AME Scripting Guide

Release 1.0

Nov 15, 2018

Introduction

CHAPTER 1

Introduction

JavaScript is a platform-independent scripting language that you can use to control many features and automate many tasks in Adobe® applications. Scripting is easier to learn and use than many other kinds of programming, and provides a convenient way of automating repetitive tasks or extending applications to provide additional tools for other users.

- If you are new to scripting, see Adobe Creative Suite: Introduction to Scripting, which introduces basic scripting concepts and describes different scripting languages that are available, including JavaScript. JavaScript and other scripting languages are object-oriented, and this book also describes the basic concepts of object-oriented programming and document object models.
- Each application that supports JavaScript also provides an application-specific Scripting Guide that introduces the object model for that application, and reference material for the objects. This document provides information about the JavaScript features, tools, and objects that are common to all Adobe applications that support JavaScript.
- This document does not teach JavaScript. If you are familiar with scripting or programming in general, but unfamiliar with JavaScript, see publicly available Web resources and documents, such as:
  - The public JavaScript standards organization web site: www.ecma-international.org
  - JavaScript: The Definitive Guide, David Flanagan, O'Reily Media Inc, 2002. ISBN 0-596-00048-0
  - JavaScript Bible, Danny Goodman, Hungry Minds Inc, 2001. ISBN 0-7645-4718-6
  - Adobe Scripting, Chandler McWilliams, Wiley Publishing, Inc., 2003. ISBN 0-7645-2455-0

Note: Check for updated versions of this document at Adobe Developer Center, http://www.adobe.com/devnet/scripting.

CHAPTER 2

ExtendScript overview

Adobe provides an extended implementation of JavaScript, called ExtendScript, that is used by many Adobe applications that provide a scripting interface. In addition to implementing the JavaScript language according to the ECMA JavaScript specification, ExtendScript provides certain additional features and utilities.

This document describes JavaScript modules, tools, utilities, and features that are available to all JavaScript-enabled Adobe applications.

Note: Some modules, and features of some modules, are optional. Check the product documentation for each application for details of which modules and features are implemented.

2.1 Example code

The Adobe ExtendScript SDK, which contains this document, also contains a set of code samples that demonstrate how to use features of ScriptUI, interapplication communication, and external communication. This book refers to these samples by name for illustration of concepts and techniques.

You can download the SDK from Adobe Developer Center, http://www.adobe.com/devnet/scripting/.

The samples are located under the ExtendScript SDK root directory:

- `SDKroot/Samples/javascript/`: sample scripts
- `SDKroot/Samples//javascript/resources/`: resources, such as image or flash files

2.2 Development and debugging tools

For help in developing, debugging, and testing scripts, Adobe provides the ExtendScript Toolkit, an interactive development and testing environment for ExtendScript, which is installed with all JavaScript-enabled applications. For complete details, see Chapter 2, the-extendscript-toolkit.

ExtendScript also provides global objects that support development and debugging:

- A global debugging object, the Dollar ($) object.
- A reporting utility for ExtendScript elements, the ExtendScript reflection interface.

For complete details, see Chapter 8, extendscript-tools-and-features.

2.3 Cross-platform file-system access

Adobe ExtendScript defines File and Folder classes that simplify cross-platform file-system access. These classes are available to all applications that support a JavaScript interface.

For complete details, see Chapter 3, file-system-access.

2.4 User-interface development tools

Adobe provides the ScriptUI module, which works with the ExtendScript JavaScript interpreter to provide JavaScript scripts with the ability to create and interact with user interface elements. It provides an object model for windows and user-interface control elements within an Adobe application. For complete details, see Chapter 4, user-interface-tools. In addition, ExtendScript provides:

- Global functions for localization of display strings; see localizing-extendscript-strings
- Global functions for displaying short messages in dialog boxes; see user-notification-dialogs.
- An object type for specifying measurement values together with their units; see specifying-measurement-values.

2.5 Interapplication communication and messaging

ExtendScript provides a common scripting environment for all Adobe JavaScript-enabled applications, and allows interapplication communication through scripts. Different levels of communication are provided through the cross-DOM and the messaging framework.

- Cross-DOM functions are a limited set of basic functions common across all message-enabled applications, which allow your script to, for example, open or print files in other applications, simply by calling the open or print function for that application. In addition to the basic set of common functions, some applications provide more extensive sets of exported JavaScript functions to other applications.
- The interapplication messaging framework is an application programming interface (API) that allows extensive control over communication between applications. The API allows you to send messages to other applications and receive results, and to receive messages sent by other applications and return results. Typically the data passed between applications are JavaScript scripts. However, the messaging framework is extensible. It allows you to define different types of data to send between applications, and to specify how they are handled.

For complete details, see Chapter 5, interapplication-communication-with-scripts.

2.6 External communication

ExtendScript offers tools for communicating with other computers or the internet using standard protocols. The Socket object supports low-level TCP connections.

For complete details, see Chapter 6, external-communication-tools.

2.7 External shared-library integration

You can extend the JavaScript DOM for an application by writing a C or C++ shared library, compiling it for the platform you are using, and loading it into JavaScript as an ExternalObject instance. A shared library is implemented by a DLL in Windows, a bundle or framework in Mac OS, or a SharedObject in UNIX.

For complete details, see Chapter 7, integrating-external-libraries.

2.8 Additional utilities and features

ExtendScript provides these utilities and features:

- **JavaScript language enhancements:**
  - Tools for combining scripts, such as a `#include` directive. See preprocessor-directives.
  - Support for extending or overriding math and logical operator behavior on a class-by-class basis. See operator-overloading.

  For complete details, see Chapter 8, extendscript-tools-and-features.
- JavaScript compilation, through the ExtendScript Toolkit. See Chapter 2, the-extendscript-toolkit.
- XML integration: ExtendScript defines the XML object, which allows you to process XML with your
- JavaScript scripts. For complete details, see Chapter 9, integrating-xml-into-javascript.
- Scripting support for XMP metadata manipulation: XMPScript provides a JavaScript API for the Adobe

CHAPTER 3

Scripting for specific applications

On startup, all Adobe JavaScript-enabled applications execute JSX files that they find in their startup directories; some of these are installed by applications, and some can be installed by scripters. The policies of different applications vary as to the locations, write access, and loading order. In addition, individual applications may look for application-specific scripts in particular directories, which may be configurable. Some applications allow access to scripts from menus; all of them allow you to load and run scripts using the ExtendScript Toolkit. For details of how to load and run scripts for any individual application, see the JavaScript Scripting Guide for that application.

3.1 Startup scripts

A script in a startup directory might be executed on startup by multiple applications. If you place a script in such a directory, it must contain code to check whether it is being run by the intended application. You can do this using the appName static property of the BridgeTalk class. For example:

```
if ( BridgeTalk.appName == "bridge" ) {
    //continue executing script
}
```

If a script that is run by one application will communicate with another application or add functionality that depends on another application, it must first check whether that application/version is installed. You can do this using the `BridgeTalk.getSpecifier()` static function. For example:

```
if ( BridgeTalk.appName == "bridge-2.0" ) {
    // Check to see that Photoshop is installed.
    if ( BridgeTalk.getSpecifier( "photoshop", 10 ) ){
        // Add the Photoshop automate menu to the Adobe Bridge UI.
    }
}
```

For details of interapplication communication, see Chapter 5, interapplication-communication-with-scripts.

3.2 JavaScript variables

Scripting shares a global environment, so any script executed at startup can define variables and functions that are available to all scripts. In all cases, variables and functions, once defined by running a script that contains them, persist in subsequent scripts during a given application session. Once the application is quit, all such globally defined variables and functions are cleared. Scripters should be careful about giving variables in scripts unique names, so that a script does not inadvertently reassign global variables intended to persist throughout a session.

CHAPTER 4

Application Object

The Application Object is the master object for controlling the application.

`app`

4.1 Attributes

4.1.1 app.launchTime

`app.launchTime`

The time in seconds AME took to launch.

Type

Number; read-only.

4.2 Methods

4.2.1 app.getFrontend()

More on the Front End Object

`app.getFrontend()`

Parameters

None.

Returns

Front End Object.

4.2.2 app.getExporter()

More on the Exporter Object

```
app.getExporter()
```

Parameters

None.

Returns

Exporter Object.

4.2.3 app.getEncoderHost()

More on the Encoder Host Object

```
app.getEncoderHost()
```

Parameters

None.

Returns

Encoder Host Object.

4.2.4 app.quit()

Quits the application.

```
app.quit()
```

Parameters

None.

Returns

None.

4.2.5 app.getBuildNumber()

Gets the application build number.

```
app.getBuildNumber()
```

Parameters

None.

Returns

String. Build number.

4.2.6 app.scheduleTask()

Schedule a task for a later time. (similar to `setTimeout()` in modern JS)

`app.scheduleTask()`

Parameters

`stringToExecute`	A string containing JavaScript to be executed.
`delay`	A number of milliseconds to wait before executing the JavaScript. A floating-point value.
`repeat`	When true, execute the script repeatedly, with the specified delay between each execution. When false the script is executed only once.

Returns

Integer, a unique identifier for this task, which can be used to cancel it with *app.cancelTask()*.

4.2.7 app.cancelTask()

Removes the specified task from the queue of tasks scheduled for delayed execution.

`app.cancelTask(taskID)`

Parameters

`taskID`	An integer that identifies the task, as returned by *app.scheduleTask()*.

Returns

Nothing.

4.2.8 app.getWatchFolder()*

`app.getWatchFolder()` **untested*

Parameters

unknown

Returns

unknown

4.2.9 app.wait()*

`app.wait()` **untested*

Parameters

unknown

Returns

unknown

4.2.10 app.assertToConsole()*

`app.assertToConsole()` **untested*

Parameters

unknown

Returns

unknown

4.2.11 app.renderFrameSequence()*

`app.renderFrameSequence()` **untested*

Parameters

unknown

Returns

unknown

4.2.12 app.isBlackVideo()*

`app.isBlackVideo()` **untested*

Parameters

unknown

Returns

unknown

4.2.13 app.isSilentAudio()*

`app.isSilentAudio()` **untested*

Parameters

unknown

Returns

unknown

4.2.14 app.simulateUIEvents()*

`app.simulateUIEvents()` **untested*

Parameters

unknown

Returns

unknown

CHAPTER 5

Front End Object

The Front End Object allows you to add items to the render queue (aka batch).

`app.getFrontend()`

5.1 Methods

5.1.1 FrontEnd.addDLToBatch()

Add Comp from After Effects or Sequences from Premiere Pro.

`app.getFrontend().addDLToBatch(project, format, preset, guid, destination)`

Parameters

`project`	Path to the project file. (e.g. .aep, .prpro)
`format`	The name of the format used in the preset EPR. (e.g. H.264, Quicktime, etc.)
`preset`	Path to the EPR preset file.
`guid`	The Dynamic Link GUID from the After Effects Comp or Premiere Pro Sequence
`destination`	Path to the destination file.

Returns

Batch Item Object

Note: Application will hang until process is complete or will throw an exception if arguments are incorrect. This method ignores work area. This method resets timecode to 00:00:00.

5.1.2 FrontEnd.addCompToBatch()

Add Comp from After Effects.

`app.getFrontend().addCompToBatch(project, preset, destination)`

Parameters

`project`	Path to the project file. (e.g. .aep, .prpro)
`preset`	Path to the EPR preset file.
`destination`	Path to the destination file.

Returns

Batch Item Object

Note: Method requires project to be structured so that 1 and only 1 comp is at the root of the project. Application will throw an exception if no comp is present in the root or more than 1 comps are present.

5.1.3 FrontEnd.addItemToBatch()

Add any footage item into the render queue.

```
app.getFrontend().addItemToBatch(file)
```

Parameters

`file`	Path to the media file.

Returns

Batch Item Object

5.1.4 FrontEnd.addFileToBatch()

Add any footage item into the render queue.

```
app.getFrontend().addFileToBatch(file, format, preset, destination)
```

Parameters

`file`	Path to the media file.
`format`	The name of the format used in the preset EPR. (e.g. H.264, Quicktime, etc.)
`preset`	Path to the EPR preset file.
`destination`	Path to the destination file.

Returns

Batch Item Object

5.1.5 FrontEnd.addFileSequenceToBatch()*

Add any footage item into the render queue.

```
app.getFrontend().addFileSequenceToBatch() *untested
```

Parameters

unknown

Returns

Batch Item Object

5.1.6 FrontEnd.addXMLToBatch()*

Add an XML project to the batch.

`app.getFrontend().addXMLToBatch()` **untested*

Parameters

unknown

Returns

Batch Item Object

5.1.7 FrontEnd.addTeamProjectsItemToBatch()*

Add team project items to the render queue.

`app.getFrontend().addTeamProjectsItemToBatch()` **untested*

Parameters

unknown

Returns

Batch Item Object

5.1.8 FrontEnd.stitchFiles()*

Stitch files for rendering.

`app.getFrontend().stitchFiles()` **untested*

Parameters

unknown

Returns

unknown

5.1.9 FrontEnd.stopBatch()*

Stops batch from running.

`app.getFrontend().stopBatch()`

Parameters

None.

Returns

Result Boolean.

CHAPTER 6

Exporter Object

The Exporter Object allows you to access and modify the render queue.

`app.getExporter()`

6.1 Attributes

6.1.1 Exporter.encodeID

`app.getExporter().encodeID`

Gets the current Encode ID.

Type

String; read-only.

6.1.2 Exporter.result

`app.getExporter().result`

unknown

Type

Boolean; read-only.

6.1.3 Exporter.elapsedMilliseconds

`app.getExporter().elapsedMilliseconds`

Elapsed milliseconds since queue started.

Type

Number; read-only.

6.2 Methods

6.2.1 Exporter.removeAllBatchItems()

Removes all items from the render queue.

```
app.getExporter().removeAllBatchItems()
```

Parameters

None.

Returns

Result Boolean.

Note: Application will hang until process is complete.

6.2.2 Exporter.trimExportForSR()*

```
app.getExporter().trimExportForSR()
```
**untested*

Parameters

unknown

Returns

unknown

6.2.3 Exporter.ExportItem()*

```
app.getExporter().exportItem()
```
**untested*

Parameters

unknown

Returns

unknown

6.2.4 Exporter.ExportGroup()*

```
app.getExporter().exportGroup()
```
**untested*

Parameters

unknown

Returns

unknown

CHAPTER 7

Encoder Host Object

The Encoder Host Object allows you to control and query the render queue.

```
app.getEncoderHost()
```

7.1 Methods

7.1.1 EncoderHost.runBatch()

Starts the render queue.

```
app.getEncoderHost().runBatch()
```

Parameters

None.

Returns

Result Boolean.

7.1.2 EncoderHost.pauseBatch()

Pauses the render queue.

```
app.getEncoderHost().pauseBatch()
```

Parameters

None.

Returns

Result Boolean.

7.1.3 EncoderHost.stopBatch()

Stops the render queue.

`app.getEncoderHost().stopBatch()`

Parameters

None.

Returns

Result Boolean.

7.1.4 EncoderHost.isBatchRunning()

Checks if the render queue is rendering.

`app.getEncoderHost().isBatchRunning()`

Parameters

None.

Returns

Result Boolean.

7.1.5 EncoderHost.getFormatList()

Checks if the render queue is rendering.

`app.getEncoderHost().getFormatList()`

Parameters

None.

Returns

Array of Format Names as Strings.

7.1.6 EncoderHost.createEncoderForFormat()

Creates an Encoder for a given Format

`app.getEncoderHost().createEncoderForFormat(format)` **untested*

Parameters

format: String. format name (H.264, Quicktime, etc.)

Returns

Encoder Object. (encode from there with `encoder.encode(src,dst)`)

7.1.7 EncoderHost.createMediaComparator()*

`app.getEncoderHost().createMediaComparator()` **untested*

Parameters

unknown

Returns

unknown

7.1.8 EncoderHost.getSupportedImportFileTypes()*

`app.getEncoderHost().getSupportedImportFileTypes()` **untested*

Parameters

unknown

Returns

unknown

7.1.9 EncoderHost.getSourceInfo()*

`app.getEncoderHost().getSourceInfo()` **untested*

Parameters

unknown

Returns

unknown

7.1.10 EncoderHost.getCurrentBatchPreview()*

`app.getEncoderHost().getCurrentBatchPreview()` **untested*

Parameters

unknown

Returns

unknown

CHAPTER 8

Batch Item Object

The Batch Item Object is returned after adding an item to the render queue, and then allows you to make further modifications and track the status of the render job.

More on adding items to the queue with Front End Object

8.1 Attributes

8.1.1 BatchItem.outputFiles

`batchItem.outputFiles`

Contains the output files from the render job once the job is complete.

Type

Array; read-only.

8.1.2 BatchItem.encodeProgress

`batchItem.encodeProgress`

The encode progress from 0 - 100 of render job.

Type

Number; read-only.

8.1.3 BatchItem.outputWidth*

`batchItem.outputWidth` **untested*

unknown

Type

Number;

8.1.4 BatchItem.outputHeight*

`batchItem.outputHeight` **untested*

unknown

Type

Number;

8.2 Methods

8.2.1 BatchItem.getEncodeProgress()

Gets the progress from 0 - 100 of the render job.

`batchItem.getEncodeProgress()`

Parameters

None.

Returns

Number.

8.2.2 BatchItem.getEncodeTime()*

`batchItem.getEncodeTime()` **untested*

Parameters

None.

Returns

Number.

8.2.3 BatchItem.getCurrentBatchPreview()*

`batchItem.getCurrentBatchPreview()` **untested*

Parameters

unknown

Returns

unknown

8.2.4 BatchItem.getPresetList()*

`batchItem.getPresetList()` **untested*

Parameters

unknown

Returns

unknown

8.2.5 BatchItem.getMissingAssets()*

`batchItem.getMissingAssets()` **untested*

Parameters

unknown

Returns

unknown

8.2.6 BatchItem.loadPreset()*

`batchItem.loadPreset()` **untested*

Parameters

unknown

Returns

unknown

8.2.7 BatchItem.loadFormat()*

`batchItem.loadFormat()` **untested*

Parameters

unknown

Returns

unknown

8.2.8 BatchItem.setWorkArea()*

`batchItem.setWorkArea()` **untested*

Parameters

unknown

Returns

unknown

8.2.9 BatchItem.setUsePreviewFiles()*

`batchItem.setUsePreviewFiles()` **untested*

Parameters

unknown

Returns

unknown

8.2.10 BatchItem.setUseMaximumRenderQuality()*

`batchItem.setUseMaximumRenderQuality()` **untested*

Parameters

unknown

Returns

unknown

8.2.11 BatchItem.setUseFrameBlending()*

`batchItem.setUseFrameBlending()` **untested*

Parameters

unknown

Returns

unknown

8.2.12 BatchItem.setIncludeSourceXMP()*

`batchItem.setIncludeSourceXMP()` **untested*

Parameters

unknown

Returns

unknown

8.2.13 BatchItem.setIncludeSourceCuePoints()*

`batchItem.setIncludeSourceCuePoints()` **untested*

Parameters

unknown

Returns

unknown

8.2.14 BatchItem.setCropState()*

`batchItem.setCropState()` **untested*

Parameters

unknown

Returns

unknown

8.2.15 BatchItem.setCropType()*

`batchItem.setCropType()` **untested*

Parameters

unknown

Returns

unknown

8.2.16 BatchItem.setCropOffsets()*

`batchItem.setCropOffsets()` **untested*

Parameters

unknown

Returns

unknown

8.2.17 BatchItem.setOutputFrameSize()*

`batchItem.setOutputFrameSize()` **untested*

Parameters

unknown

Returns

unknown

8.2.18 BatchItem.setXMPData()*

`batchItem.setXMPData()` **untested*

Parameters

unknown

Returns

unknown

8.2.19 BatchItem.setCuePointData()*

`batchItem.setCuePointData()` **untested*

Parameters

unknown

Returns

unknown

8.2.20 BatchItem.setScaleType()*

`batchItem.setScaleType()` **untested*

Parameters

unknown

Returns

unknown