starting in Mac OS X version 10.5 (v10.5), com.apple.cocoa.data.
- Shell script actions: Accepts and provides types must inherit from com.apple.cocoa.string or, starting in Mac OS X v10.5, com.apple.cocoa.data.
- AppleScript actions: Accepts and provides types must inherit from com.apple.applescript.object.

`anAction`

The action from which the *`input`* object was obtained.

`errorInfo`

If an error occurs, the action returns an error dictionary in this parameter. The keys and values for this dictionary are:

- `OSAScriptErrorNumber` (a string constant) — The value for this key is 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` (a string constant) —The value for this key is an instance of `NSString` describing the error.

For an example of how to create such a dictionary, see "Implementing runWithInput:fromAction:error:" in "Implementing an Objective-C Action" in *Automator Programming Guide*.

Return Value
An object containing one or more objects of a data type compatible with a type specified in the receiving action's `AMProvides` property. If the receiver does not modify the data passed in *`input`*, it should return it unchanged.

Discussion
The input and output objects for actions are usually instances of `NSArray`. If the receiver encounters problems, it should return by indirection an error dictionary that describes the error.

This method is intended to be overridden. AppleScript actions, however, usually will not need to override this method because the same functionality is provided by an AppleScript script.

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

See Also

- `reset` (page 16)
- `runAsynchronouslyWithInput:` (page 16)
- `stop` (page 18)

Declared In

`AMAction.h`

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 16)

Declared In

`AMAction.h`

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 15)

Declared In

`AMAction.h`

willFinishRunning

Invoked by Automator when the receiver has essentially completed its run phase.

```
- (void)willFinishRunning
```

Discussion
This method is intended to be overridden. An action can use this method to perform cleanup operations, such as closing windows and deallocating memory.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMAction.h`

writeToDictionary:

Examines the parameters and other configuration information specified in the passed dictionary and add its own information to it if appropriate.

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

Parameters

dictionary

Possibly contains parameter and other configuration information about the receiver.

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 14)

Declared In
`AMAction.h`

AMAppleScriptAction Class Reference

Inherits from	AMBundleAction : AMAction : NSObject
Conforms to	NSCoding (AMBundleAction) NSCopying (AMBundleAction) NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Automator/AMApppleScriptAction.h
Companion guide	Automator Programming Guide

Overview

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 17) 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.

Tasks

Accessing the Script

`- script` (page 22)
   Returns the `OSAScript` object representing the receiver's script containing the `on run` command handler.

`- setScript:` (page 22)
   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.

Declared In

`AMAppleScriptAction.h`

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.

Declared In

`AMAppleScriptAction.h`

AMBundleAction Class Reference

Inherits from	AMAction : NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Automator/AMBundleAction.h
Companion guide	Automator Programming Guide
Related sample code	Apply Firmware Password AutomatorHandsOn CoreRecipes UnsharpMask

Overview

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 17), which is declared by the superclass `AMAction`.

Subclassing Notes

As noted in "Class Description" (page 23), 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 17) (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 17) to implement the task performed by the action. If you have added any instance variables, you must override the `initWithDefinition:fromArchive:` (page 26) method and the `writeToDictionary:` method of `AMAction` to work with them.

Tasks

Initializing the Action

- `awakeFromBundle` (page 25)
  Sent to the receiver when all objects in its bundle have been unarchived.
- `initWithDefinition:fromArchive:` (page 26)
  Initializes and returns an allocated `AMBundleAction` object.

Setting and Getting Action Properties

- `bundle` (page 25)
  Returns the receiver's bundle object.
- `hasView` (page 25)
  Returns whether the receiver has a view associated with it.
- `view` (page 27)
  Returns the receiver's view object.
- `parameters` (page 26)
  Returns the receiver's parameters.
- `setParameters:` (page 27)
  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 26)

Declared In
`AMBundleAction.h`

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 27)

Declared In
`AMBundleAction.h`

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 27)

Declared In
`AMBundleAction.h`

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 19) 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 25)
`- definition` (page 12) (AMAction)

Declared In
`AMBundleAction.h`

parameters

Returns the receiver's parameters.

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

Discussion
The parameters of an action reflect the 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 27)

Related Sample Code
CoreRecipes

Declared In
`AMBundleAction.h`

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 26) and update individual parameters.

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

See Also
`- parameters` (page 26)

Declared In
`AMBundleAction.h`

view

Returns the receiver's view object.

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

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

See Also
`- hasView` (page 25)
`- bundle` (page 25)

Declared In
`AMBundleAction.h`

AMShellScriptAction Class Reference

Inherits from	AMBundleAction : AMAction : NSObject
Conforms to	NSCoding (AMBundleAction) NSCopying (AMBundleAction) NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	Automator/AMShellScriptAction.h
Companion guide	Automator Programming Guide

Overview

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 17) 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.

Tasks

Handling the I/O Separator Character

`– inputFieldSeparator` (page 30)
: Returns the character that is used as the delimiter between items in the string passed in through standard input.

`– outputFieldSeparator` (page 30)
: Returns the character used as a delimiter in the string output of the receiver.

`– remapLineEndings` (page 30)
: 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.

Declared In
`AMShellScriptAction.h`

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 30). 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.

Declared In
`AMShellScriptAction.h`

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.

Declared In

`AMShellScriptAction.h`

AMWorkflow Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Automator/AMWorkflow.h
Companion guides	Automator Programming Guide Automator Framework Reference
Related sample code	AutoSample

Overview

The `AMWorkflow` class lets you use an Automator workflow in your application. You can display a workflow with an instance of `AMWorkflowView` and control its operation with an instance of `AMWorkflowController`.

A workflow consists of one or more actions (discrete tasks), which together can perform complex automation tasks. Your application can use workflows to package its own features and to take advantage of features provided by other applications. You create actions with Xcode, while you create workflows with the Automator application.

You can load and run a workflow with minimal overhead by using the `AMWorkflow` class method `runWorkflowAtURL:withInput:error:` (page 35). However, in situations where you need greater control, such as the ability to start and stop the workflow, you can use an instance of the `AMWorkflowController` class instead. In that case, you'll have to create and initialize both a workflow and a workflow controller object.

In either case, the workflow is run in a separate process so that any actions it contains are executed in a separate memory space. That helps to insulate your application from crashes, memory leaks, or exceptions that might occur from running the actions in the workflow.

Tasks

Running a Specified Workflow

`+ runWorkflowAtURL:withInput:error:` (page 35)
Loads and runs the specified workflow file.

Creating and Initializing a Workflow

`- initWithContentsOfURL:error:` (page 36)
Creates and initializes a workflow based on the contents of the specified file.

Saving Changes to a Workflow

`- writeToURL:error:` (page 40)
Writes the workflow to the specified file.

Getting Information About the Workflow

`- actions` (page 35)
Returns an array of the workflow's actions.

`- fileURL` (page 36)
Returns a URL that specifies the location of the workflow file.

`- input` (page 37)
Returns the input data that is passed to the first action in the workflow.

`- setInput:` (page 38)
Sets the input data that is passed to the first action in the workflow.

`- setValue:forVariableWithName:` (page 39)
Sets the value of the workflow variable with the specified name.

`- valueForVariableWithName:` (page 39)
Returns the value of the workflow variable with the specified name.

Manipulating the Workflow's Actions

`- addAction:` (page 36)
Adds the specified action at the end of the receiving workflow.

`- insertAction:atIndex:` (page 37)
Inserts the specified action at the specified position of the receiving workflow.

`- moveActionAtIndex:toIndex:` (page 38)
Moves the action from the specified start position to the specified end position in the receiving workflow.

`- removeAction:` (page 38)
: Removes the specified action from the receiver.

Class Methods

runWorkflowAtURL:withInput:error:

Loads and runs the specified workflow file.

```
+ (id)runWorkflowAtURL:(NSURL *)fileURL withInput:(id)input error:(NSError **)error
```

Parameters

fileURL
: URL that specifies the location of a workflow file.

input
: Specifies the input for the first action in the workflow. Pass `nil` if the first action doesn't need input.

error
: If no workflow is found or if an error occurs in initializing or running it, upon return contains an instance of `NSError` that describes the problem.

Return Value
On error, returns `nil`. Otherwise, returns the output of the last action in the workflow. Your application may need to convert the data to a desired type.

Discussion
Use this method to run a workflow without the overhead of performing a separate allocation, setting up a workflow controller, and so on. However, in situations where you need greater control, such as the ability to start and stop the workflow, use an instance of the `AMWorkflowController` class instead.

The workflow is run in a separate process so that any actions it contains are executed in a separate memory space. This helps to insulate the application from crashes, memory leaks, or exceptions that might occur from running the actions in the workflow.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

Instance Methods

actions

Returns an array of the workflow's actions.

```
- (NSArray *)actions
```

Return Value
An array of actions for the workflow file. Actions are instances of classes such as `AMBundleAction`, `AMAppleScriptAction`, and `AMShellScriptAction`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

addAction:

Adds the specified action at the end of the receiving workflow.

```
- (void)addAction:(AMAction *)action
```

Parameters

action
The action to add.

Discussion
The workflow retains the action but does not copy it.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

fileURL

Returns a URL that specifies the location of the workflow file.

```
- (NSURL *)fileURL
```

Return Value
URL that specifies the location of the workflow file.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

initWithContentsOfURL:error:

Creates and initializes a workflow based on the contents of the specified file.

```
- (id)initWithContentsOfURL:(NSURL *)fileURL error:(NSError **)outError
```

Parameters

fileURL
: URL that specifies the location of a workflow file.

outError
: If the workflow file can't be found, or if an error occurs in initializing the workflow, upon return contains an instance of `NSError` that describes the problem.

Return Value
The initialized workflow object. On error, returns nil.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

input

Returns the input data that is passed to the first action in the workflow.

```
- (id)input
```

Return Value
The input for the first action in the workflow. Should be a data type the action can use, or a type that can be converted to one the action can use.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setInput:` (page 38)

Declared In
`AMWorkflow.h`

insertAction:atIndex:

Inserts the specified action at the specified position of the receiving workflow.

```
- (void)insertAction:(AMAction *)action atIndex:(NSUInteger)index
```

Parameters

action
: The action to insert.

index
: The position in the receiver at which to insert the action. If the position is invalid, this method does nothing.

Discussion
The workflow retains the action but does not copy it.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

moveActionAtIndex:toIndex:

Moves the action from the specified start position to the specified end position in the receiving workflow.

```
- (void)moveActionAtIndex:(NSUInteger)startIndex toIndex:(NSUInteger)endIndex
```

Parameters

startIndex
The start position of the action to move.

endIndex
The end position for the action that is moved.

Discussion
If either index is invalid, this method does nothing.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

removeAction:

Removes the specified action from the receiver.

```
- (void)removeAction:(AMAction *)action
```

Parameters

action
The action to be removed.

Discussion
The action receives an `AMAction closed` message before being released.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflow.h`

setInput:

Sets the input data that is passed to the first action in the workflow.

```
- (void)setInput:(id)input
```

Parameters

input

The input for the first action in the workflow. Should be a data type the action can use, or a type that can be converted to one the action can use.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- input` (page 37)

Declared In

`AMWorkflow.h`

setValue:forVariableWithName:

Sets the value of the workflow variable with the specified name.

```
- (BOOL)setValue:(id)value forVariableWithName:(NSString *)variableName
```

Parameters

value

The value to set for the named variable.

variableName

The name of a variable to set the value for.

Return Value

`YES` if *variableName* was found and its value set; otherwise `NO`.

Discussion

This method does nothing if the variable specified by *variableName* is not found.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`AMWorkflow.h`

valueForVariableWithName:

Returns the value of the workflow variable with the specified name.

```
- (id)valueForVariableWithName:(NSString *)variableName
```

Parameters

variableName

The variable name.

Return Value

The value for the variable. Returns `nil` if no variable is found with the specified name.

Availability

Available in Mac OS X v10.5 and later.

See Also
`- setValue:forVariableWithName:` (page 39)

Declared In
`AMWorkflow.h`

writeToURL:error:

Writes the workflow to the specified file.

```
- (BOOL)writeToURL:(NSURL *)fileURL error:(NSError **)outError
```

Parameters

fileURL
URL that specifies the file location to write the workflow to.

outError
If the workflow file can't be written, upon return contains an instance of `NSError` that describes the problem.

Return Value
`YES` if the workflow was successfully written; otherwise `NO`.

Discussion
You might want to save the workflow, for example, because you have made changes to a variable it contains.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setValue:forVariableWithName:` (page 39)

Declared In
`AMWorkflow.h`

AMWorkflowController Class Reference

Inherits from	NSController : NSObject
Conforms to	NSCoding (NSController) NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Automator/AMWorkflowController.h
Companion guides	Automator Programming Guide Automator Framework Reference
Related sample code	AutoSample

Overview

The `AMWorkflowController` class lets you manage an Automator workflow in your application. You use the `AMWorkflow` class to instantiate a workflow and an instance of `AMWorkflowView` to display it.

A controller can run and stop a workflow and obtain information about its state. The controller's delegate can receive messages as the workflow is executed and its actions are run.

You can load and run a workflow with minimal overhead by using the `AMWorkflow` class method `runWorkflowAtURL:withInput:error:` (page 35). When you use `AMWorkflowController`, you get more control of the process, but there's more work, as you must create and initialize both the workflow and the workflow controller objects.

A workflow is run in a separate process so that any actions it contains are executed in a separate memory space. That helps to insulate your application from crashes, memory leaks, or exceptions that might occur from running the actions in a workflow.

Tasks

Accessing the Workflow

`– setWorkflow:` (page 45)
Sets the receiver's workflow.

– `workflow` (page 46)
    Returns the receiver's workflow.

Accessing the Workflow View

– `setWorkflowView:` (page 45)
    Sets the receiver's workflow view.

– `workflowView` (page 46)
    Returns the receiver's workflow view.

Accessing the Delegate

– `delegate` (page 43)
    Returns the receiver's delegate.

– `setDelegate:` (page 44)
    Sets the receiver's delegate.

Controlling the Workflow

– `canRun` (page 43)
    Returns a Boolean value that indicates whether the receiver's workflow is able to run.

– `isRunning` (page 44)
    Returns a Boolean value that indicates whether the receiver's workflow is currently running.

– `run:` (page 44)
    Runs the associated workflow, after first clearing any results stored by its actions during any previous run.

– `stop:` (page 45)
    Stops the associated workflow.

– `workflowControllerWillRun:` (page 48) *delegate method*
    Invoked when the workflow is about to run.

– `workflowControllerDidRun:` (page 48) *delegate method*
    Invoked after the workflow is run.

– `workflowControllerWillStop:` (page 48) *delegate method*
    Invoked when the workflow is about to stop.

– `workflowControllerDidStop:` (page 48) *delegate method*
    Invoked after the workflow stops.

Running an Action in the Workflow

– `workflowController:didRunAction:` (page 47) *delegate method*
    Invoked when an action in the receiver's workflow is finished running.

– `workflowController:willRunAction:` (page 47) *delegate method*
    Invoked when an action in the receiver's workflow is about to run.

Errors Messages

`– workflowController:didError:` (page 46) *delegate method*
Invoked when the receiver's workflow encounters an error.

Instance Methods

canRun

Returns a Boolean value that indicates whether the receiver's workflow is able to run.

```
- (BOOL)canRun
```

Return Value
`YES` if the controller's workflow is able to run; `NO` otherwise.

Discussion
You might use this method to determine when to enable a "Run" button or other UI element you use to run the workflow.

Availability
Available in Mac OS X v10.5 and later.

See Also
`– isRunning` (page 44)

Declared In
`AMWorkflowController.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The controller's delegate.

Availability
Available in Mac OS X v10.5 and later.

See Also
`– setDelegate:` (page 44)

Declared In
`AMWorkflowController.h`

isRunning

Returns a Boolean value that indicates whether the receiver's workflow is currently running.

```
- (BOOL)isRunning
```

Return Value
`YES` if the controller's workflow is currently running; `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- canRun` (page 43)

Declared In
`AMWorkflowController.h`

run:

Runs the associated workflow, after first clearing any results stored by its actions during any previous run.

```
- (IBAction)run:(id)sender
```

Parameters

sender
    Object that initiated the run action.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- stop:` (page 45)

Declared In
`AMWorkflowController.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate
    The delegate object to set. This object will receive updates on the progress and state of the workflow controller.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- delegate` (page 43)

Declared In
`AMWorkflowController.h`

setWorkflow:

Sets the receiver's workflow.

```
- (void)setWorkflow:(AMWorkflow *)workflow
```

Parameters

`workflow`
A workflow object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- workflow` (page 46)

Declared In
`AMWorkflowController.h`

setWorkflowView:

Sets the receiver's workflow view.

```
- (void)setWorkflowView:(AMWorkflowView *)view
```

Parameters

`view`
A workflow view object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- workflowView` (page 46)

Declared In
`AMWorkflowController.h`

stop:

Stops the associated workflow.

```
- (IBAction)stop:(id)sender
```

Parameters

`sender`
Object that initiated the stop action.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- run:` (page 44)

Declared In
`AMWorkflowController.h`

workflow

Returns the receiver's workflow.

```
- (AMWorkflow *)workflow
```

Return Value
The controller's workflow.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setWorkflow:` (page 45)

Declared In
`AMWorkflowController.h`

workflowView

Returns the receiver's workflow view.

```
- (AMWorkflowView *)workflowView
```

Return Value
The controller's workflow view.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setWorkflowView:` (page 45)

Declared In
`AMWorkflowController.h`

Delegate Methods

workflowController:didError:

Invoked when the receiver's workflow encounters an error.

```
- (void)workflowController:(AMWorkflowController *)controller didError:(NSError
    *)error
```

Parameters

controller
: The controller object sending the message.

error
: If a workflow error occurs, upon return contains an instance of `NSError` that describes the problem.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflowController.h`

workflowController:didRunAction:

Invoked when an action in the receiver's workflow is finished running.

```
- (void)workflowController:(AMWorkflowController *)controller didRunAction:(AMAction
    *)action
```

Parameters

controller
: The controller object sending the message.

action
: The workflow action that ran.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflowController.h`

workflowController:willRunAction:

Invoked when an action in the receiver's workflow is about to run.

```
- (void)workflowController:(AMWorkflowController *)controller willRunAction:(AMAction
    *)action
```

Parameters

controller
: The controller object sending the message.

action
: The workflow action that will run.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflowController.h`

workflowControllerDidRun:

Invoked after the workflow is run.

```
- (void)workflowControllerDidRun:(AMWorkflowController *)controller
```

Parameters

controller

The controller object sending the message.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`AMWorkflowController.h`

workflowControllerDidStop:

Invoked after the workflow stops.

```
- (void)workflowControllerDidStop:(AMWorkflowController *)controller
```

Parameters

controller

The controller object sending the message.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`AMWorkflowController.h`

workflowControllerWillRun:

Invoked when the workflow is about to run.

```
- (void)workflowControllerWillRun:(AMWorkflowController *)controller
```

Parameters

controller

The controller object sending the message.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`AMWorkflowController.h`

workflowControllerWillStop:

Invoked when the workflow is about to stop.

```
- (void)workflowControllerWillStop:(AMWorkflowController *)controller
```

Parameters

`controller`

The controller object sending the message.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`AMWorkflowController.h`

AMWorkflowView Class Reference

Inherits from	NSView : NSResponder : NSObject
Conforms to	NSAnimatablePropertyContainer (NSView) NSCoding (NSResponder) NSObject (NSObject)
Framework	/System/Library/Frameworks/Automator.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	Automator/AMWorkflowView.h
Companion guides	Automator Programming Guide Automator Framework Reference
Related sample code	AutoSample

Overview

You use the `AMWorkflowView` class to provide viewing and editing of Automator workflows in your application. You use an instance of `AMWorkflow` to instantiate a workflow and an instance of `AMWorkflowController` to control its execution. Together, these three classes provide a Model-View-Controller suite for working with workflows.

You can use Interface Builder to add an instance of `AMWorkflowView` to a window in your application. You can then add an `AMWorkflowController` object to the nib window and use the controller's `workflowView` outlet to connect it to the workflow view. The controller object also has `run` and `stop` actions that can be connected to buttons or other user interface elements.

Tasks

Configuring the Workflow View

`– isEditable` (page 52)
  Returns a Boolean value indicating whether the workflow view is editable.

`– setEditable:` (page 52)
  Sets whether the workflow view is editable.

`- setWorkflowController:` (page 52)
  Sets the receiver's controller to the passed workflow controller.

`- workflowController` (page 53)
  Returns the receiver's workflow controller.

Instance Methods

isEditable

Returns a Boolean value indicating whether the workflow view is editable.

```
- (BOOL)isEditable
```

Return Value
`YES` if the workflow view is editable, otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflowView.h`

setEditable:

Sets whether the workflow view is editable.

```
- (void)setEditable:(BOOL)flag
```

Parameters

`flag`
  `YES` to make the workflow view editable, otherwise `NO`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AMWorkflowView.h`

setWorkflowController:

Sets the receiver's controller to the passed workflow controller.

```
- (void)setWorkflowController:(AMWorkflowController *)workflowController
```

Parameters

`workflowController`
  The controller to set for the receiver.

workflowController

Returns the receiver's workflow controller.

```
- (AMWorkflowController *)workflowController
```

Return Value
The receiver's workflow controller, or `nil` if it doesn't have a controller.

Constants

Automator Constants Reference

Framework: Automator.framework

Declared in Automator/AutomatorErrors.h

Overview

This document defines constants in the Automator framework that are not associated with a particular class.

Constants

Enumerations

NSError Codes

`NSError` codes in the Automator error domain.

```
enum {

    // Workflow errors
    AMWorkflowNewerVersionError = -100,
    AMWorkflowPropertyListInvalidError = -101,
    AMWorkflowNewerActionVersionError = -111,
    AMWorkflowOlderActionVersionError = -112,

    // Workflow runtime errors
    AMUserCanceledError = -128,
    // Action errors
    AMNoSuchActionError = -200,
    AMActionNotLoadableError = -201,
    AMActionArchitectureMismatchError = -202,
    AMActionRuntimeMismatchError = -203,
    AMActionLoadError = -204,
    AMActionLinkError = -205,
    AMActionApplicationResourceError = -206,
    AMActionApplicationVersionResourceError = -207,
    AMActionFileResourceError = -208,
    AMActionLicenseResourceError = -209,
    AMActionRequiredActionResourceError = -210,
    AMActionInitializationError = -211,
    AMActionExecutionError = -212,
    AMActionExceptionError = -213,
    AMActionPropertyListInvalidError = -214,
    AMActionInsufficientDataError = -215,
    AMActionIsDeprecatedError = -216,

    // Data conversion errors
    AMConversionNotPossibleError = -300,
    AMConversionNoDataError = -301,
    AMConversionFailedError = -302
};
```

Constants

`AMWorkflowNewerVersionError`

Attempt to open a workflow document that was saved with a newer version of Automator.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMWorkflowPropertyListInvalidError`

Attempt to open a workflow document whose property list (`document.wflow`) could not be read; the property list document could be missing, damaged, or constructed improperly.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMWorkflowNewerActionVersionError`

An action in a workflow is newer than the installed action; this error is presented to the user as a warning.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMWorkflowOlderActionVersionError`

An action in a workflow is older than the installed action; this error is presented to the user as a warning.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMUserCanceledError`

The user cancelled. This error is the same as the AppleScript error `userCanceledErr`, defined in `MacErrors.h`. When an Apple Event is canceled by the user, a running action may return this error. Automator ignores the error and stops the workflow gracefully, without displaying the error to the user.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMNoSuchActionError`

The action could not be located on the system.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionNotLoadableError`

The action's executable is of a type that is not loadable in the current process. If the action uses a custom subclass of `AMBundleAction` or `AMAppleScriptAction`, then the most likely problem is that the bundle's executable is missing or the `NSPrincipleClass` is not set in the `Info.plist`. Users are likely to be confused by a “missing bundle” error, so Automator presents it as the more generic “action not loadable” error.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionArchitectureMismatchError`

The action's binary is not compatible with the current processor; actions compiled for PowerPC, for example, would encounter this error on Intel systems.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionRuntimeMismatchError`

An attempt was made to load an action that is not compiled in a way that is compatible with the current application; for example, the action may be compiled for 32-bit applications or it may not be compatible with garbage collection.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionLoadError`

The action's executable failed to load; for example, there may have been a problem with a library it depends on.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionLinkError`

The action's executable failed to load due to linking issues.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionApplicationResourceError`

An application required by the action is not found.

Available in Mac OS X v10.5 and later.

Declared in `AutomatorErrors.h`.

`AMActionApplicationVersionResourceError`
: An application required by the action is the wrong version.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionFileResourceError`
: A file required by the action is not found.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionLicenseResourceError`
: A license required by the action was not found. The only license currently supported is QuickTime Pro.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionRequiredActionResourceError`
: An action required by the action is not loaded.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionInitializationError`
: Automator is unable to initialize an action (reason unknown).

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionExecutionError`
: An action encounters an error while running.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionExceptionError`
: An action encounters an exception while running.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionPropertyListInvalidError`
: The property list (`Info.plist`) for an action is invalid; it could be missing, damaged, or constructed improperly.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionInsufficientDataError`
: The action requires input data to run, but none was supplied.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionIsDeprecatedError`
: The action has been deprecated. Use a replacement action, if available.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMConversionNotPossibleError`
: The converter determines that it is unable to convert from one data type to another. Not displayed to the user.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMConversionNoDataError`
: The converter determines that a given conversion, though possible, would produce a `nil` result. Not displayed to the user.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMConversionFailedError`
: Occurs when, for example, the converter encounters an error converting data from one type to another.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

Discussion

The constants in this enumeration are `NSError` code numbers in the Automator error domain (`AMAutomatorErrorDomain`). You'll obtain these error codes from the instances of `NSError` returned, for example, by certain methods of `AMWorkflow` and `AMWorkflowController`. For related information, see `AMActionErrorKey` (page 61).

Declared In

`AutomatorErrors.h`

Constants

Automator Constants

These constants are used in the Automator framework.

```
#define AMAutomatorErrorDomain @"com.apple.Automator"
#define AMActionErrorKey       @"AMActionErrorKey"
```

Constants

`AMAutomatorErrorDomain`
: This constant defines the Automator error domain.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

`AMActionErrorKey`
: Use this key to obtain, from the `userInfo` dictionary of an instance of `NSError`, a reference to the action (class `AMAction`) that caused the error.

  Available in Mac OS X v10.5 and later.

  Declared in `AutomatorErrors.h`.

Declared In

`AutomatorErrors.h`

Document Revision History

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

Date	Notes
2006-10-26	Added links for new Leopard classes and constants.
	New links include *Automator Constants Reference*, as well as *AMWorkflow Class Reference*, *AMWorkflowController Class Reference*, and *AMWorkflowView Class Reference*.