CloudTest
by SOASTA

CloudTest Script Guide

SOASTA CloudTest Script Guide

Table of Contents

About SOASTA CloudTest™ Scripts

SOASTA CloudTest is a revolutionary suite of visual software products designed to automate load, performance, and functional testing for web applications—for use by any developer, sales engineer, business analyst, or tester.

Via its scripting support, SOASTA CloudTest's design also includes the ability to access the underlying message complexity both by the Message Editor and by extension using JavaScript objects. User created test scenarios can be further enhanced by the use of scripts. SOASTA CloudTest provides a simple scripting environment to use JavaScript to traverse and modify the SOASTA CloudTest Object Model.

Scripts can be positioned within Test Clips just like messages and browser actions, and incorporated into Test Compositions one or many times. All test clips, including any Scripts they contain, are reusable.

Note: As a convenience, an API reference is included below. However, the authoritative Script API Reference for CloudTest is generated with each build and is posted as Script API Reference (Latest Build) on CloudLink. When in doubt, refer to the CloudLink API Reference as the final authority.

For a build specific API reference, refer to the Documentation page and locate your build in the Release Notes list (the build-specific API Reference is posted for each build in the row that pertains to that build).

JavaScript Support

SOASTA CloudTest currently supports JavaScript. Mozilla's "Rhino" embeddable JavaScript language is used. Since JavaScript is a general language many things not supported declaratively by SOASTA CloudTest are possible via Scripts.

About This Guide

This Script Guide presents some common, step-by-step procedures for using scripts in Test Clips as part of Test Compositions. This guide also presents a complete SOASTA CloudTest scripting reference (see the Reference section at the end of this guide).

Quick Overview

Scripts can play an important role in the efficient use of Test Compositions. The scripts themselves, like the rest of test elements within SOASTA CloudTest, reside within test clips, and offer an under-the-hood means of testing complex web sites and applications.

CloudTest Architecture

In SOASTA CloudTest, messages, browser actions, scripts, checkpoints and other elements are combined into Test Clips that correspond to an individual test case. Test clips are then combined into multi-track test compositions.

Test messages are built to the schema of the given target service being tested, such as a web service, web site, or queuing system.

The following terms have specific meaning within the system:

- **Targets**

  Targets are defined items in the SOASTA Repository that contain a specification of the information needed to properly format and send messages to a web service or web application.

- **Messages**

  Modern composite web applications accomplish work by sending and receiving messages.

- **Browser Actions**

  A browser action is any user event that takes places within a web user interface. Browser actions are particularly important in the context of today's Rich Internet Applications (RIAs) based on Ajax. SOASTA CloudTest captures the user's browser actions, such as click, type, mouse down, mouse up, mouse over, select, drag and drop, and submit.

- **Test Clips**

  A clip is a collection of messages. SOASTA recommends that clips correspond to the individual unit test in the test scenario. For example, a set of boundary test cases for a SOAP "Add Customer" operation in a CRM web service.

- **Test Compositions**

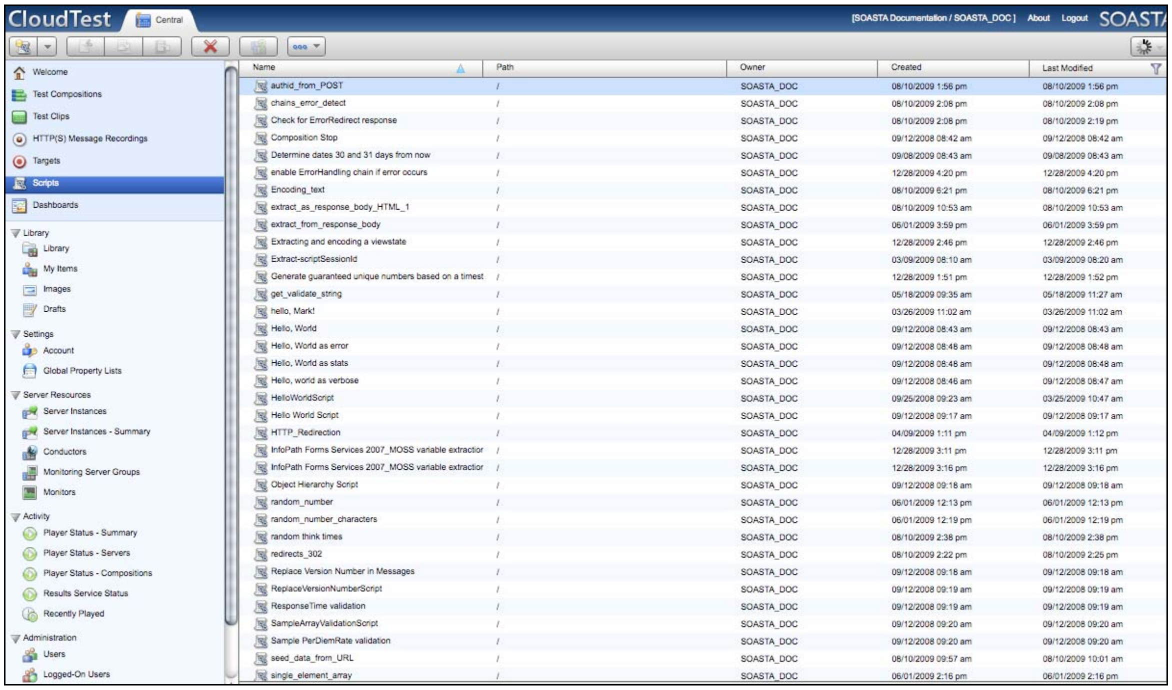
  The composition is the test. A composition contains test clips arranged on tracks and governed by user-specified sequence and tempo. Compositions are created, revised, and played within the Composition Editor.

- **Scripts**

  A script specifies a set of actions to be taken at a specific point in the execution of a test composition. For example, a script that modifies the input data for a message based upon a set of status codes from the previous message response.

Scripts in SOASTA CloudTest

The Central list > Scripts page shows currently available scripts.



Name	Path	Owner	Created	Last Modified
authid_from_POST	/	SOASTA_DOC	08/10/2009 1:56 pm	08/10/2009 1:56 pm
chains_error_detect	/	SOASTA_DOC	08/10/2009 2:08 pm	08/10/2009 2:08 pm
Check for ErrorRedirect response	/	SOASTA_DOC	08/10/2009 2:08 pm	08/10/2009 2:19 pm
Composition Stop	/	SOASTA_DOC	09/12/2008 08:42 am	09/12/2008 08:42 am
Determine dates 30 and 31 days from now	/	SOASTA_DOC	09/08/2009 08:43 am	09/08/2009 08:43 am
enable ErrorHandling chain if error occurs	/	SOASTA_DOC	12/28/2009 4:20 pm	12/28/2009 4:20 pm
Encoding_text	/	SOASTA_DOC	08/10/2009 6:21 pm	08/10/2009 6:21 pm
extract_as_response_body_HTML_1	/	SOASTA_DOC	08/10/2009 10:53 am	08/10/2009 10:53 am
extract_from_response_body	/	SOASTA_DOC	06/01/2009 3:59 pm	06/01/2009 3:59 pm
Extracting and encoding a viewstate	/	SOASTA_DOC	12/28/2009 2:46 pm	12/28/2009 2:46 pm
Extract-scriptSessionId	/	SOASTA_DOC	03/09/2009 08:10 am	03/09/2009 08:20 am
Generate guaranteed unique numbers based on a timest	/	SOASTA_DOC	12/28/2009 1:51 pm	12/28/2009 1:52 pm
get_validate_string	/	SOASTA_DOC	05/18/2009 09:35 am	05/18/2009 11:27 am
hello, Mark!	/	SOASTA_DOC	03/26/2009 11:02 am	03/26/2009 11:02 am
Hello, World	/	SOASTA_DOC	09/12/2008 08:43 am	09/12/2008 08:43 am
Hello, World as error	/	SOASTA_DOC	09/12/2008 08:48 am	09/12/2008 08:48 am
Hello, World as stats	/	SOASTA_DOC	09/12/2008 08:48 am	09/12/2008 08:48 am
Hello, world as verbose	/	SOASTA_DOC	09/12/2008 08:46 am	09/12/2008 08:47 am
HelloWorldScript	/	SOASTA_DOC	09/25/2008 09:23 am	03/25/2009 10:47 am
Hello World Script	/	SOASTA_DOC	09/12/2008 09:17 am	09/12/2008 09:17 am
HTTP_Redirection	/	SOASTA_DOC	04/09/2009 1:11 pm	04/09/2009 1:12 pm
InfoPath Forms Services 2007_MOSS variable extractior	/	SOASTA_DOC	12/28/2009 3:11 pm	12/28/2009 3:11 pm
InfoPath Forms Services 2007_MOSS variable extractior	/	SOASTA_DOC	12/28/2009 3:16 pm	12/28/2009 3:16 pm
Object Hierarchy Script	/	SOASTA_DOC	09/12/2008 09:18 am	09/12/2008 09:18 am
random_number	/	SOASTA_DOC	06/01/2009 12:13 pm	06/01/2009 12:13 pm
random_number_characters	/	SOASTA_DOC	06/01/2009 12:19 pm	06/01/2009 12:19 pm
random think times	/	SOASTA_DOC	08/10/2009 2:38 pm	08/10/2009 2:38 pm
redirects_302	/	SOASTA_DOC	08/10/2009 2:22 pm	08/10/2009 2:25 pm
Replace Version Number in Messages	/	SOASTA_DOC	09/12/2008 09:18 am	09/12/2008 09:18 am
ReplaceVersionNumberScript	/	SOASTA_DOC	09/12/2008 09:19 am	09/12/2008 09:19 am
ResponseTime validation	/	SOASTA_DOC	09/12/2008 09:19 am	09/12/2008 09:19 am
SampleArrayValidationScript	/	SOASTA_DOC	09/12/2008 09:20 am	09/12/2008 09:20 am
Sample PerDiemRate validation	/	SOASTA_DOC	09/12/2008 09:20 am	09/12/2008 09:20 am
seed_data_from_URL	/	SOASTA_DOC	08/10/2009 09:57 am	08/10/2009 10:01 am
single_element_array	/	SOASTA_DOC	06/01/2009 2:16 pm	06/01/2009 2:16 pm

Within the context of your Test Clip, a Script is a set of actions to be taken in-between two elements in the clip.

- If your script doesn't exist yet, with Central list > Scripts selected, click New. When you do so, the Script Editor opens with a blank page.

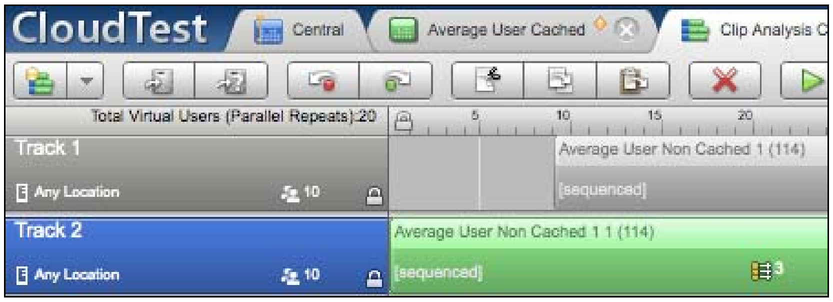
Once a script exists, it can be used within a Test Clip like any other SOASTA CloudTest object.

1. To add a script to a test clip, open an existing clip from the Central list > Test Clips page or click New to create a new Test Clip. The Clip Editor appears.

2. Once your test clip is open in the Test Clip Editor, click the Scripts tab in the lower pane and drag-and-drop your clip to add it into the new clip context.

Once your Test Clip(s) includes the necessary elements, including your script(s), you are ready to build a Test Composition using those test clips.

A Test Composition contains one or more Test Clips that execute when played.



- Use Central list > Test Compositions > New to build a new Test Composition, or open an existing Test Composition by double-clicking it in the Test Compositions list.

- Once your Test Composition is open in the Test Composition Editor, locate your test clip in the list below and then double-click or drag-and-drop it into the provided workspace.

Script creation can be automated using the Session Template Package Wizard and/or Session Template Wizard. For more information, refer to the Session Template Packages and Session Templates topics in Find Out How.

SOASTA CloudTest Object Model

In a web browser, the Document Object Model (DOM) is represented by a tree structure whose nodes can be navigated and then manipulated by a programmer using scripts. Unlike the DOM, where added complexity can make locating a specific object in the model a challenge, the object model within SOASTA CloudTest is relatively simple. As with the DOM, a programmer can use scripts to manipulate objects.

The SOASTA CloudTest Object Model contains pre-defined JavaScript objects to which every script has access. Together these objects make up an "object model" that allows access to the Test Composition and to all of its component parts.

Within this object model, the Test Composition is the root element, while other Composition objects also have an object in the model.

JavaScript Extensions

The SOASTA CloudTest Object Model is presented via JavaScript extensions that allow user-defined scripts to access and modify the currently running Test Composition. Scripts are added as elements within one or more Test Clips within that test composition.

These JavaScript extensions offer an object model variable via *($context*), as well as direct manipulation of properties by variables *($prop*, *$sysprop*, and *$globalprop*).

Note: SOASTA CloudTest offers three property variables ($prop, $sysprop, and $globalprop) that simplify setting composition, system, and global properties.

SOASTA CloudTest Objects

Since scripts are used in the same context in Test Clips as any other element—and Test Clips are contained by Test Compositions within Tracks—the resulting object model is fairly simple.

The following objects are contained in the object model and can be accessed in scripts using the $context variable.

- **composition**

  Specifies the current composition. For example,

  ```
  $context.composition.stop()
  ```

  A test composition is the root element in the SOASTA CloudTest Object Model and may contain some or all of the remaining objects.

  Properties*: Name, Parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem*

  Methods: *abort*, *stop*, *getChild*, and *getItemViaPath*

- **currentBand**

  Specifies the current Band according to context. Null if there is no band.

  The band is a container for track, and as a result, everything else in the object model may be contained within a band. For example,

  ```
  $context.currentBand.getChild()
  ```

  Properties*: name, parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT*, and *REPEAT_DISTRIBUTION_CONSTANT*

  Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

- **currentTrack**

  Specifies the current track according to context. A track may contain one or more clips, which may in turn contain messages, checkpoints, chains, and scripts. Null if there is no current track.

  Properties*: name, parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT*, and *REPEAT_DISTRIBUTION_CONSTANT*

  Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

- **currentClip**

  Specifies the current Test Clip according to context. The test clip may contain messages, checkpoints, and scripts. Null if there is no current clip.

  Properties*: name, parent*, *propertyList*, systemPropertyList, *type*, *children, index, nextItem, previousItem, targets, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT,* and *REPEAT_DISTRIBUTION_CONSTANT*

  Methods: *getChild, getTarget, getItemViaPath, clearRepeat, setRepeat*

- **currentChain**

  Specifies the current chain. A chain is a grouping of messages, scripts, or checkpoints within the currentClip. This grouping may be used to control the timing of the chain members. Null if there is no current chain.

  Properties*: name, parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT*, and *REPEAT_DISTRIBUTION_CONSTANT*

  Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

- **currentMessage**

  Specifies the current message. The message is the basic unit of Test Compositions. Null if there is no current message.

  Properties*: name*, *parent*, *propertyList*, *systemPropertyList*, *type*, *children, index, nextItem, previousItem, target, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT, REPEAT_DISTRIBUTION_CONSTANT, responseTime, bytesSentCount, bytesReceivedCount, retryCount, retryTotalTime*

  Methods: *getChild, getItemViaPath, getResponse, clearRepeat, setRepeat, getMessage, setMessage*

- **currentBrowserAction**

  Specifies the current Browser Action according to the context in which the current script is executing. Null if there is no current Browser Action.

  Properties*: name*, *parent*, *propertyList*, *systemPropertyList*, *type*, *children, target, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT,* and *REPEAT_DISTRIBUTION_CONSTANT*

Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

- **currentCheckpoint**

  Specifies the current checkpoint. The checkpoint is a way of imposing a dependency, such as stopping action on whatever comes after it until an expected response is received.

  Properties*:* *name, parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT*, and *REPEAT_DISTRIBUTION_CONSTANT*

  Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

- **currentScript**

  Specifies the current script. A script is another term for a script. Null if there is no current script.

  Properties*:* *name, parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT*, and *REPEAT_DISTRIBUTION_CONSTANT*

  Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

- **currentTarget**

  Specifies the current target. The target is the URL or destination of messages within the clip.

  Properties*:* *name, parent, propertyList, systemPropertyList, type, children, index, nextItem, and previousItem*

  Methods: *getChild, getItemViaPath*

- **currentDelay**

  Specifies the current delay according to the context in which the current script is executing. Null if there is no current Delay component.

  Properties*:* *name, parent, propertyList, systemPropertyList, type, children, index, nextItem, previousItem, REPEAT_TIMING_PARALLEL, REPEAT_TIMING_SERIAL, REPEAT_TYPE_COUNT_CONSTANT*, and *REPEAT_DISTRIBUTION_CONSTANT*

Methods: *getChild, getItemViaPath, clearRepeat, setRepeat*

Result Object

The Result object is a special case within the SOASTA CloudTest Object Model since results are a byproduct of "playing" a test composition that are not "parts' of that composition. The $context variable includes results just as it does any of the objects that occur within a composition. For example,

```
$context.result.postMessage($context.result.LEVEL_INFO, "hello, world");
```

The above line of code uses the postMessage method to write a message to the result for the current composition.

You can post messages to the result object for the current context using the following properties: LEVEL_ERROR, LEVEL_STATISTICS, LEVEL_INFO, and LEVEL_VERBOSE.

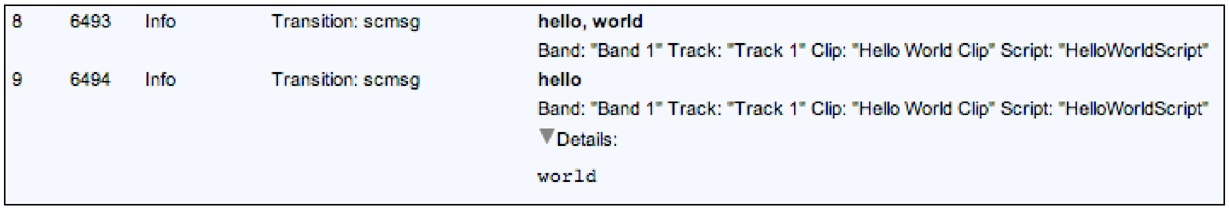
Send Hello, World to a Result

CloudTest scripts can be quite simple, as the following example demonstrates. Other scripts within this guide deal with more complex issues such as redirection, or extraction, encoding, and unencoding.

The following line of code placed into a Test Clip (and subsequently a Test Composition) provides a quick sense of the power of SOASTA CloudTest Extensions, as well as of the special status of the Result object within the object model:

```
$context.result.postMessage($context.result.LEVEL_INFO, "hello, world");
—Or—
$context.result.postMessage($context.result.LEVEL_INFO, "hello", "world");
```

The first line of code posts a text string to the result object for the current composition. The second line posts the same string as well as a detail message about the execution of that line as shown below.



```
8    6493   Info    Transition: scmsg    hello, world
                                         Band: "Band 1" Track: "Track 1" Clip: "Hello World Clip" Script: "HelloWorldScript"
9    6494   Info    Transition: scmsg    hello
                                         Band: "Band 1" Track: "Track 1" Clip: "Hello World Clip" Script: "HelloWorldScript"
                                         ▼Details:
                                         world
```

Results are accessed via Central list > Recently Played and include a variety of informational postings about played Test Compositions).

SOASTA CloudTest also includes the constants LEVEL_VERBOSE, LEVEL_STATISTICS, and LEVEL_ERROR. For example,

```
$context.result.postMessage ($context.result.LEVEL_VERBOSE, "hello, world
[as 'verbose']");
```

```
$context.result.postMessage ($context.result.LEVEL_STATISTICS, "hello, world
[as 'statistics']");

$context.result.postMessage ($context.result.LEVEL_ERROR, "hello, world [as
'error']");
```

Stop a Composition from a Script

Stopping a composition using a script is a simple, one-line:

$context.composition.stop();

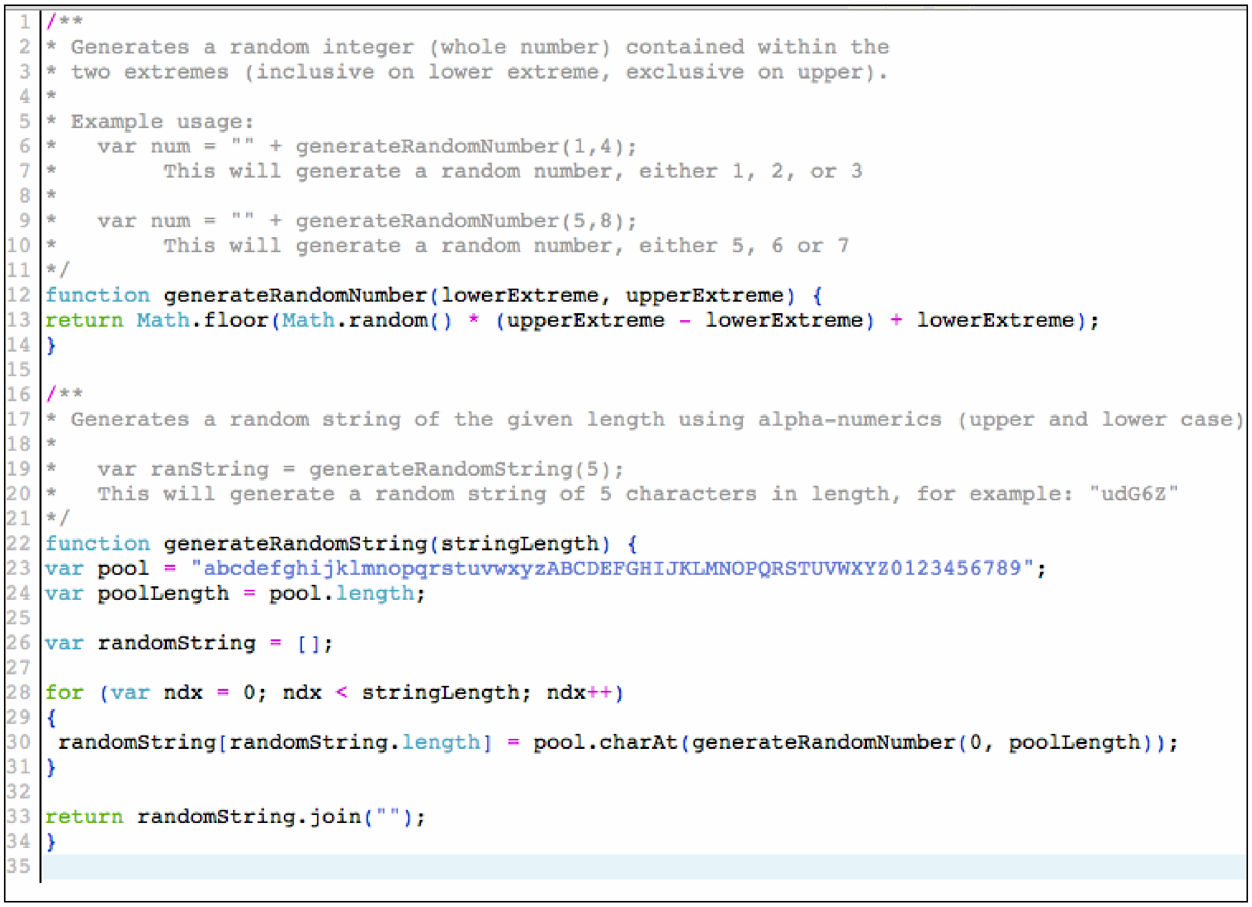
Stopping a composition can be used to conditionally terminate a composition on the basis of some error.

Data Generation Scripts

This section presents a variety of SOASTA CloudTest data generation scripts. Click the link for each script below to view it in online. Online scripts can be copied and pasted directly into the SOASTA CloudTest Script Editor using Central > Scripts. Shorter scripts are presented inline.

Script 1: Random Number/String Generator

The Random number/string generator script will generate a random number and place it into a test clip.



```
/**
* Generates a random integer (whole number) contained within the
* two extremes (inclusive on lower extreme, exclusive on upper).
*
* Example usage:
*   var num = "" + generateRandomNumber(1,4);
*        This will generate a random number, either 1, 2, or 3
*
*   var num = "" + generateRandomNumber(5,8);
*        This will generate a random number, either 5, 6 or 7
*/
function generateRandomNumber(lowerExtreme, upperExtreme) {
return Math.floor(Math.random() * (upperExtreme - lowerExtreme) + lowerExtreme);
}

/**
* Generates a random string of the given length using alpha-numerics (upper and lower case)
*
*   var ranString = generateRandomString(5);
*   This will generate a random string of 5 characters in length, for example: "udG6Z"
*/
function generateRandomString(stringLength) {
var pool = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
var poolLength = pool.length;

var randomString = [];

for (var ndx = 0; ndx < stringLength; ndx++)
{
 randomString[randomString.length] = pool.charAt(generateRandomNumber(0, poolLength));
}

return randomString.join("");
}
```

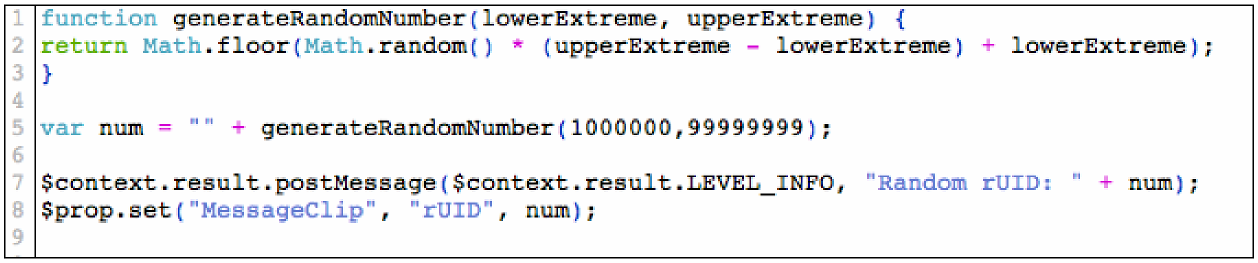
The function to the right generates a random integer (whole number) contained within the two extremes (inclusive on lower extreme, exclusive on upper).	`function generateRandomNumber(lowerExtreme, upperExtreme) {`
	`return Math.floor(Math.random() * (upperExtreme - lowerExtreme)`
	`+ lowerExtreme);`
	`}`

For example, generate a random number, either 1, 2, or 3:	`var num = "" + generateRandomNumber(1,4);`
Generate a random number either 5, 6 or 7:	`var num = "" + generateRandomNumber(5,8);`
Generate a random string of the given length using alpha-numerics (upper and lower case).	`var ranString = generateRandomString(5);`

The next example generates a random string of five characters in length, such as: "udG6Z"	`function generateRandomString(stringLength) {`
	`var pool =`
	`"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"`
	`;`
	`var poolLength = pool.length;`
	
	`var randomString = [];`
	
	`for (var ndx = 0; ndx < stringLength; ndx++)`
	`{`
	`randomString[randomString.length] =`
	`pool.charAt(generateRandomNumber(0, poolLength));`
	`}`
	
	`return randomString.join("");`
	`}`

Script 2: Generate a Random Number of 7 or 8 Digits

The Random number/string generator with seven-eight characters script will generate a random number with seven or eight characters and place it into a test clip.



```
1 function generateRandomNumber(lowerExtreme, upperExtreme) {
2 return Math.floor(Math.random() * (upperExtreme - lowerExtreme) + lowerExtreme);
3 }
4
5 var num = "" + generateRandomNumber(1000000,99999999);
6
7 $context.result.postMessage($context.result.LEVEL_INFO, "Random rUID: " + num);
8 $prop.set("MessageClip", "rUID", num);
9
```

Lines 1-3 create a function to generate a random number.	`function generateRandomNumber(lowerExtreme, upperExtreme) {`
	`return Math.floor(Math.random() * (upperExtreme - lowerExtreme)`
	`+ lowerExtreme);`
	`}`

Line 5 creates a variable, num, to store the random number, which will contain seven to eight characters.	`var num = "" + generateRandomNumber(1000000,99999999);`

Lines 7-8 place the random number into the current test clip context as a property.	`$context.result.postMessage($context.result.LEVEL_INFO, "Random`
	`rUID: " + num);`
	`$prop.set("MessageClip", "rUID", num);`

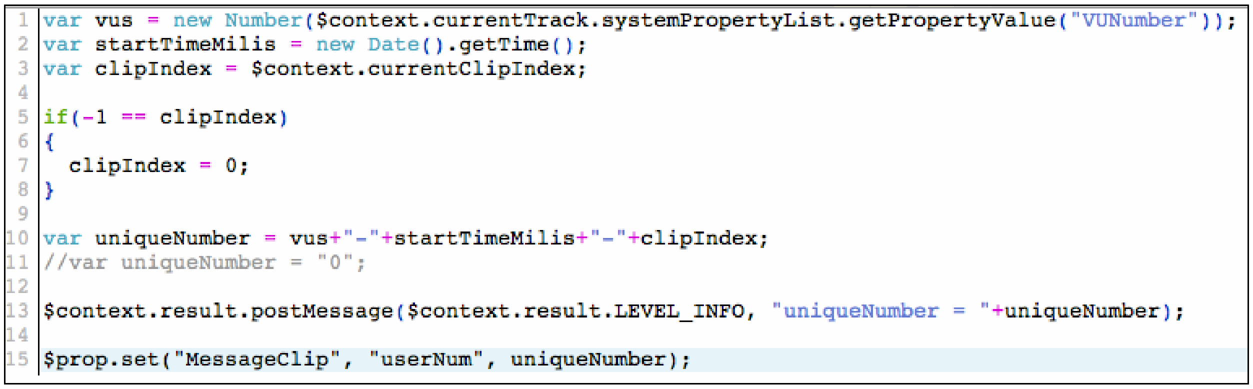
Script 3: Using Properties to Generate Unique Data

This simple script uses established CloudTest system properties to generate unique data:

```
{%%Sys-Prop:Composition:StartTimeMillis%%}-{%%Sys-Prop:Track:VUNumber%%}-{%%
expr: $context.currentClipIndex %%}
```

Script 4: Generate Guaranteed Unique Numbers Based on a Timestamp

The Generate guaranteed unique numbers based on a timestamp script will generate unique numbers based on a timestamp. This script is used with an established System Property, VUNumber, which corresponds to the number of virtual users in a test composition.

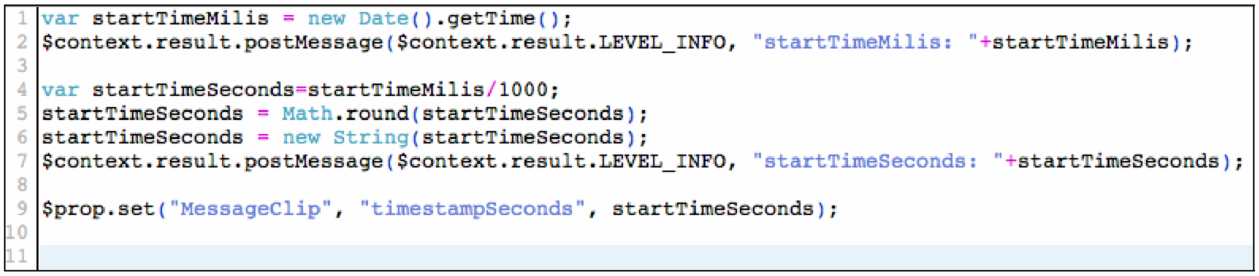


```
1  var vus = new Number($context.currentTrack.systemPropertyList.getPropertyValue("VUNumber"));
2  var startTimeMilis = new Date().getTime();
3  var clipIndex = $context.currentClipIndex;
4  
5  if(-1 == clipIndex)
6  {
7    clipIndex = 0;
8  }
9  
10 var uniqueNumber = vus+"-"+startTimeMilis+"-"+clipIndex;
11 //var uniqueNumber = "0";
12 
13 $context.result.postMessage($context.result.LEVEL_INFO, "uniqueNumber = "+uniqueNumber);
14 
15 $prop.set("MessageClip", "userNum", uniqueNumber);
```

Use a System Property in conjunction with this script and virtual users to generate a unique timestamp-based user number.	`var vus = new Number($context.currentTrack.systemPropertyList.getPropertyValue ("VUNumber"));` `var startTimeMilis = new Date().getTime();` `var clipIndex = $context.currentClipIndex;` `if(-1 == clipIndex)` `{` `clipIndex = 0;` `}`
Declare a variable, uniqueNumber, to store the number created in Lines 1-8 above.	`var uniqueNumber = vus+"-"+startTimeMilis+"-"+clipIndex;` `//var uniqueNumber = "0";`
Post a message to results and set a clip property, userNum, to store the uniqueNumber per user.	`$context.result.postMessage($context.result.LEVEL_INFO, "uniqueNumber = "+uniqueNumber);` `$prop.set("MessageClip", "userNum", uniqueNumber);`

Script 5: Generate Current Timestamp in Milliseconds or Seconds

The Generate current timestamp in milliseconds or seconds script guarantees a timestamp in milliseconds or seconds:



```
 1 var startTimeMilis = new Date().getTime();
 2 $context.result.postMessage($context.result.LEVEL_INFO, "startTimeMilis: "+startTimeMilis);
 3
 4 var startTimeSeconds=startTimeMilis/1000;
 5 startTimeSeconds = Math.round(startTimeSeconds);
 6 startTimeSeconds = new String(startTimeSeconds);
 7 $context.result.postMessage($context.result.LEVEL_INFO, "startTimeSeconds: "+startTimeSeconds);
 8
 9 $prop.set("MessageClip", "timestampSeconds", startTimeSeconds);
10
11
```

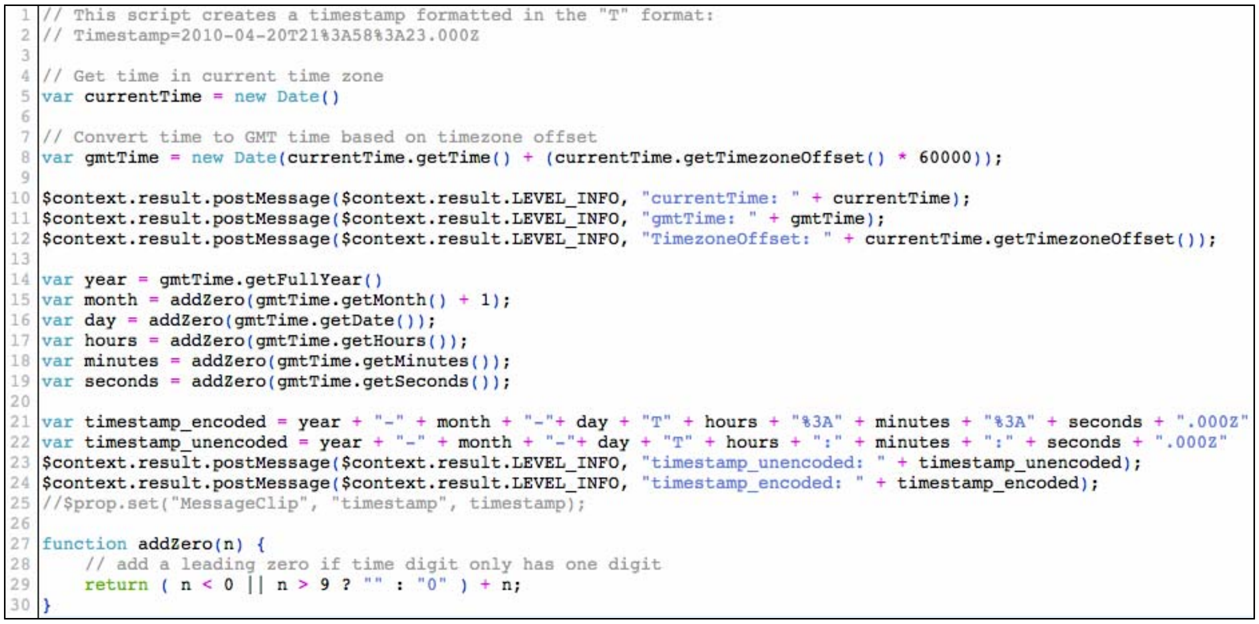
Lines 1-2 declare a variable for "start time" and gets the current date/time stamp in milliseconds and then Line 4 posts it to a message	`var startTimeMilis = new Date().getTime();`
	`$context.result.postMessage($context.result.LEVEL_INFO, "startTimeMilis: "+startTimeMilis);`
	
	`$prop.set("MessageClip", "timestampSeconds", startTimeSeconds);`

Lines 3-6 declare a variable, startTimeSeconds, which equals 1000 ms and gives it the necessary math. Line 7 posts a message to results.	`var startTimeSeconds=startTimeMilis/1000;`
	`startTimeSeconds = Math.round(startTimeSeconds);`
	`startTimeSeconds = new String(startTimeSeconds);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "startTimeSeconds: "+startTimeSeconds);`

Finally, set a property for the custom clip property and its value.	`$prop.set("MessageClip", "timestampSeconds", startTimeSeconds);`

Script 6: Generate Current Timestamp in "T" Format

The Generate current timestamp in "T" format script creates a timestamp formatted in the "T" format: Timestamp=2010-04-20T21%3A58%3A23.000Z



```
 1 // This script creates a timestamp formatted in the "T" format:
 2 // Timestamp=2010-04-20T21%3A58%3A23.000Z
 3 
 4 // Get time in current time zone
 5 var currentTime = new Date()
 6 
 7 // Convert time to GMT time based on timezone offset
 8 var gmtTime = new Date(currentTime.getTime() + (currentTime.getTimezoneOffset() * 60000));
 9 
10 $context.result.postMessage($context.result.LEVEL_INFO, "currentTime: " + currentTime);
11 $context.result.postMessage($context.result.LEVEL_INFO, "gmtTime: " + gmtTime);
12 $context.result.postMessage($context.result.LEVEL_INFO, "TimezoneOffset: " + currentTime.getTimezoneOffset());
13 
14 var year = gmtTime.getFullYear()
15 var month = addZero(gmtTime.getMonth() + 1);
16 var day = addZero(gmtTime.getDate());
17 var hours = addZero(gmtTime.getHours());
18 var minutes = addZero(gmtTime.getMinutes());
19 var seconds = addZero(gmtTime.getSeconds());
20 
21 var timestamp_encoded = year + "-" + month + "-"+ day + "T" + hours + "%3A" + minutes + "%3A" + seconds + ".000Z"
22 var timestamp_unencoded = year + "-" + month + "-"+ day + "T" + hours + ":" + minutes + ":" + seconds + ".000Z"
23 $context.result.postMessage($context.result.LEVEL_INFO, "timestamp_unencoded: " + timestamp_unencoded);
24 $context.result.postMessage($context.result.LEVEL_INFO, "timestamp_encoded: " + timestamp_encoded);
25 //$prop.set("MessageClip", "timestamp", timestamp);
26 
27 function addZero(n) {
28     // add a leading zero if time digit only has one digit
29     return ( n < 0 || n > 9 ? "" : "0" ) + n;
30 }
```

Line 1 gets the time in the current time zone and then

Line 8 declares a variable for the time in Greenwich Mean Time and calculates the difference based on the offset.

Lines 10-13 post these facts to results

Lines 14-19 declare variables for each unit of time descending from year to seconds.

```
var currentTime = new Date()

// Convert time to GMT time based on timezone offset
var gmtTime = new Date(currentTime.getTime() +
(currentTime.getTimezoneOffset() * 60000));

$context.result.postMessage($context.result.LEVEL_INFO,
"currentTime: " + currentTime);
$context.result.postMessage($context.result.LEVEL_INFO,
"gmtTime: " + gmtTime);
$context.result.postMessage($context.result.LEVEL_INFO,
"TimezoneOffset: " + currentTime.getTimezoneOffset());

var year = gmtTime.getFullYear()
var month = addZero(gmtTime.getMonth() + 1);
var day = addZero(gmtTime.getDate());
var hours = addZero(gmtTime.getHours());
var minutes = addZero(gmtTime.getMinutes());
var seconds = addZero(gmtTime.getSeconds());

var timestamp_encoded = year + "-" + month + "-"+ day + "T" +
hours + "%3A" + minutes + "%3A" + seconds + ".000Z"
var timestamp_unencoded = year + "-" + month + "-"+ day + "T" +
hours + ":" + minutes + ":" + seconds + ".000Z"
$context.result.postMessage($context.result.LEVEL_INFO,
"timestamp_unencoded: " + timestamp_unencoded);
$context.result.postMessage($context.result.LEVEL_INFO,
"timestamp_encoded: " + timestamp_encoded);
//$prop.set("MessageClip", "timestamp", timestamp);
function addZero(n) {
    // add a leading zero if time digit only has one digit
    return ( n < 0 || n > 9 ? "" : "0" ) + n;
}
```

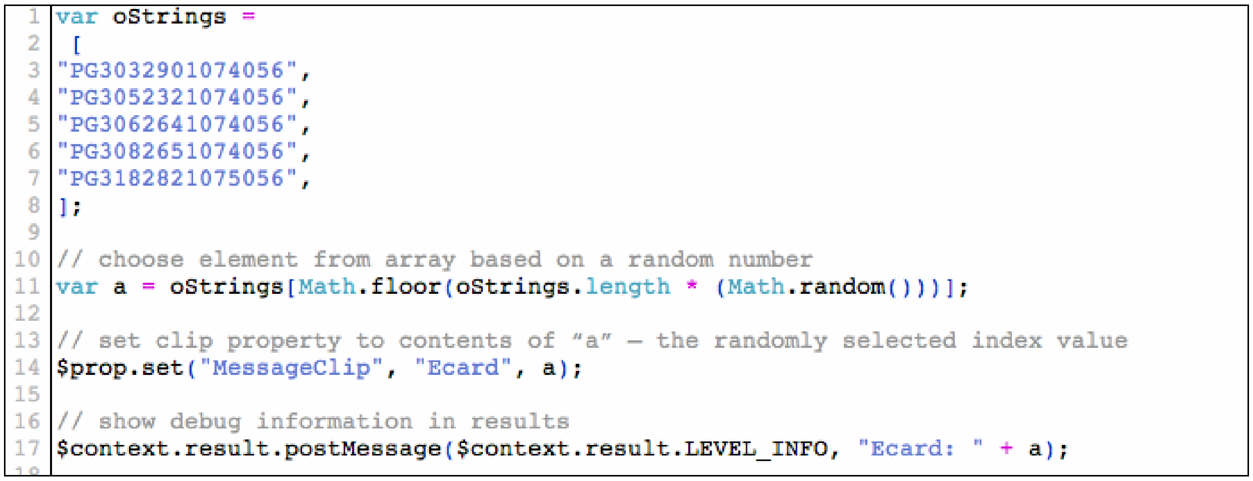
Lines 21-30 create the encoding for the “T” format.	

Data Seeding Scripts

This section presents a variety of SOASTA CloudTest data seeding scripts. Scripts can be copied and pasted directly into the SOASTA CloudTest Script Editor using Central > Scripts. Longer scripts have an accompanying link that can be clicked to pop out the full example script. Shorter scripts are presented inline.

Script 1: Single-element Data Array, Choose One Randomly

The Single-element data array, choose one randomly script creates an array and then chooses a random value from that array. See also: Using Dynamic Data > Array File.

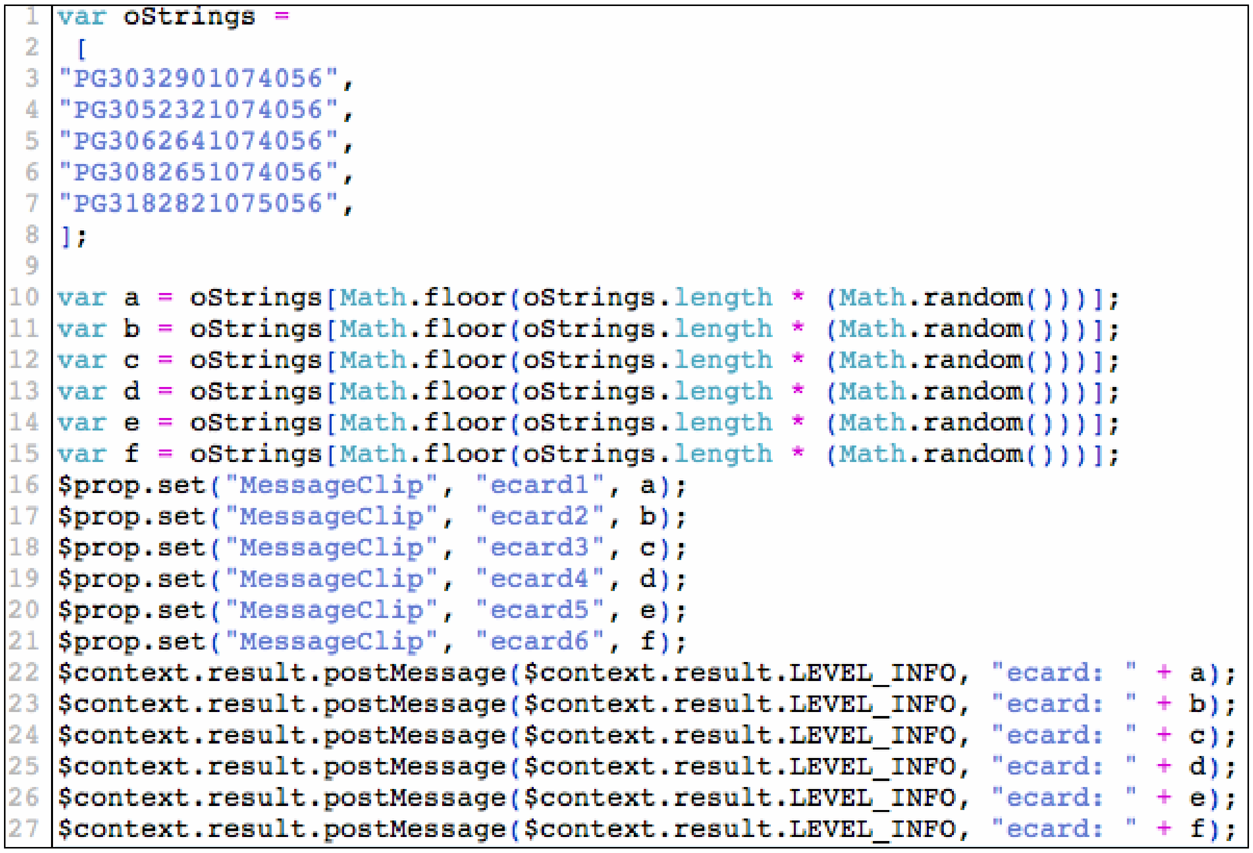


```
var oStrings =
 [
"PG3032901074056",
"PG3052321074056",
"PG3062641074056",
"PG3082651074056",
"PG3182821075056",
];

// choose element from array based on a random number
var a = oStrings[Math.floor(oStrings.length * (Math.random()))];

// set clip property to contents of "a" – the randomly selected index value
$prop.set("MessageClip", "Ecard", a);

// show debug information in results
$context.result.postMessage($context.result.LEVEL_INFO, "Ecard: " + a);
```

First, declare and format the array:	`var oStrings = [ "PG3032901074056", "PG3052321074056", "PG3062641074056", "PG3082651074056", "PG31`
Next, choose an element from the array based on a random number.	`var a = oStrings[Math.floor(oStrings.length * (Math.random()))];`
Set a clip property to contents of “a” – the randomly selected index value.	`$prop.set("MessageClip", "PropertyName", a);`
Show debug information in the results.	`$context.result.postMessage($context.result.LEVEL_INFO, "PropertyName: " + a);`

Script 2: Single-element Data Array, Choose Six Randomly

The Single-element data array, choose six randomly script creates a single-element array, and then chooses elements randomly from that array.



```
 1 var oStrings =
 2  [
 3 "PG3032901074056",
 4 "PG3052321074056",
 5 "PG3062641074056",
 6 "PG3082651074056",
 7 "PG3182821075056",
 8 ];
 9
10 var a = oStrings[Math.floor(oStrings.length * (Math.random()))];
11 var b = oStrings[Math.floor(oStrings.length * (Math.random()))];
12 var c = oStrings[Math.floor(oStrings.length * (Math.random()))];
13 var d = oStrings[Math.floor(oStrings.length * (Math.random()))];
14 var e = oStrings[Math.floor(oStrings.length * (Math.random()))];
15 var f = oStrings[Math.floor(oStrings.length * (Math.random()))];
16 $prop.set("MessageClip", "ecard1", a);
17 $prop.set("MessageClip", "ecard2", b);
18 $prop.set("MessageClip", "ecard3", c);
19 $prop.set("MessageClip", "ecard4", d);
20 $prop.set("MessageClip", "ecard5", e);
21 $prop.set("MessageClip", "ecard6", f);
22 $context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + a);
23 $context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + b);
24 $context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + c);
25 $context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + d);
26 $context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + e);
27 $context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + f);
```

Lines 1-8 create and format an array::	`var oStrings =`
	`[`
	`"PG3032901074056",`
	`"PG3052321074056",`
	`"PG3062641074056",`
	`"PG3082651074056",`
	`"PG3182821075056",`
	`];`

Next, lines 10-15 create six variables, a-f, to store six random numbers:	`var a = oStrings[Math.floor(oStrings.length * (Math.random()))];`
	`var b = oStrings[Math.floor(oStrings.length * (Math.random()))];`
	`var c = oStrings[Math.floor(oStrings.length * (Math.random()))];`
	`var d = oStrings[Math.floor(oStrings.length * (Math.random()))];`
	`var e = oStrings[Math.floor(oStrings.length * (Math.random()))];`
	`var f = oStrings[Math.floor(oStrings.length * (Math.random()))];`

Next, lines 16-21 populate clip properties with the random numbers by variable.	`$prop.set("MessageClip", "prop1", a);`
	`$prop.set("MessageClip", "prop2", b);`
	`$prop.set("MessageClip", "prop3", c);`
	`$prop.set("MessageClip", "prop4", d);`
	`$prop.set("MessageClip", "prop5", e);`
	`$prop.set("MessageClip", "prop6", f);`

Lines 22-27 set debug messages for each variable.	`$context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + a);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + b);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + c);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + d);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + e);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "ecard: " + f);`

Script 3: Single-element Data Array, Select Unique Values

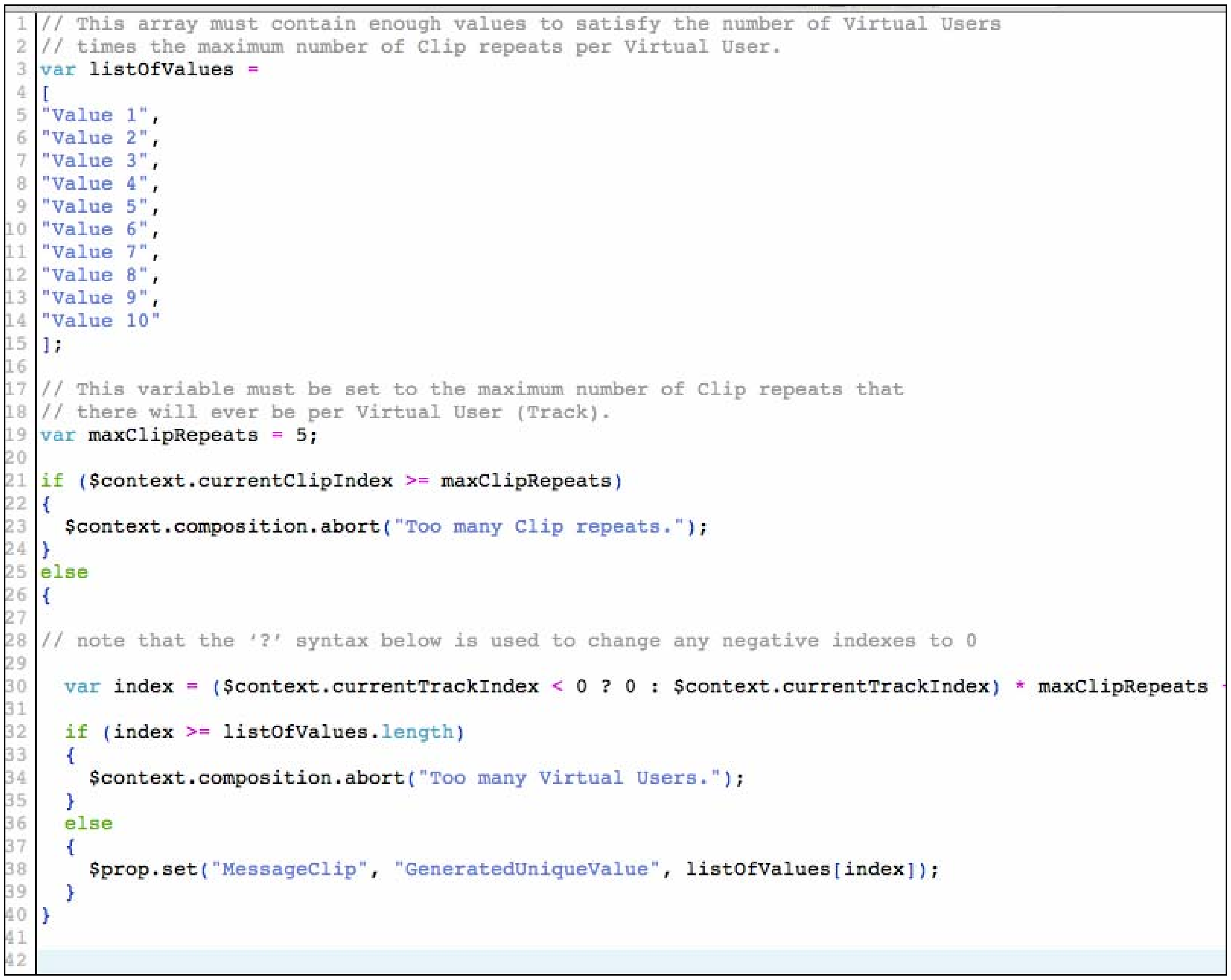
The Single-element data array, select unique values script shows how a script can be used to ensure that only a single value from an array is used by a clip/composition.

This script takes into account track and clip indexes to make sure that only unique values are used in the test. If the serial repeats, or the number of virtual users exceeds the number defined in the script, the composition will abort.

Note: There are two important restrictions:

(1) There must be enough values to handle the number of Virtual Users times the maximum number of repeats of the Clip within each Virtual User.

(2) The script must be set ahead of time to specify the maximum number of repeats of the Clip within each Virtual User (see the variable maxClipRepeats).



```
 1 // This array must contain enough values to satisfy the number of Virtual Users
 2 // times the maximum number of Clip repeats per Virtual User.
 3 var listOfValues =
 4 [
 5 "Value 1",
 6 "Value 2",
 7 "Value 3",
 8 "Value 4",
 9 "Value 5",
10 "Value 6",
11 "Value 7",
12 "Value 8",
13 "Value 9",
14 "Value 10"
15 ];
16
17 // This variable must be set to the maximum number of Clip repeats that
18 // there will ever be per Virtual User (Track).
19 var maxClipRepeats = 5;
20
21 if ($context.currentClipIndex >= maxClipRepeats)
22 {
23   $context.composition.abort("Too many Clip repeats.");
24 }
25 else
26 {
27
28 // note that the '?' syntax below is used to change any negative indexes to 0
29
30   var index = ($context.currentTrackIndex < 0 ? 0 : $context.currentTrackIndex) * maxClipRepeats
31
32   if (index >= listOfValues.length)
33   {
34     $context.composition.abort("Too many Virtual Users.");
35   }
36   else
37   {
38     $prop.set("MessageClip", "GeneratedUniqueValue", listOfValues[index]);
39   }
40 }
41
42
```

Lines 5-14 create and format an array::	`[`
	`"Value 1",`
	`"Value 2",`
	`"Value 3",`
	`"Value 4",`
	`"Value 5",`
	`"Value 6",`
	`"Value 7",`
	`"Value 8",`
	`"Value 9",`
	`"Value 10"`
	`];`

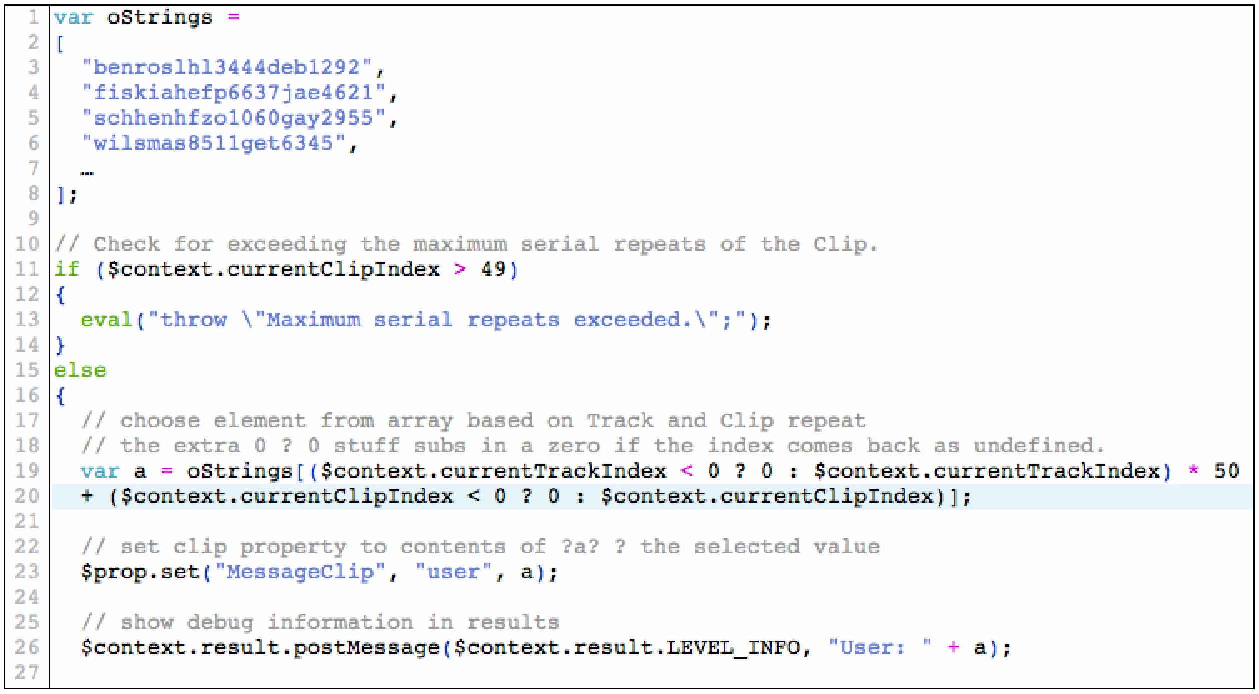
Set the maximum number of Clip repeats that there will ever be per Virtual User (Track) via the variable, `maxClipRepeats`.	`if ($context.currentClipIndex >= maxClipRepeats)`
	`{`
	`$context.composition.abort("Too many Clip repeats.");`
	`}`
	`else`
	`{`

Note that the '?' syntax on the right is used to change any negative indexes to 0.

```
var index = ($context.currentTrackIndex < 0 ? 0 :
$context.currentTrackIndex) * maxClipRepeats +
($context.currentClipIndex < 0 ? 0 : $context.currentClipIndex);

if (index >= listOfValues.length)
{
$context.composition.abort("Too many Virtual Users.");
   }
else
{
$prop.set("MessageClip", "GeneratedUniqueValue",
listOfValues[index]);
   }
 }
```

Script 4: Single-element Data Array, Select Unique Values – second example

The incrementing in this script is done based on track and clip indexes. This is necessary when each serial repeat of a clip needs a different element in the array. This example is different because it substitutes in zeros (0) instead of aborting the composition.



```
1  var oStrings =
2  [
3    "benroslhl3444deb1292",
4    "fiskiahefp6637jae4621",
5    "schhenhfzo1060gay2955",
6    "wilsmas8511get6345",
7    …
8  ];
9
10 // Check for exceeding the maximum serial repeats of the Clip.
11 if ($context.currentClipIndex > 49)
12 {
13   eval("throw \"Maximum serial repeats exceeded.\";");
14 }
15 else
16 {
17   // choose element from array based on Track and Clip repeat
18   // the extra 0 ? 0 stuff subs in a zero if the index comes back as undefined.
19   var a = oStrings[($context.currentTrackIndex < 0 ? 0 : $context.currentTrackIndex) * 50
20   + ($context.currentClipIndex < 0 ? 0 : $context.currentClipIndex)];
21
22   // set clip property to contents of ?a? ? the selected value
23   $prop.set("MessageClip", "user", a);
24
25   // show debug information in results
26   $context.result.postMessage($context.result.LEVEL_INFO, "User: " + a);
27
```

Lines 2 through 8 set up the data array `oStrings`.

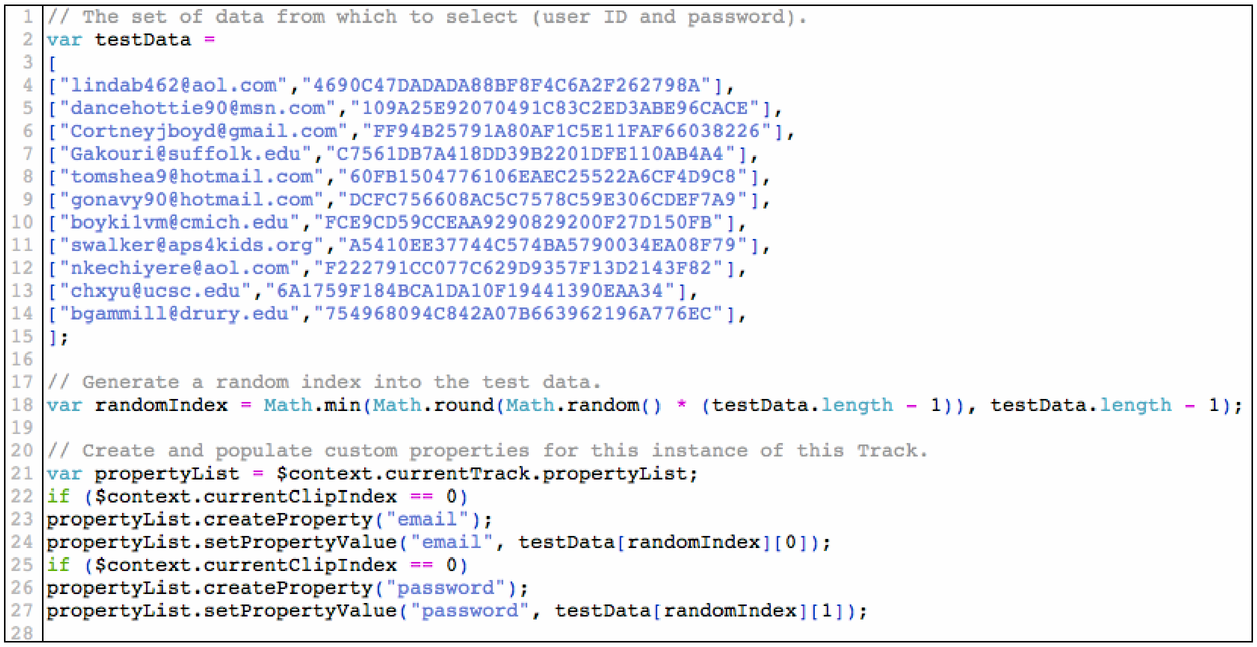
```
var oStrings =
[
"benroslhl3444deb1292",
"fiskiahefp6637jae4621",
"schhenhfzo1060gay2955",
"wilsmas8511get6345",
…
];
```

Next, check to ensure that the maximum serial repeats of the test clip are not exceeded.	`if ($context.currentClipIndex > 49)`
	`{`
	`   eval("throw \"Maximum serial repeats exceeded.\";");`
	` }`
	` else`
	` {`

Choose an element from the array based on the Track and Clip repeat. The extra 0 ? 0 code subs in a zero if the index comes back as undefined.	`var a = oStrings[($context.currentTrackIndex < 0 ? 0 : $context.currentTrackIndex) * 50 + ($context.currentClipIndex < 0 ? 0 : $context.currentClipIndex)];     /`
Set the clip property to the contents of ?a? ? the selected value	`$prop.set("MessageClip", "user", a);`
Show debug information in results.	`$context.result.postMessage($context.result.LEVEL_INFO, "User: " + a);  }`

Script 5: Two-element Data Array, Choose One Record Randomly

The Two-element data array, choose one record randomly script is run at the beginning of the clip. It creates two track properties behind the scenes (i.e. the track properties are not created in the composition editor). It loads each element of the array into its own property. This type of script is used when the first element of the array is related to the second element – as in username/password combinations.



```
// The set of data from which to select (user ID and password).
var testData =
[
["lindab462@aol.com","4690C47DADADA88BF8F4C6A2F262798A"],
["dancehottie90@msn.com","109A25E92070491C83C2ED3ABE96CACE"],
["Cortneyjboyd@gmail.com","FF94B25791A80AF1C5E11FAF66038226"],
["Gakouri@suffolk.edu","C7561DB7A418DD39B2201DFE110AB4A4"],
["tomshea9@hotmail.com","60FB1504776106EAEC25522A6CF4D9C8"],
["gonavy90@hotmail.com","DCFC756608AC5C7578C59E306CDEF7A9"],
["boyki1vm@cmich.edu","FCE9CD59CCEAA9290829200F27D150FB"],
["swalker@aps4kids.org","A5410EE37744C574BA5790034EA08F79"],
["nkechiyere@aol.com","F222791CC077C629D9357F13D2143F82"],
["chxyu@ucsc.edu","6A1759F184BCA1DA10F19441390EAA34"],
["bgammill@drury.edu","754968094C842A07B663962196A776EC"],
];

// Generate a random index into the test data.
var randomIndex = Math.min(Math.round(Math.random() * (testData.length - 1)), testData.length - 1);

// Create and populate custom properties for this instance of this Track.
var propertyList = $context.currentTrack.propertyList;
if ($context.currentClipIndex == 0)
propertyList.createProperty("email");
propertyList.setPropertyValue("email", testData[randomIndex][0]);
if ($context.currentClipIndex == 0)
propertyList.createProperty("password");
propertyList.setPropertyValue("password", testData[randomIndex][1]);
```

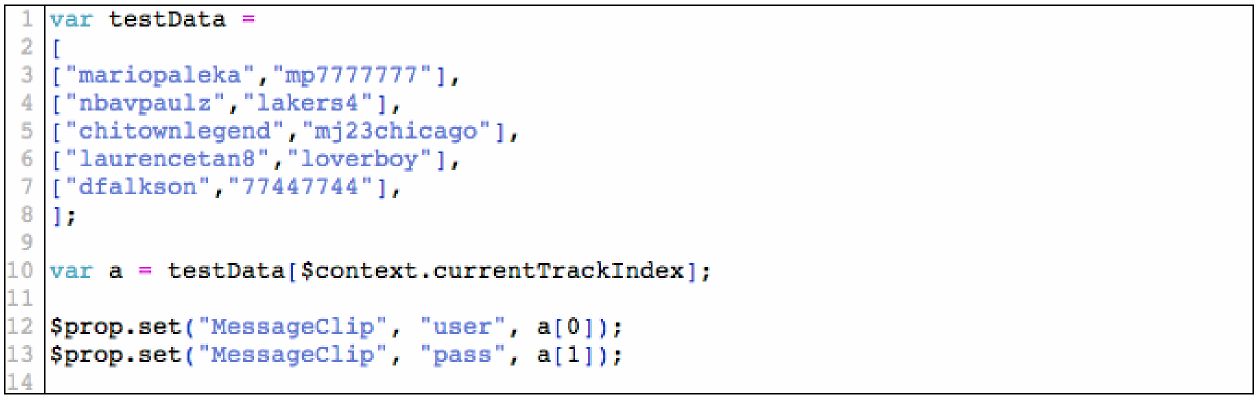
First, define the set of data from which to select (user ID and password).	`var testData =`
	`[`
	`["lindab462@aol.com","4690C47DADADA88BF8F4C6A2F262798A"],`
	`["dancehottie90@msn.com","109A25E92070491C83C2ED3ABE96CACE"],`
	`["Cortneyjboyd@gmail.com","FF94B25791A80AF1C5E11FAF66038226"],`
	`["Gakouri@suffolk.edu","C7561DB7A418DD39B2201DFE110AB4A4"],`
	`["tomshea9@hotmail.com","60FB1504776106EAEC25522A6CF4D9C8"],`
	`["gonavy90@hotmail.com","DCFC756608AC5C7578C59E306CDEF7A9"],`
	`["boyki1vm@cmich.edu","FCE9CD59CCEAA9290829200F27D150FB"],`
	`["swalker@aps4kids.org","A5410EE37744C574BA5790034EA08F79"],`
	`["nkechiyere@aol.com","F222791CC077C629D9357F13D2143F82"],`
	`["chxyu@ucsc.edu","6A1759F184BCA1DA10F19441390EAA34"],`
	`["bgammill@drury.edu","754968094C842A07B663962196A776EC"],`
	`];`

Generate a random index into the test data.	`var randomIndex = Math.min(Math.round(Math.random() *`
	`(testData.length - 1)), testData.length - 1);`

Create and populate custom properties for this instance of this Track.	`var propertyList = $context.currentTrack.propertyList;`
	`if ($context.currentClipIndex == 0)`
	`propertyList.createProperty("email");`
	`propertyList.setPropertyValue("email",`
	`testData[randomIndex][0]);`
	`if ($context.currentClipIndex == 0)`
	`propertyList.createProperty("password");`
	`propertyList.setPropertyValue("password",`
	`testData[randomIndex][1]);`

Script 6: Two-element data array, increment through values

The Two-element data array, increment through values script is run at the beginning of each clip. It loads each element of the array into its own clip property based on the track index (virtual user number). The user selected in this script is used throughout the track – it doesn't change on the same track (`currentTrackIndex`).



```
var testData =
[
["mariopaleka","mp7777777"],
["nbavpaulz","lakers4"],
["chitownlegend","mj23chicago"],
["laurencetan8","loverboy"],
["dfalkson","77447744"],
];

var a = testData[$context.currentTrackIndex];

$prop.set("MessageClip", "user", a[0]);
$prop.set("MessageClip", "pass", a[1]);
```

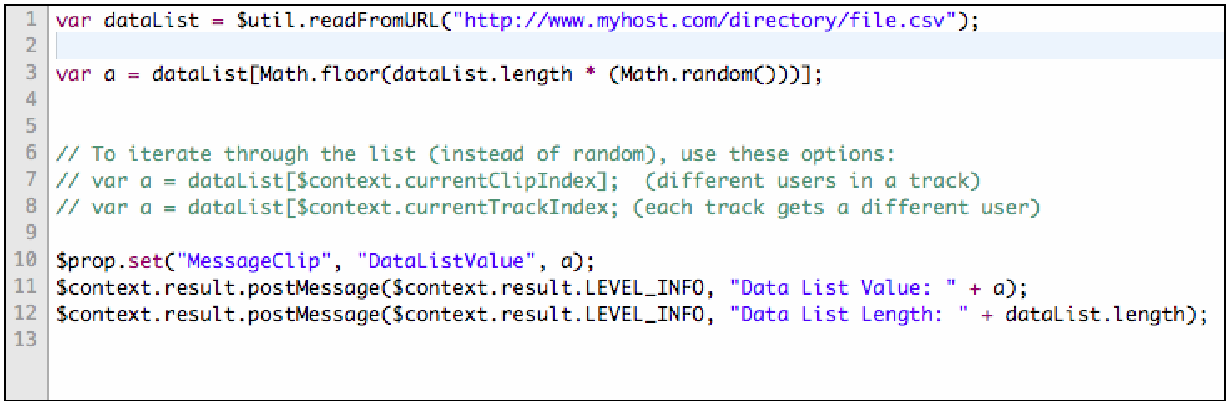
To change the user for each repeat of the test clip, use `currentClipIndex` instead.
 	
Note: Ensure that you have enough data in the script to handle the number of virtual users and serial repeats of the test clip.
	`var testData =`
	`[`
	`["mariopaleka","mp7777777"],`
	`["nbavpaulz","lakers4"],`
	`["chitownlegend","mj23chicago"],`
	`["laurencetan8","loverboy"],`
	`["dfalkson","77447744"],`
	`];`
	
	`var a = testData[$context.currentTrackIndex];`
	
	`$prop.set("MessageClip", "user", a[0]);`
	`$prop.set("MessageClip", "pass", a[1]);`

Script 7: Seed data from URL

The Seed Data from URL script randomly selects one of the lines from a list and puts that value into a clip property named DataListValue.

This example assumes there is just a single element on each row in the CSV file. Meaning, it is a simple list of values to be used in the test. The data would look like this in the file:

```
Username1
Username2
Username3
...
```



```
1  var dataList = $util.readFromURL("http://www.myhost.com/directory/file.csv");
2  
3  var a = dataList[Math.floor(dataList.length * (Math.random()))];
4  
5  
6  // To iterate through the list (instead of random), use these options:
7  // var a = dataList[$context.currentClipIndex];  (different users in a track)
8  // var a = dataList[$context.currentTrackIndex; (each track gets a different user)
9  
10 $prop.set("MessageClip", "DataListValue", a);
11 $context.result.postMessage($context.result.LEVEL_INFO, "Data List Value: " + a);
12 $context.result.postMessage($context.result.LEVEL_INFO, "Data List Length: " + dataList.length);
13 
```

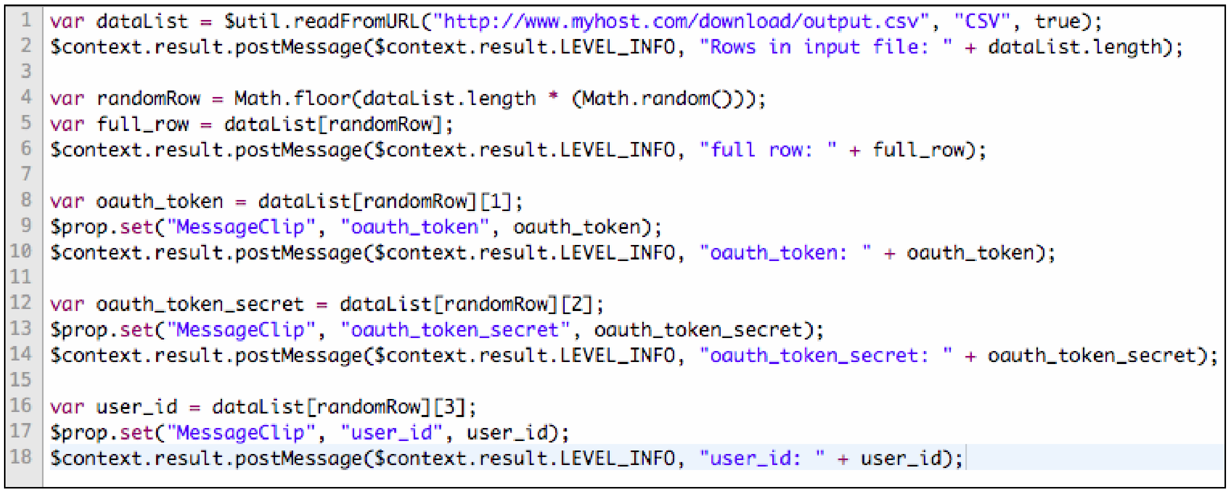
In line 1, declare a variable using `util.readFromURL`.	`var dataList = $util.readFromURL("http://www.myhost.com/directory/file.csv");`
Make a variable, a, for the increment.	`var a = dataList[Math.floor(dataList.length * (Math.random()))];`

To iterate through the list (instead of random), use these options:
 	
To iterate through the list (instead of random), use these options:
 	
1. `var a = dataList[$context.currentClipIndex];`
 	
(different users in a track)
 	
2. `var a = dataList[$context.currentTrackIndex];`
 	
(each track gets a different user)
	`$prop.set("MessageClip", "DataListValue", a);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "Data List Value: " + a);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "Data List Length: " + dataList.length);`

This functionality can be expanded to extract data from multiple comma-separated values on the same line. Here are additional examples. If the file contained the following lines:
`San Francisco,CA,94103`
`Timbuktu`
`Caribou,Aroostook County,Maine,USA`
`Hill Valley,CA,91905`
The array of strings on the right would be returned:
	`dataList[0][0]  San Francisco`
	`dataList[0][1]  CA`
	`dataList[0][2]  94103`
	
	`dataList[1]  Timbuktu`
	
	`dataList[2][0]  Caribou`
	`dataList[2][1]  Aroostook County`
	`dataList[2][2]  Maine`
	`dataList[2][3]  USA`
	
	`dataList[3][0]  Hill Valley`
	`dataList[3][1]  CA`
	`dataList[3][2]  91905`

Script 8: Seed Data from URL – second option

This second Seed Data from URL script is similar to the first, but instead extracts multiple columns in the input comma-separated values (CSV) file.

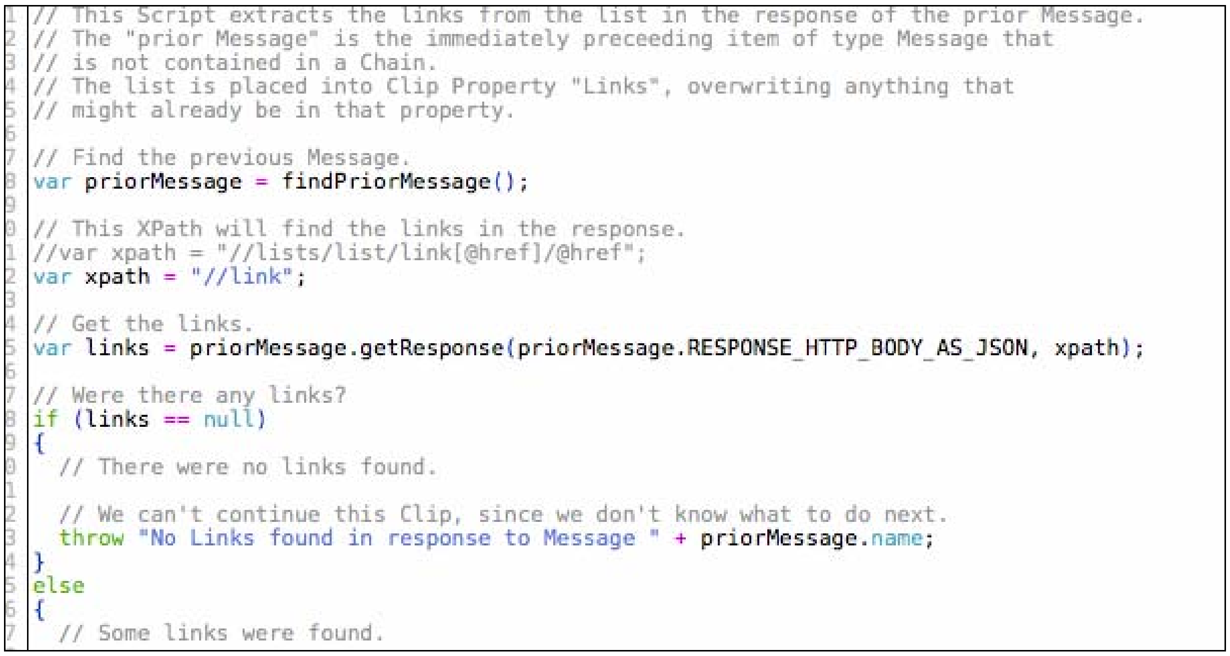


```
var dataList = $util.readFromURL("http://www.myhost.com/download/output.csv", "CSV", true);
$context.result.postMessage($context.result.LEVEL_INFO, "Rows in input file: " + dataList.length);

var randomRow = Math.floor(dataList.length * (Math.random()));
var full_row = dataList[randomRow];
$context.result.postMessage($context.result.LEVEL_INFO, "full row: " + full_row);

var oauth_token = dataList[randomRow][1];
$prop.set("MessageClip", "oauth_token", oauth_token);
$context.result.postMessage($context.result.LEVEL_INFO, "oauth_token: " + oauth_token);

var oauth_token_secret = dataList[randomRow][2];
$prop.set("MessageClip", "oauth_token_secret", oauth_token_secret);
$context.result.postMessage($context.result.LEVEL_INFO, "oauth_token_secret: " + oauth_token_secret);

var user_id = dataList[randomRow][3];
$prop.set("MessageClip", "user_id", user_id);
$context.result.postMessage($context.result.LEVEL_INFO, "user_id: " + user_id);
```

Remember that the first column is index value 0. In the example below, it is taking columns 2, 3, and 4 (index values 1, 2, and 3).	`var dataList = $util.readFromURL("http://www.soasta.com/download/output.csv", "CSV", true);`
	
	`$context.result.postMessage($context.result.LEVEL_INFO, "Rows in input file: " + dataList.length);`
	
	`var randomRow = Math.floor(dataList.length * (Math.random()))`
	`var full_row = dataList[randomRow];`
	`$context.result.postMessage($context.result.LEVEL_INFO, "full row: " + full_row);`
	
	`var oauth_token = dataList[randomRow][1];`
	`$prop.set("MessageClip", "oauth_token", oauth_token);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "oauth_token: " + oauth_token);`
	
	`var oauth_token_secret = dataList[randomRow][2];`
	`$prop.set("MessageClip", "oauth_token_secret", oauth_token_secret);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "oauth_token_secret: " + oauth_token_secret);`
	
	`var user_id = dataList[randomRow][3];`
	`$prop.set("MessageClip", "user_id", user_id);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "user_id: " + user_id);`

Script 9: Put an Array into a Clip Property

The Put an Array into a Clip Property script loads "links" into an array (red text shows actual setting of array to a clip property). This script demonstrates how to save variable context between chain (or collection) loops. The script extracts the links from the list in the response of the prior Message. The "prior Message" is the immediately preceeding item of type Message that is not contained in a chain. The list is placed into Clip Property "Links", overwriting anything that might already be in that property.



```
// This Script extracts the links from the list in the response of the prior Message.
// The "prior Message" is the immediately preceeding item of type Message that
// is not contained in a Chain.
// The list is placed into Clip Property "Links", overwriting anything that
// might already be in that property.

// Find the previous Message.
var priorMessage = findPriorMessage();

// This XPath will find the links in the response.
//var xpath = "//lists/list/link[@href]/@href";
var xpath = "//link";

// Get the links.
var links = priorMessage.getResponse(priorMessage.RESPONSE_HTTP_BODY_AS_JSON, xpath);

// Were there any links?
if (links == null)
{
  // There were no links found.

  // We can't continue this Clip, since we don't know what to do next.
  throw "No Links found in response to Message " + priorMessage.name;
}
else
{
  // Some links were found.
```

Find the previous Message.	`var priorMessage = findPriorMessage();`
This XPath will find the links in the response.	`var xpath = "//link";`
Get the links.	`var links = priorMessage.getResponse(priorMessage.RESPONSE_HTTP_BODY_AS_JSON, xpath);`

Were there any links?
 	
If no links throw an error message. If links are found, display in the result.
	`if (links == null)`
	`{`
	`else`
	`{`
	`  displayLinks(links.length + " Links found in response to Message " + priorMessage.name, links);`

Put the links in the Clip property as a list of links in the Result in a single entry.	`  $context.currentClip.propertyList.setPropertyValue("Links", links);`
	`}`

The caller specifies the title and the details contains the list.
function displayLinks (title, links)
	`{`
	`  var text = "";`
	`  for (var i = 0; i < links.length; i++)`
	`  {`
	`  text += "[" + i + "] " + links[i] + "\n";`
	` }`
	
	` $context.result.postMessage($context.result.LEVEL_INFO, title, text);`
	`}`

This function finds the immediately preceding Message from the current position, ignoring anything nested inside Chains. Throws an error if there is no prior Message.	`function findPriorMessage()`
	`{`
	`  var priorMessage = $context.currentItem.previousItem;`
	`  for (;;)`
	`  {`
	`    if (priorMessage == null)`
	`      throw "No prior Message";`
	`    if (priorMessage.type == "Message")`
	`      break;`
	`    priorMessage = priorMessage.previousItem;`
	`  }`
	`  return priorMessage;`
	`}`

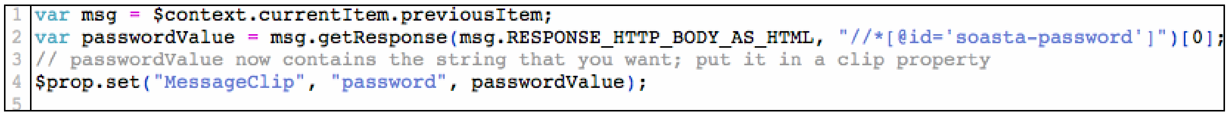
Extraction Scripts

This section presents a variety of CloudTest extraction scripts. Scripts can be copied and pasted directly into the SOASTA CloudTest Script Editor using Central > Scripts. Longer scripts have an accompanying link that can be clicked to pop out the full example script. Shorter scripts are presented inline.

Script 1: Extract from Response Body (HTML)

The Extract from Response Body (HTML) script extracts an element from a page response and sets a clip property. This is a versatile example because you can change the XPath to extract different elements. The XPath below extracts the soasta-password value from a file that returns a response like this:

```
<div id="soasta-password" style="display:none">rogxWFmmYR</div></div>
```



```
1 var msg = $context.currentItem.previousItem;
2 var passwordValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@id='soasta-password']")[0];
3 // passwordValue now contains the string that you want; put it in a clip property
4 $prop.set("MessageClip", "password", passwordValue);
5
```

The script itself is relatively simple. The first two lines determine the context and search for the id `soasta-password`. Line 1 sets the test clip context for the message	`var msg = $context.currentItem.previousItem;`
Line 2 extracts the value of “soasta-password” from the message response using XPath.	`var passwordValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@id='soasta-password']")[0];`

The last line sets a clip property, `passwordValue`, to contain the relevant string from the variable of the same name.	`$prop.set("MessageClip", "password", passwordValue);`

The following script presents a slight variation of the prior script by extracting an id from an HTML page:

Set the context to the preceding message, and then declare a variable, `semesterInputValue`, to get the XPath specified from the HTML response.	`var msg = $context.currentItem.previousItem.previousItem;`
	
	`var semesterInputValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@id='term_13']/@value")[0];`
	
	`$prop.set("MessageClip", "term", semesterInputValue);`
	
	`$context.result.postMessage($context.result.LEVEL_INFO, "sem: " + semesterInputValue);`

The `semesterInputValue` now contains the string to put into a clip property:	`$prop.set("MessageClip", "term", semesterInputValue);`

XPath Samples Table

The XPath Samples Table presents some common XPath values used in the sample CloudTest scripts in this guide along with the HTML that they retrieve.

Note: Browser add-ons that capture or identify XPaths within the Document Object Model (DOM) of a given HTML page and browser are useful tools in test creation. For example, XPather for Firefox, or Safari XPath.

Retrieves	**XPath**
`<div id="soasta-password" style="display:none">rogxWFmmYR</div></div>`	`//*[@id='soasta-password']`

`<input type="hidden" name="form_build_id" id="form-5287344f04574e13fe662dce074dc3fa" value="form-5287344f04574e13fe662dce074dc3fa" />`	`//*[@name='form_build_id']/@value`
`name=form_token id=edit-form-token value=92b837a209278461ab88190c8e912af6`	`//*[@name='form_token']/@value`
`<li class="leaf" id="menu-leaf50000Myaccount21"><a href="/user/16">My account</a></li>`	`//li[@id='menu-leaf50000Myaccount21']/a/@href` (extracts the /user/16 portion)

Script 2: Extract from Response Body – SubString

This example does multiple substrings of substrings.

Line 1 establishes the context where the property will be set, while line 2 declares a variable to store the JSON element from the response. Lines 3-8 retrieve the search string by position. Lines 10-11 set clip property to store the result and posts a message to results.	`var msg = $context.currentItem.previousItem;`
	`var origText = msg.getResponse(msg.RESPONSE_HTTP_BODY);`
	
	`var searchString = "<ufid>";`
	`var pos = origText.indexOf(searchString);`
	`var newText = origText.substring(pos + searchString.length);`
	`var pos = newText.indexOf("</ufid>");`
	`newText = newText.substring(0, pos);`
	
	`$prop.set("MessageClip", "ufid", newText);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "ufid: " + newText);`

Script 3: Extract from Response Body (JSON)

The Extract from Response Body (JSON) script extracts an element from a JSON page response and sets a clip property.

Line 1 establishes the context where the property will be set, while line 2 declares a variable to store the JSON element.	`var msg = $context.currentItem.previousItem;` `var authid = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_JSON, "//authid")[0];`
Line 3 sets debug information to be posted into test results.	`$context.result.postMessage($context.result.LEVEL_INFO, "authid: " + authid);`
Finally, line 4 sets a clip property using the variable from line 2.	`$prop.set("MessageClip", "authid", authid);`

Script 4: Extract from Response Header

This script extracts data from a Location header using the orderId. The orderId is in the middle of the Location header and needs to be extracted using a substring.

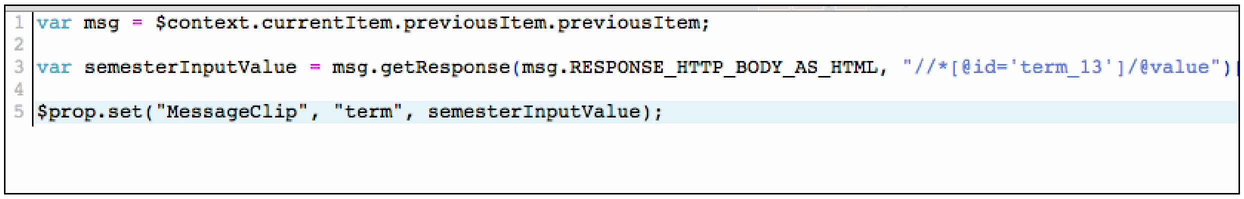
This script gets the recorded response from the previous message in a test clip using a variable, origText, which gets a specified HTTP Header.	`// This Script looks at the response to the Message that` `// precedes it and Extracts the OrderID` `// Get the Message that immediately precedes this Script.` `var msg = $context.currentItem.previousItem;` `var origText = msg.getResponse(msg.RESPONSE_HTTP_HEADER, "Location");` `var pos = origText.indexOf("orderId=");` `var newText = origText.substring(pos + 8);` `var pos = newText.indexOf("&deliveryDate");` `newText = newText.substring(0, pos);` `$prop.set("MessageClip", "OrderID", newText);` `$context.result.postMessage($context.result.LEVEL_INFO, "order ID: " + newText);`

This script also uses `previousItem` to extract data from a header in a recorded response. In this case, the header value (krypto) is the last item on the Location header.
 	
Note: In some cases, non-visible trailing spaces may be present. To get an accurate extraction, make sure you trim any spaces to the right of the value to extract.
	`// Get the Message that immediately precedes this Script.`
	`var msg = $context.currentItem.previousItem;`
	
	`var origText = msg.getResponse(msg.RESPONSE_HTTP_HEADER,`
	`"Location");`
	
	`var pos = origText.indexOf("krypto=");`
	`var newText = origText.substring(pos + 7);`
	
	`$prop.set("MessageClip", "kryptoA", newText);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "kryptoA`
	`" + newText);`

`Once again using previousItem to extract from a prior message in a test clip, this time extracting a JSESSIONID from a “Set-Cookie” header.`	`// This Script looks at the response to the Message that`
	`// precedes it and Extracts the jsessionId cookie`
	
	`// Get the Message that immediately precedes this Script.`
	`var msg = $context.currentItem.previousItem;`
	
	`var origText = msg.getResponse(msg.RESPONSE_HTTP_HEADER, "Set-`
	`Cookie");`
	
	`var pos = origText.indexOf("JSESSIONID=");`
	`var newText = origText.substring(pos + 11);`
	`var pos = newText.indexOf(";");`
	`newText = newText.substring(0, pos);`
	
	`$prop.set("MessageClip", "jsessionId", newText);`
	`$context.result.postMessage($context.result.LEVEL_INFO,`
	`"JSESSIONID: " + newText);`

Script 5: Extracting Data from a Message TWO Prior

The Extracting Data from a Message TWO Prior script extracts information from TWO MESSAGES prior using XPath and then puts that information into a property..

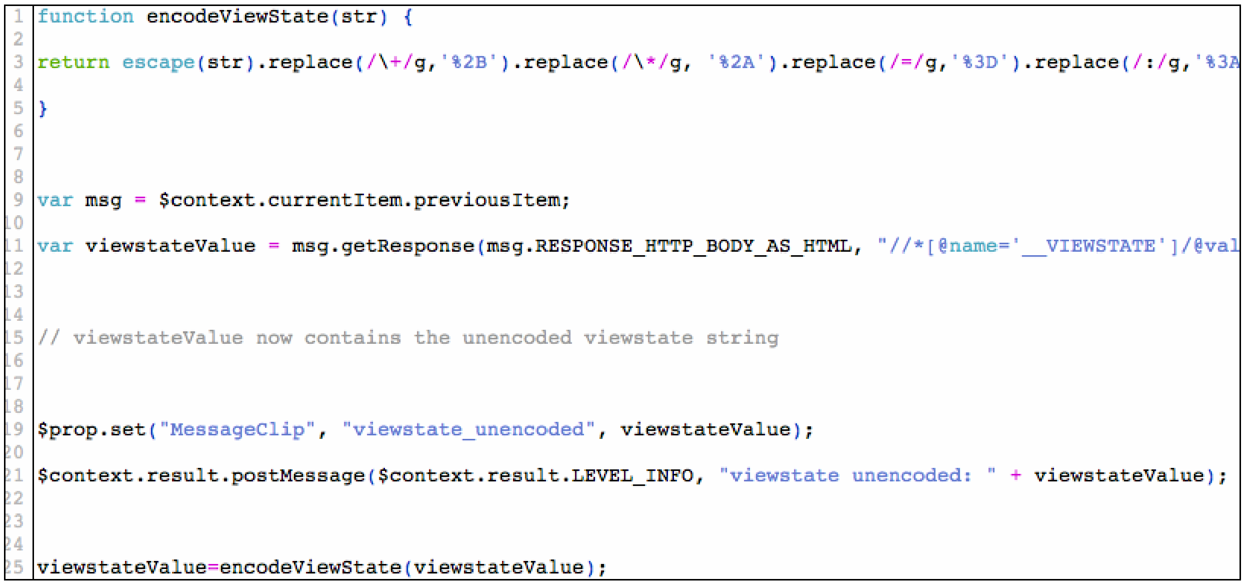


```
1 var msg = $context.currentItem.previousItem.previousItem;
2
3 var semesterInputValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@id='term_13']/@value")
4
5 $prop.set("MessageClip", "term", semesterInputValue);
```

Line 1 establishes the clip element TWO items before the script as the context.	`var msg = $context.currentItem.previousItem.previousItem;`
Line 3 declares the variable, semesterInputValue, and specifies the XPath ID to find.	`var semesterInputValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@id='term_13']/@value")[0];`
Line 5 places the extracted data into a clip property	`$prop.set("MessageClip", "term", semesterInputValue);`

Script 6: Extracting and Encoding a VIEWSTATE and EVENTVALIDATION

Use the Extracting and encoding a VIEWSTATE and EVENTVALIDATION script to extract the common .Net fields, VIEWSTATE and EVENTVALIDATION, from recorded responses. These fields are typically hidden in an HTML file.



```
 1 function encodeViewState(str) {
 2
 3 return escape(str).replace(/\+/g,'%2B').replace(/\*/g, '%2A').replace(/=/g,'%3D').replace(/:/g,'%3A
 4
 5 }
 6
 7
 8
 9 var msg = $context.currentItem.previousItem;
10
11 var viewstateValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='__VIEWSTATE']/@val
12
13
14
15 // viewstateValue now contains the unencoded viewstate string
16
17
18
19 $prop.set("MessageClip", "viewstate_unencoded", viewstateValue);
20
21 $context.result.postMessage($context.result.LEVEL_INFO, "viewstate unencoded: " + viewstateValue);
22
23
24
25 viewstateValue=encodeViewState(viewstateValue);
```

Lines 1-5 create an encoding function as string, `encodeViewState`.	`function encodeViewState(str) {`
	`return escape(str).replace(/\+/g,'%2B').replace(/\*/g, '%2A').replace(/=/g,'%3D').replace(/:/g,'%3A').replace(/;/g,'%3B').replace(/\//g, '%2F');`
	`}`

This script will use the `previousItem` position to access a message. The message must have local or public scope.	`var msg = $context.currentItem.previousItem;`
A variable, viewstateValue, is declared to contain the unencoded string. The variable now has the unencoded string.	`var viewstateValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='__VIEWSTATE']/@value")[0];`

A test clip custom property of the same name is created to take the value.	`$prop.set("MessageClip", "viewstate_unencoded", viewstateValue);`

Debug information is posted to test results.	`$context.result.postMessage($context.result.LEVEL_INFO, "viewstate unencoded: " + viewstateValue);`
	`encoded: " + viewstateValue);`

Using the viewstateValue, an encoded version is created for use in subsequent message requests.	`viewstateValue=encodeViewState(viewstateValue);`
A test clip custom property of the same name is created for the encoded viewstate.	`$prop.set("MessageClip", "viewstate_encoded", viewstateValue);`
Set debug information for results.	`$context.result.postMessage($context.result.LEVEL_INFO, "viewstate`
The viewstateValue now contains the unencoded viewstate string	`var viewstateValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='__VIEWSTATE']/@value")[0];`
A test clip custom property of the same name is created for the unencoded viewstate.	`$prop.set("MessageClip", "viewstate_unencoded", viewstateValue);`
Set debug information for results.	`$context.result.postMessage($context.result.LEVEL_INFO, "viewstate unencoded: " + viewstateValue);`

The viewstateValue now contains the encoded viewstate string., so set a property in the next line.	`viewstateValue=encodeViewState(viewstateValue);`

A test clip custom property of the same name is created and the value is inserted.	`$prop.set("MessageClip", "viewstate_encoded", viewstateValue);`
	`name='__EVENTVALIDATION']/@value")[0];`

Set debug information for results.	`$context.result.postMessage($context.result.LEVEL_INFO, "viewstate encoded: " + viewstateValue);`
Declare a variable for the eventvalidationValue extraction and then get the recorded response as HTML using the XPath.	`var eventvalidationValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, , "//*[@name='__EVENTVALIDATION']/@value")[0];`
Now that the eventvalidationValue contains the unencoded string, set a property for use in subsequent messages.	`$prop.set("MessageClip", "eventvalidation_unencoded", eventvalidationValue);`
Set debug information for results.	`$context.result.postMessage($context.result.LEVEL_INFO, "eventvalidation unencoded: " + eventvalidationValue);`

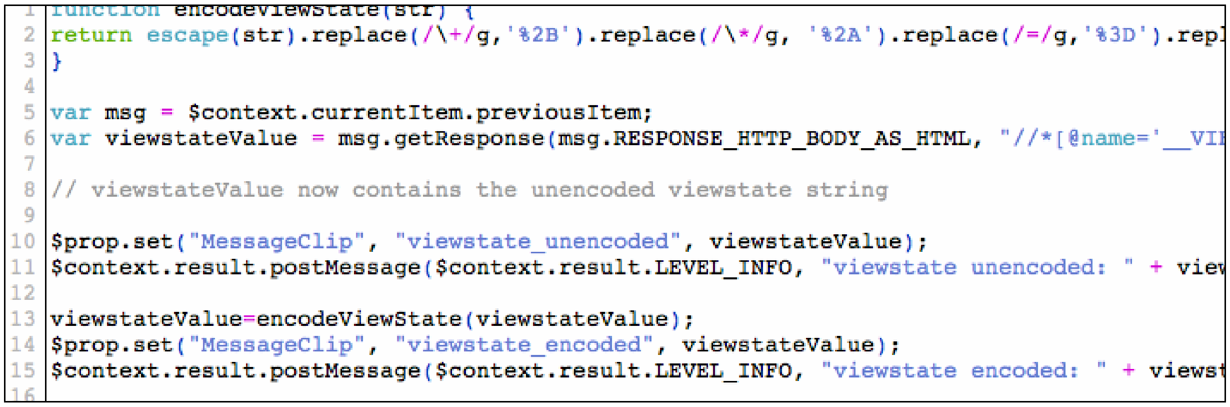
Encode the `eventvalidationValue` for use in subsequent message requests.	`eventvalidationValue=encodeURIComponentencodeViewState(eventvalidationValue);`
	`encoded: " + eventvalidationValue);`

Set a test clip custom property.	`$prop.set("MessageClip", "eventvalidation_encoded", eventvalidationValue);`
Set debug information for results.	`$context.result.postMessage($context.result.LEVEL_INFO, "eventvalidation`

Script 7: Extracting and Encoding a VIEWSTATE

The Extracting and encoding a VIEWSTATE script extracts only a VIEWSTATE value. The VIEWSTATE is typically a hidden field in an HTML file and is frequently used in .NET applications.

Most POST messages in a .NET application have a VIEWSTATE while not all have EVENTVALIDATION parameters. This script is sometimes used in combination with the prior script and is applied when a given field has no EVENTVALIDATION parameter.



```
 1 function encodeViewState(str) {
 2 return escape(str).replace(/\+/g,'%2B').replace(/\*/g, '%2A').replace(/=/g,'%3D').repl
 3 }
 4
 5 var msg = $context.currentItem.previousItem;
 6 var viewstateValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='__VI
 7
 8 // viewstateValue now contains the unencoded viewstate string
 9
10 $prop.set("MessageClip", "viewstate_unencoded", viewstateValue);
11 $context.result.postMessage($context.result.LEVEL_INFO, "viewstate unencoded: " + view
12
13 viewstateValue=encodeViewState(viewstateValue);
14 $prop.set("MessageClip", "viewstate_encoded", viewstateValue);
15 $context.result.postMessage($context.result.LEVEL_INFO, "viewstate encoded: " + viewst
16
```

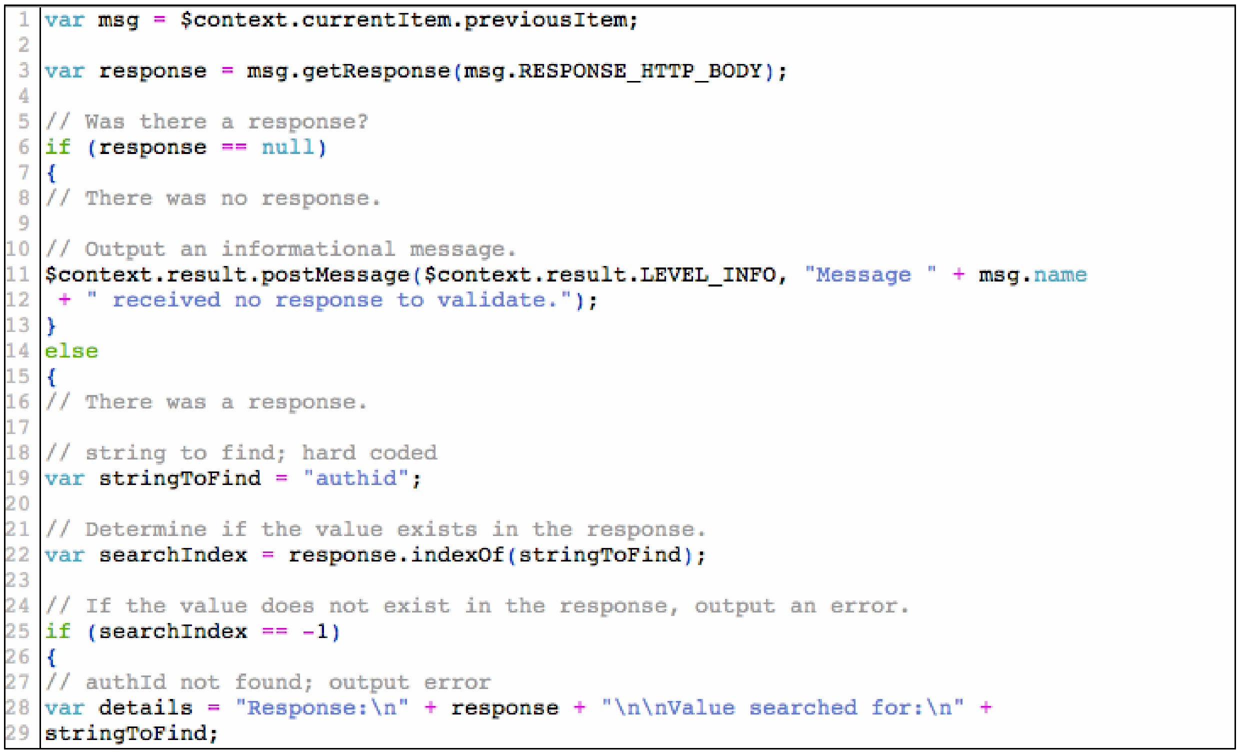
Lines 1-5 create an encoding function as string, `encodeViewState`.	`function encodeViewState(str)`
	`{`
	` return escape(str).replace(/\+/g,'%2B').replace(/\*/g, '%2A').replace(/=/g,'%3D').replace(/:/g,'%3A').replace(/;/g,'%3B').replace(/\//g, '%2F');`
	`}`

Get the message response from recorded HTML and extract the `viewstateValue` using XPath, after which `viewstateValue` contains the unencoded viewstate string.	`var msg = $context.currentItem.previousItem; var viewstateValue = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='__VIEWSTATE']/@value")[0];`
	
	`state encoded: " + viewstateValue);`

Set a custom clip property of the same name and insert the unencoded value.	`$prop.set("MessageClip", "viewstate_unencoded", viewstateValue);`
Post debug information to results.	`$context.result.postMessage($context.result.LEVEL_INFO, "viewstate unencoded: " + viewstateValue);`
Create an encoded version of the `viewstateValue` for use in subsequent message requests.	`viewstateValue=encodeURIComponentencodeViewState(viewstateValue);`
Set the custom clip property value of the same name and insert the encoded value.	`$prop.set("MessageClip", "viewstate_encoded", viewstateValue);`
Post debug information to results.	`$context.result.postMessage($context.result.LEVEL_INFO, "eventvalidation encoded: " + eventvalidationValue);`

Script 8: Verify Something Exists in a Response before Attempting Extraction

In the Verify something exists in a response before the extraction is attempted, first verify data before extracting data for use elsewhere, such as in a clipProperty that stores an `authID` to be used in the `nextItem` in a test clip. Use `searchIndex` to find a string in the response.



```
 1 var msg = $context.currentItem.previousItem;
 2
 3 var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);
 4
 5 // Was there a response?
 6 if (response == null)
 7 {
 8 // There was no response.
 9
10 // Output an informational message.
11 $context.result.postMessage($context.result.LEVEL_INFO, "Message " + msg.name
12  + " received no response to validate.");
13 }
14 else
15 {
16 // There was a response.
17
18 // string to find; hard coded
19 var stringToFind = "authid";
20
21 // Determine if the value exists in the response.
22 var searchIndex = response.indexOf(stringToFind);
23
24 // If the value does not exist in the response, output an error.
25 if (searchIndex == -1)
26 {
27 // authId not found; output error
28 var details = "Response:\n" + response + "\n\nValue searched for:\n" +
29 stringToFind;
```

Line 1 sets the context to the `previousItem`, which must have local or public scope.	`var msg = $context.currentItem.previousItem;`
Line 3 declares a variable for the response.	`var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);`

Lines 6-15 are an "if...else" statement that determine if there was a response and outputs debug information to results if null.	`if (response == null)`
	`{`
	`// There was no response.`
	
	`// Output an informational message.`
	`$context.result.postMessage($context.result.LEVEL_INFO, "Message`
	`" + msg.name`
	` + " received no response to validate.");`
	`}`
	`else`
	`{`
	`// There was a response.`

Line 19 defines the variable `stringToFind` and sets its value as `authid`.	`var stringToFind = "authid";`

Lines 22-30 declares a variable, `searchIndex`, which gets the search result for `stringToFind`. If the string is not found Set debug information for results.	`var searchIndex = response.indexOf(stringToFind);`
	
	`// If the value does not exist in the response, output an error.`
	`if (searchIndex == -1)`
	`{`
	`// authId not found; output error`
	`var details = "Response:\n" + response + "\n\nValue searched`
	`for:\n" +`
	`stringToFind;`
	`$context.result.postMessage($context.result.LEVEL_ERROR,`
	`"Expected value not found in response to Message " + msg.name,`
	`details);`

Lines 36-43 definite and if...else statement that sets the next message play count=1. The `nextItem` is a SignIn message (duplicate of the message before this script) that retries the login. The else (line 42) comes into effect if the login succeeds.	`var nextItem = $context.currentItem.nextItem;`
	`  nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL,`
	`nextItem.REPEAT_TYPE_COUNT_CONSTANT, 1,`
	`nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);`
	
	`}`
	`else`
	`{`

In Lines 46-49, if the `authid` exists, put it in a test clip custom property.	`var authId = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_JSON, "//authid")[0];`
	`$context.result.postMessage($context.result.LEVEL_INFO, "authId: " +`
	`authId);`
	`$prop.set("MessageClip", "authId", authId);`

In Lines 51-59, reset the play count of the nextItem to 0 and then clear the response.	`var nextItem = $context.currentItem.nextItem;`
	`nextItem.nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL,`
	`nextItem.REPEAT_TYPE_COUNT_CONSTANT, 0,`
	`nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);`
	
	`}`
	`}`
	
	`msg.clearResponse();`

Script 9: Extract HTTP Status Code from Response

An HTTP status code can be extracted from a response header using the following one-line script:

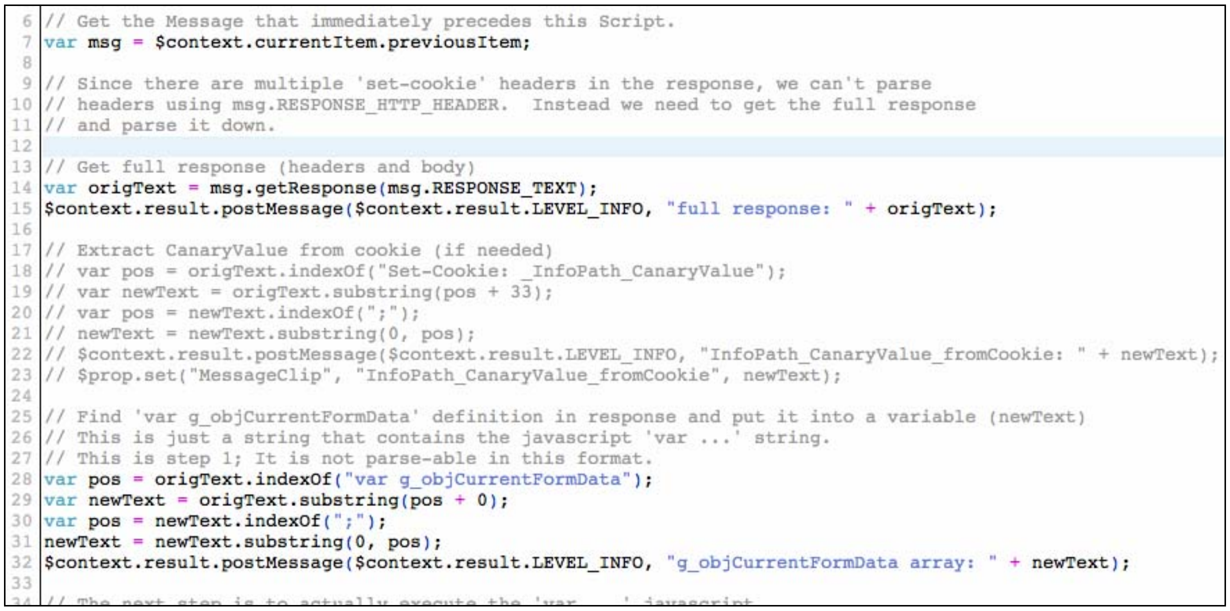
```
var StatusCode = msg.getResponse(msg.RESPONSE_HTTP_STATUSCODE);
```

Script 10: InfoPath Forms Services 2007/MOSS variable extraction

Variables in InfoPath Forms Services (IPFS) have the following general characteristics:

- IPFS has two main transactions when interacting with a form: Invocation and Postback
- Editing Session Id, Solution Id, and Canary are dynamic. They will need to be extracted from responses.
- Editing Session Id, Solution Id, and Canary are used in the postback request. Their values will need to be substituted in the postback body.
- Editing Session Id and Solution Id do not change between postbacks. They will only need to be extracted for the invocation request.
- Canary changes after every postback. It needs to be extracted for every request.

The first IFPS script extracts all 3 variables after the initial form load. /This Script looks at the response to the Message that precedes it and Extracts variables specific to MOSS InfoPath applications. This is the first script which extracts 3 variables. EditingSessionId and SolutionId do not change after this initial extraction, but the CanaryValue does change after each form postback.



```
// Get the Message that immediately precedes this Script.
var msg = $context.currentItem.previousItem;

// Since there are multiple 'set-cookie' headers in the response, we can't parse
// headers using msg.RESPONSE_HTTP_HEADER.  Instead we need to get the full response
// and parse it down.

// Get full response (headers and body)
var origText = msg.getResponse(msg.RESPONSE_TEXT);
$context.result.postMessage($context.result.LEVEL_INFO, "full response: " + origText);

// Extract CanaryValue from cookie (if needed)
// var pos = origText.indexOf("Set-Cookie: _InfoPath_CanaryValue");
// var newText = origText.substring(pos + 33);
// var pos = newText.indexOf(";");
// newText = newText.substring(0, pos);
// $context.result.postMessage($context.result.LEVEL_INFO, "InfoPath_CanaryValue_fromCookie: " + newText);
// $prop.set("MessageClip", "InfoPath_CanaryValue_fromCookie", newText);

// Find 'var g_objCurrentFormData' definition in response and put it into a variable (newText)
// This is just a string that contains the javascript 'var ...' string.
// This is step 1; It is not parse-able in this format.
var pos = origText.indexOf("var g_objCurrentFormData");
var newText = origText.substring(pos + 0);
var pos = newText.indexOf(";");
newText = newText.substring(0, pos);
$context.result.postMessage($context.result.LEVEL_INFO, "g_objCurrentFormData array: " + newText);

// The next step is to actually execute the 'var ... ' javascript
```

Get the message that precedes this script.	`var msg = $context.currentItem.previousItem;`
Since there are multiple 'set-cookie' headers in the response, we can't parse headers using `msg.RESPONSE_HTTP_HEADER`. Instead, get the full response and parse it to the relevant portion.	`var origText = msg.getResponse(msg.RESPONSE_TEXT);` `$context.result.postMessage($context.result.LEVEL_INFO, "full response: " + origText);`

Find `var g_objCurrentFormData` in the response and put it into a variable (`newText`). This is just a string that contains the JavaScript `'var . . .'` string. This is step 1; It is not parse-able in this format.	`var pos = origText.indexOf("var g_objCurrentFormData");`
	`var newText = origText.substring(pos + 0);`
	`var pos = newText.indexOf(";");`
	`newText = newText.substring(0, pos);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "g_objCurrentFormData array: " + newText);`

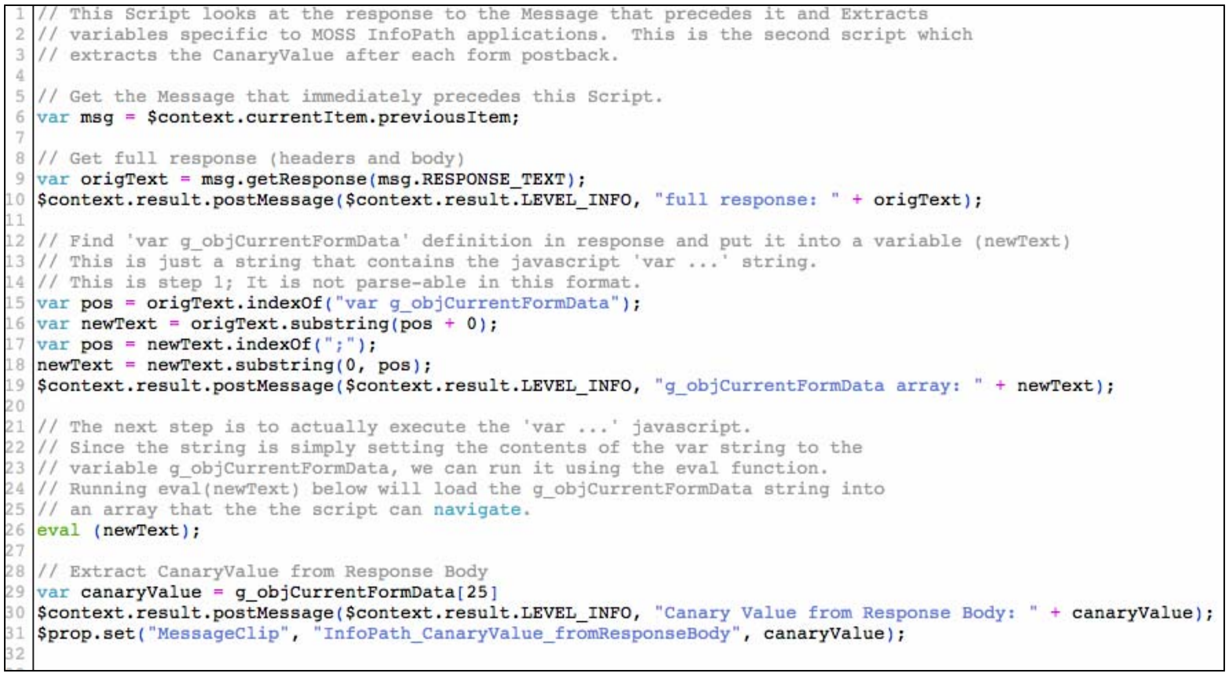
The next step is to execute the `'var ...'` JavaScript. Since the string is simply setting the contents of the var string to the variable `g_objCurrentFormData`, can be run using the eval function. Running `eval(newText)` will load the `g_objCurrentFormData` string into an array that the script can navigate.	`eval (newText);`

Extract the `sessionID`.	`var editingSessionId = g_objCurrentFormData[3]`
	`$context.result.postMessage($context.result.LEVEL_INFO, "InfoPath_editingSessionId: " + editingSessionId);`
	`$prop.set("MessageClip", "InfoPath_editingSessionId", editingSessionId);`

Extract the `solutionID`	`var solutionId_unencoded = g_objCurrentFormData[4]`
	`$context.result.postMessage($context.result.LEVEL_INFO, "InfoPath_solutionId unencoded: " + solutionId_unencoded);`
	`$prop.set("MessageClip", "InfoPath_solutionId_unencoded", solutionId_unencoded);`

Extract `CanaryValue` from Response Body	`var canaryValue = g_objCurrentFormData[25]` `$context.result.postMessage($context.result.LEVEL_INFO, "Canary Value from Response Body: " + canaryValue);` `$prop.set("MessageClip", "InfoPath_CanaryValue_fromResponseBody", canaryValue);`

Here is the second IPFS script that extracts just the CanaryValue after each form postback:



```
// This Script looks at the response to the Message that precedes it and Extracts
// variables specific to MOSS InfoPath applications.  This is the second script which
// extracts the CanaryValue after each form postback.

// Get the Message that immediately precedes this Script.
var msg = $context.currentItem.previousItem;

// Get full response (headers and body)
var origText = msg.getResponse(msg.RESPONSE_TEXT);
$context.result.postMessage($context.result.LEVEL_INFO, "full response: " + origText);

// Find 'var g_objCurrentFormData' definition in response and put it into a variable (newText)
// This is just a string that contains the javascript 'var ...' string.
// This is step 1; It is not parse-able in this format.
var pos = origText.indexOf("var g_objCurrentFormData");
var newText = origText.substring(pos + 0);
var pos = newText.indexOf(";");
newText = newText.substring(0, pos);
$context.result.postMessage($context.result.LEVEL_INFO, "g_objCurrentFormData array: " + newText);

// The next step is to actually execute the 'var ...' javascript.
// Since the string is simply setting the contents of the var string to the
// variable g_objCurrentFormData, we can run it using the eval function.
// Running eval(newText) below will load the g_objCurrentFormData string into
// an array that the the script can navigate.
eval (newText);

// Extract CanaryValue from Response Body
var canaryValue = g_objCurrentFormData[25]
$context.result.postMessage($context.result.LEVEL_INFO, "Canary Value from Response Body: " + canaryValue);
$prop.set("MessageClip", "InfoPath_CanaryValue_fromResponseBody", canaryValue);
```

Get the message that precedes this script.	`var msg = $context.currentItem.previousItem;`
Get the full response of headers and body.	`var origText = msg.getResponse(msg.RESPONSE_TEXT);` `$context.result.postMessage($context.result.LEVEL_INFO, "full response: " + origText);`

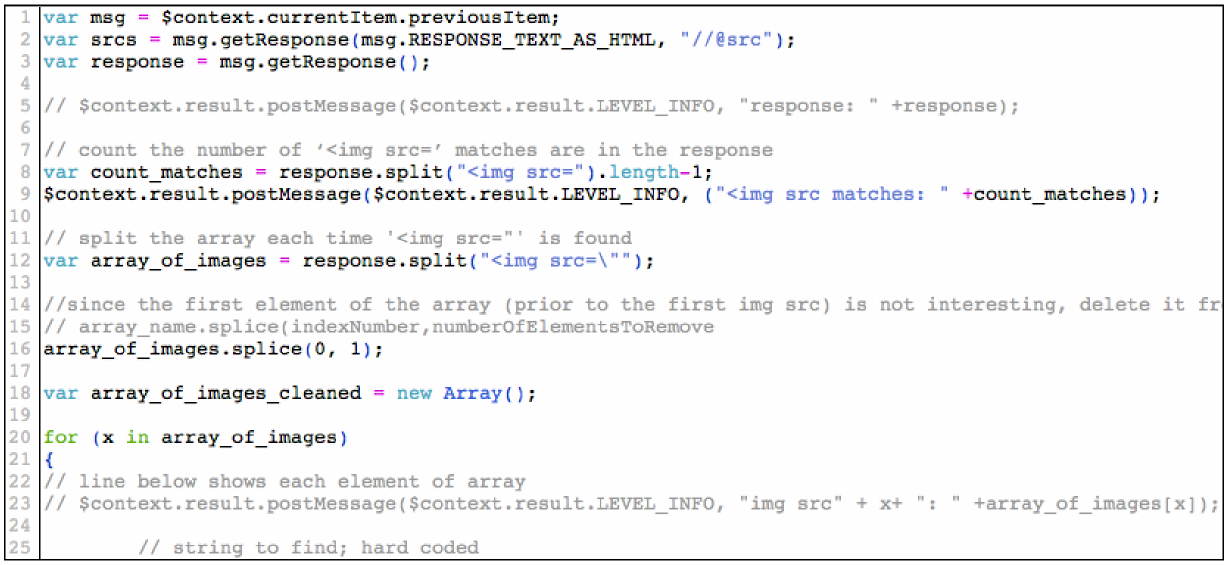
Find `'var g_objCurrentFormData'` definition in response and put it into a variable (`newText`)
 	
This is just a string that contains the JavaScript `'var` . . .' string.. It is not parse-able in this format.
	`var pos = origText.indexOf("var g_objCurrentFormData");`
	`var newText = origText.substring(pos + 0);`
	`var pos = newText.indexOf(";");`
	`newText = newText.substring(0, pos);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "g_objCurrentFormData array: " + newText);`

Running `eval(newText)` will load the `g_objCurrentFormData` string into an array that the script can navigate.	`eval (newText);`

Extract the `CanaryValue` from Response Body	`var canaryValue = g_objCurrentFormData[25]`
	`$context.result.postMessage($context.result.LEVEL_INFO, "Canary Value from Response Body: " + canaryValue);`
	`$prop.set("MessageClip", "InfoPath_CanaryValue_fromResponseBody", canaryValue);`

Script 11: Find all JPG/PNG 'IMG SRC' Links in an HTML Response

The Find All IMG SRC Links in an HTML Response script takes an HTML response, parses through it and extracts just the PNG and JPG 'img src' links. The final result is an array that contains each of the links. An important part of this script is that it removes duplicates from the array so you are left with just the unique links on the page.



```
 1 var msg = $context.currentItem.previousItem;
 2 var srcs = msg.getResponse(msg.RESPONSE_TEXT_AS_HTML, "//@src");
 3 var response = msg.getResponse();
 4
 5 // $context.result.postMessage($context.result.LEVEL_INFO, "response: " +response);
 6
 7 // count the number of '<img src=' matches are in the response
 8 var count_matches = response.split("<img src=").length-1;
 9 $context.result.postMessage($context.result.LEVEL_INFO, ("<img src matches: " +count_matches));
10
11 // split the array each time '<img src="' is found
12 var array_of_images = response.split("<img src=\"");
13
14 //since the first element of the array (prior to the first img src) is not interesting, delete it fr
15 // array_name.splice(indexNumber,numberOfElementsToRemove
16 array_of_images.splice(0, 1);
17
18 var array_of_images_cleaned = new Array();
19
20 for (x in array_of_images)
21 {
22 // line below shows each element of array
23 // $context.result.postMessage($context.result.LEVEL_INFO, "img src" + x+ ": " +array_of_images[x]);
24
25         // string to find; hard coded
```

Lines 1-3 declare variables for the message, source images (as parsed via XPath), and the message response.	`var msg = $context.currentItem.previousItem;•`
	`var srcs = msg.getResponse(msg.RESPONSE_TEXT_AS_HTML, "//@src");•`
	`var response = msg.getResponse();`
	`••// $context.result.postMessage($context.result.LEVEL_INFO, "response: " +response);••// count the number of '<img src=' matches are in the response•var count_matches = response.split("<img src=").length-1;•$context.result.postMessage($context.result.LEVEL_INFO, ("<img src matches: " +count_matches));••// split the array each time '<img src="' is found•var array_of_images = response.split("<img src=\"");•`

Lines 8-17 declare a variable to count the matches and creates an array of images.	`var count_matches = response.split("<img src=").length-1;`
	`$context.result.postMessage($context.result.LEVEL_INFO, ("<img src matches: " +count_matches));`
	
	`// split the array each time '<img src="' is found`
	`var array_of_images = response.split("<img src=\"");`
	
	`//since the first element of the array (prior to the first img src) is not interesting, delete it from the array`
	`// array_name.splice(indexNumber,numberOfElementsToRemove`
	`array_of_images.splice(0, 1);`

The remaining lines create an `array_of_images`, and then defines logic to distinguish between PNG and JPG images, (as well as to present a list of IMG SRC links in the file). Pertinent Set debug information for results.

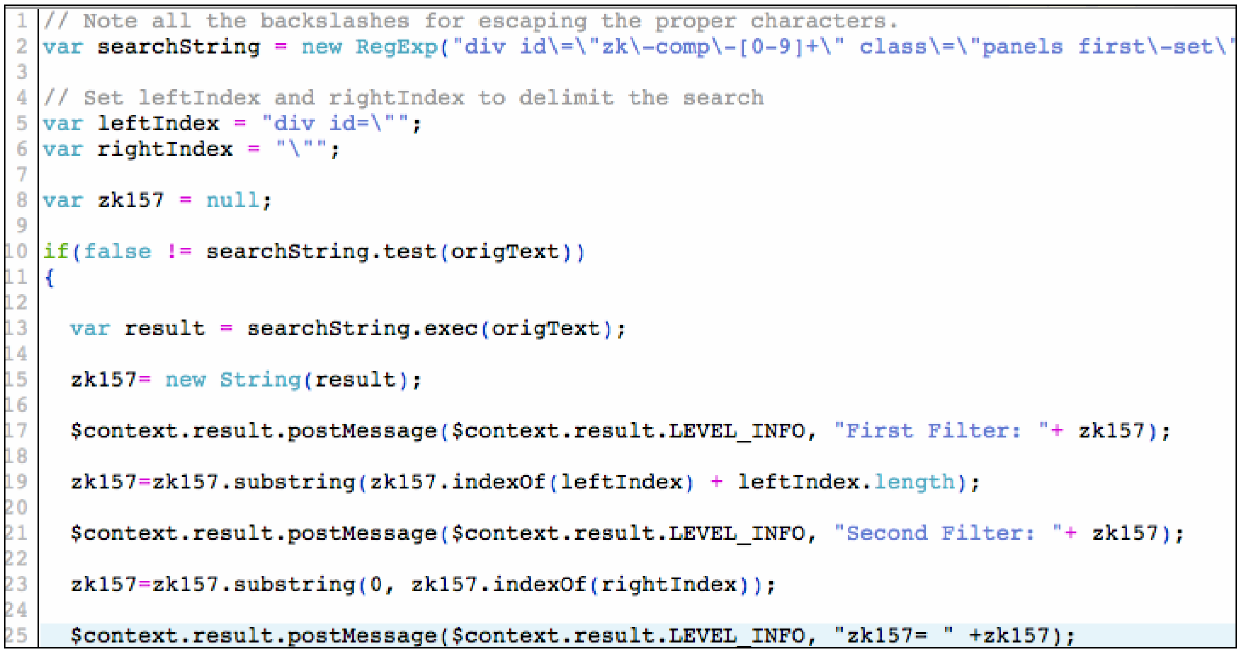
```
var array_of_images_cleaned = new Array();

for (x in array_of_images)
{
// line below shows each element of array
// $context.result.postMessage($context.result.LEVEL_INFO, "img
src" + x+ ": " +array_of_images[x]);

       // string to find; hard coded
              var stringToFind = ".png";

       // Determine if the value exists in the response.
              var searchIndex =
array_of_images[x].indexOf(stringToFind);

       // Determine if the value exists in the response. A -1
Index indicates the
// value was not found in the response. A positive integer
indicates where
// in the response the value was found.

              if (searchIndex == -1)
              {
              //
$context.result.postMessage($context.result.LEVEL_INFO, ".png not
found ");
              // string not found, so look for a .jpg

                     // string to find; hard coded
                     var stringToFind = ".jpg";

              // Determine if the value exists in the response.
                     var searchIndex =
array_of_images[x].indexOf(stringToFind);

                     if (searchIndex == -1)
                     {
                     // string not found


       $context.result.postMessage($context.result.LEVEL_INFO,
".jpg not found either ");
                     }
              // If the value is found, output a confirmation

                     else {

       //$context.result.postMessage($context.result.LEVEL_INFO,
".jpg found");

       //$context.result.postMessage($context.result.LEVEL_INFO,
"searchIndex: "+searchIndex);
                     final_url = array_of_images[x].substring(0,
searchIndex+4);

       //$context.result.postMessage($context.result.LEVEL_INFO,
"final_url: "+final_url);


                     array_of_images_cleaned.push(final_url);

                     }

              }

       // If the value is found, output a confirmation
              else {
              //
$context.result.postMessage($context.result.LEVEL_INFO, ".png
```

Script 12: Extract a Substring Using Regex in JavaScript

The Extract a Substring Using Regular Expression script creates some complex Substrings using regular expression. The value to extract comes from the “ZK - Direct RIA“ Ajax library. The relevant value to extract is shown in **BOLD** in the following sample code:

```
div id="zk-comp-154" class="panels first-set"><div id="zk-comp-155"
class="es-tab-panel"><div id="zk-comp-156" class="es-scrollable-tall"><div
id="zk-comp-157"
```

Since the example above contains more than one dynamic value everything with "zk-comp" is extracted in the subsequent script below.



```
 1 // Note all the backslashes for escaping the proper characters.
 2 var searchString = new RegExp("div id\=\"zk\-comp\-[0-9]+\" class\=\"panels first\-set\"
 3
 4 // Set leftIndex and rightIndex to delimit the search
 5 var leftIndex = "div id=\"";
 6 var rightIndex = "\"";
 7
 8 var zk157 = null;
 9
10 if(false != searchString.test(origText))
11 {
12
13   var result = searchString.exec(origText);
14
15   zk157= new String(result);
16
17   $context.result.postMessage($context.result.LEVEL_INFO, "First Filter: "+ zk157);
18
19   zk157=zk157.substring(zk157.indexOf(leftIndex) + leftIndex.length);
20
21   $context.result.postMessage($context.result.LEVEL_INFO, "Second Filter: "+ zk157);
22
23   zk157=zk157.substring(0, zk157.indexOf(rightIndex));
24
25   $context.result.postMessage($context.result.LEVEL_INFO, "zk157= " +zk157);
```

Line 2 establishes a variable, searchString, from the regular expression to the right of the equal sign.	`var searchString = new RegExp("div id\=\"zk\-comp\-[0-9]+\" class\=\"panels first\-set\"\>\<div id\=\"zk\-comp\-[0-9]+\" class\=\"es\-tab\-panel\"\>\<div id\=\"zk\-comp\-[0-9]+\" class\=\"es\-scrollable\-tall\"\>");`
Lines 5-6 set leftIndex and rightIndex to delimit the search	`var leftIndex = "div id=\"";` `var rightIndex = "\"";`

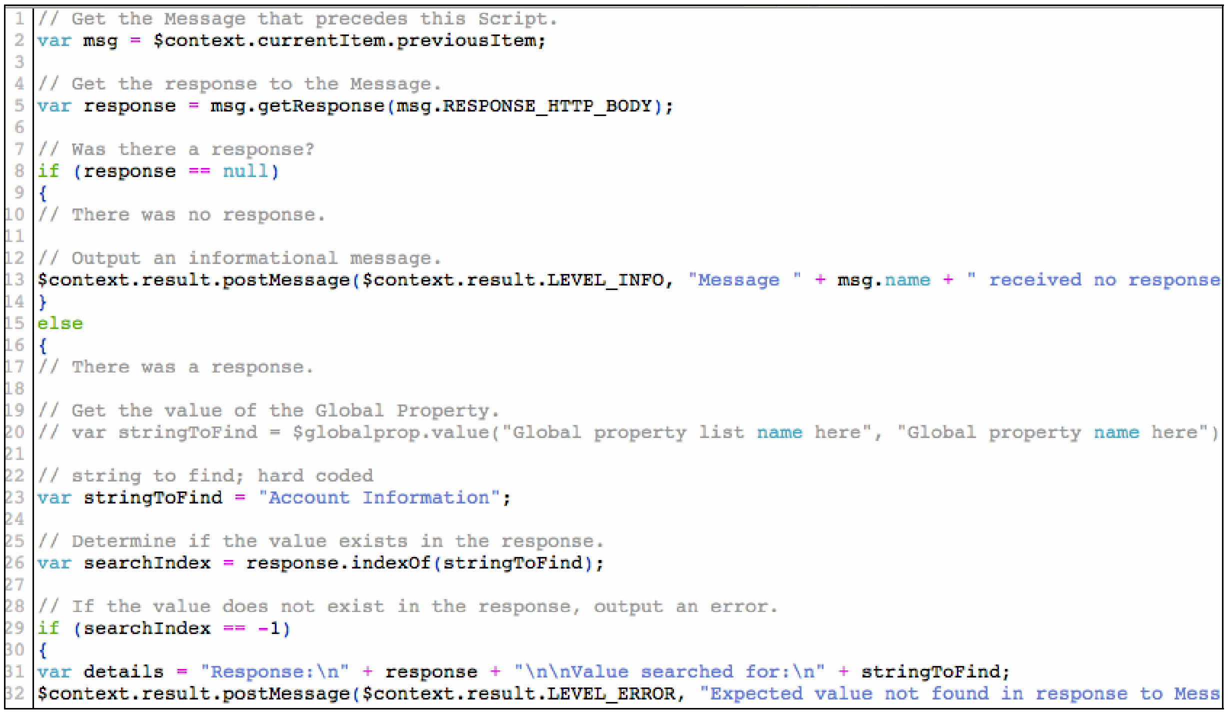
Lines 8-24 perform the extraction using origText and the indices already extracted. Results are posted for each filter.	`var zk157 = null;`
	
	`if(false != searchString.test(origText))`
	`{`
	
	`  var result = searchString.exec(origText);`
	
	`  zk157= new String(result);`
	
	`  $context.result.postMessage($context.result.LEVEL_INFO, "First Filter: "+ zk157);`
	
	`  zk157=zk157.substring(zk157.indexOf(leftIndex) + leftIndex.length);`
	
	`  $context.result.postMessage($context.result.LEVEL_INFO, "Second Filter: "+ zk157);`
	
	`  zk157=zk157.substring(0, zk157.indexOf(rightIndex));`
	`$context.result.postMessage($context.result.LEVEL_INFO, "zk157= " +zk157);`
	`}`

Validation Scripts

This section presents a variety of SOASTA CloudTest validation scripts. Scripts can be copied and pasted directly into the SOASTA CloudTest Script Editor using Central > Scripts. Longer scripts have an accompanying image that can be clicked to pop out the full example script. Shorter scripts are presented inline (in the right column).

Script 1: Throw an Error If a Specified Value Does Not Exist in the Prior Response

The Throw an error if a specified value does not exist in the prior response script provides an easy way to perform text validation.



```
 1 // Get the Message that precedes this Script.
 2 var msg = $context.currentItem.previousItem;
 3 
 4 // Get the response to the Message.
 5 var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);
 6 
 7 // Was there a response?
 8 if (response == null)
 9 {
10 // There was no response.
11 
12 // Output an informational message.
13 $context.result.postMessage($context.result.LEVEL_INFO, "Message " + msg.name + " received no response
14 }
15 else
16 {
17 // There was a response.
18 
19 // Get the value of the Global Property.
20 // var stringToFind = $globalprop.value("Global property list name here", "Global property name here")
21 
22 // string to find; hard coded
23 var stringToFind = "Account Information";
24 
25 // Determine if the value exists in the response.
26 var searchIndex = response.indexOf(stringToFind);
27 
28 // If the value does not exist in the response, output an error.
29 if (searchIndex == -1)
30 {
31 var details = "Response:\n" + response + "\n\nValue searched for:\n" + stringToFind;
32 $context.result.postMessage($context.result.LEVEL_ERROR, "Expected value not found in response to Mess
```

Line 2 gets the message that precedes the script in the test clip.	`var msg = $context.currentItem.previousItem;`
Line 5 declares a response variable that gets the HTTP BODY.	`var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);`

Lines 7-16 set up an "if…else" statement that checks for a null response and post debug information to results if so.

```
// Was there a response?
if (response == null)
{
// There was no response.

// Output an informational message.
$context.result.postMessage($context.result.LEVEL_INFO, "Message
" + msg.name + " received no response to validate.");
}
else
{
```

If there was a response, Line 23 declares a variable, stringToFind, and searches the response body for the value Account Information.

```
var stringToFind = "Account Information";
```

Line 26 declares the variable `searchIndex` that equals the response index of the `stringToFind` from Line 23.

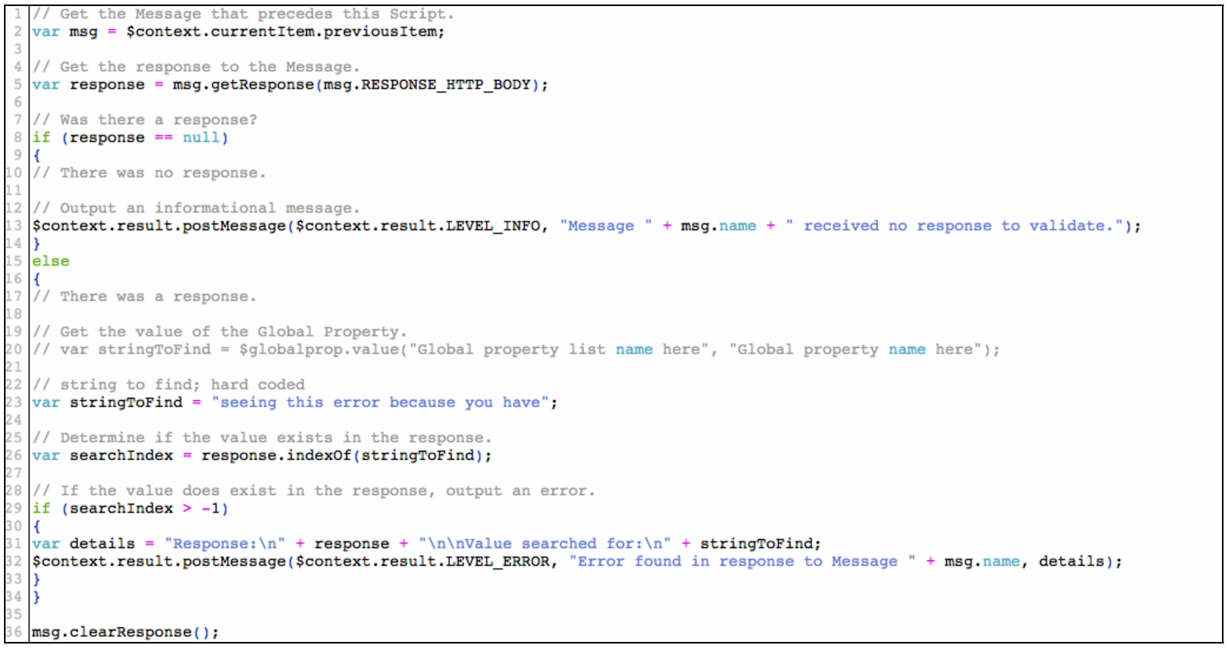
```
var searchIndex = response.indexOf(stringToFind);
```

If the value doesn't exist, Lines 29-34 posts that fact to results. Line 36 clears the response.

```
if (searchIndex == -1)
{
var details = "Response:\n" + response + "\n\nValue searched
for:\n" + stringToFind;
$context.result.postMessage($context.result.LEVEL_ERROR,
"Expected value not found in response to Message " + msg.name,
details);
}
}

msg.clearResponse();
```

Script 2: Throw an error if a specified value does exist in the prior response

The Throw an error if a specified value does exist in the prior response script will throw an error if the specified text appears in the prior response.



```
// Get the Message that precedes this Script.
var msg = $context.currentItem.previousItem;

// Get the response to the Message.
var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);

// Was there a response?
if (response == null)
{
// There was no response.

// Output an informational message.
$context.result.postMessage($context.result.LEVEL_INFO, "Message " + msg.name + " received no response to validate.");
}
else
{
// There was a response.

// Get the value of the Global Property.
// var stringToFind = $globalprop.value("Global property list name here", "Global property name here");

// string to find; hard coded
var stringToFind = "seeing this error because you have";

// Determine if the value exists in the response.
var searchIndex = response.indexOf(stringToFind);

// If the value does exist in the response, output an error.
if (searchIndex > -1)
{
var details = "Response:\n" + response + "\n\nValue searched for:\n" + stringToFind;
$context.result.postMessage($context.result.LEVEL_ERROR, "Error found in response to Message " + msg.name, details);
}
}

msg.clearResponse();
```

Get the Message that precedes this Script.	`var msg = $context.currentItem.previousItem;`

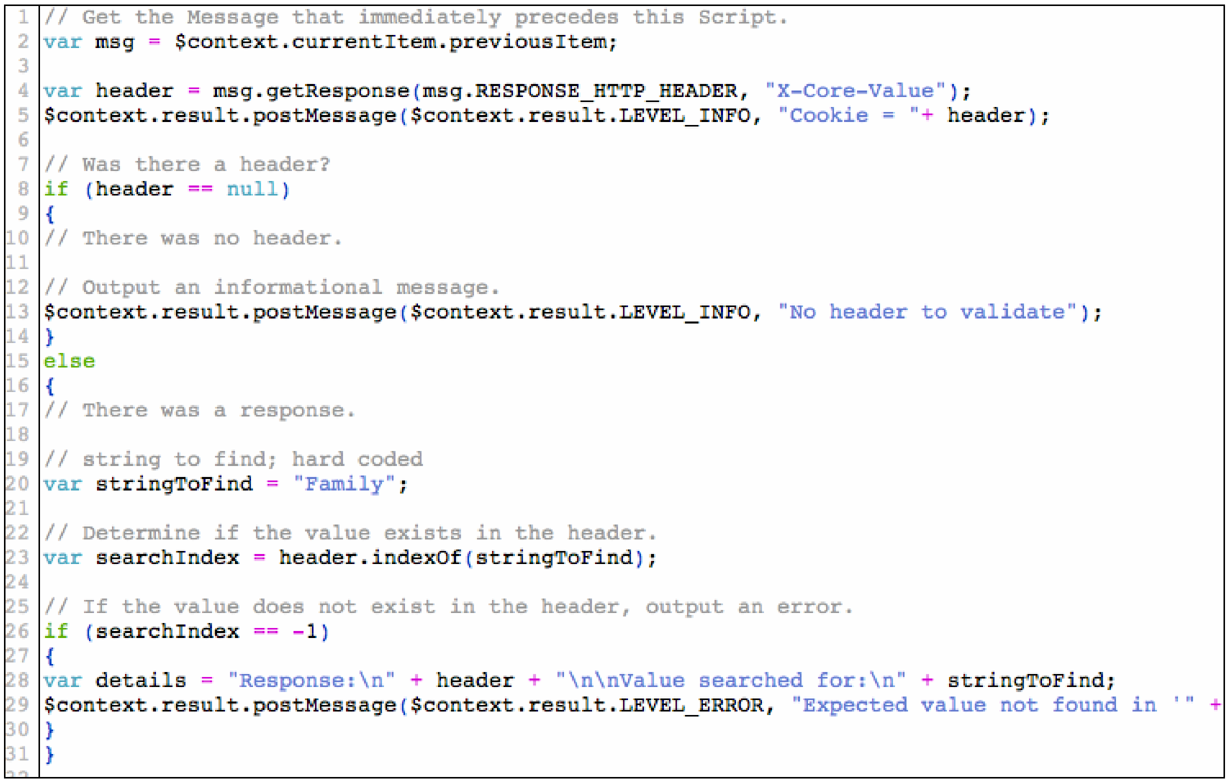
Then, get the response from the HTTP_BODY. If there was no response, output an informational message as such.	`var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);`
	`if (response == null)`
	`$context.result.postMessage($context.result.LEVEL_INFO, "Message " + msg.name + " received no response to validate.");`
	`else`

Then, specify the string to find.	`var stringToFind = "seeing this error because you have";`

Check for the value in the response. If the value does exist in the response, output an error.	`if (searchIndex > -1)`
	`{`
	`var details = "Response:\n" + response + "\n\nValue searched for:\n" + stringToFind;`
	`$context.result.postMessage($context.result.LEVEL_ERROR, "Error found in response to Message " + msg.name, details);`
	`}`
	`}`
	
	`msg.clearResponse();`

Script 3: Throw an Error if a Specified Value Does Not Exist in the Prior Header

The Throw an error if a specified value does not exist in the prior header script looks at the previous message's response header to see if a particular string is present. If the string is present, the validation passes; if not, the validation fails.



```
// Get the Message that immediately precedes this Script.
var msg = $context.currentItem.previousItem;

var header = msg.getResponse(msg.RESPONSE_HTTP_HEADER, "X-Core-Value");
$context.result.postMessage($context.result.LEVEL_INFO, "Cookie = "+ header);

// Was there a header?
if (header == null)
{
// There was no header.

// Output an informational message.
$context.result.postMessage($context.result.LEVEL_INFO, "No header to validate");
}
else
{
// There was a response.

// string to find; hard coded
var stringToFind = "Family";

// Determine if the value exists in the header.
var searchIndex = header.indexOf(stringToFind);

// If the value does not exist in the header, output an error.
if (searchIndex == -1)
{
var details = "Response:\n" + header + "\n\nValue searched for:\n" + stringToFind;
$context.result.postMessage($context.result.LEVEL_ERROR, "Expected value not found in '" +
}
}
```

Get the Message that immediately precedes this Script.	`var msg = $context.currentItem.previousItem;`
Declare a variable, header, and	`var header = msg.getResponse(msg.RESPONSE_HTTP_HEADER, "X-Core-Value");`
Post debug information to results.	`$context.result.postMessage($context.result.LEVEL_INFO, "Cookie = "+ header);`

If there was no header, post debug information to results.	`if (header == null) {`
	`$context.result.postMessage($context.result.LEVEL_INFO, "No`
	`header to validate"); }`

There was a response.	`else`
	`{`

The hard-coded string to find.	`var stringToFind = "Family";`
Determine if the value exists in the header.	`var searchIndex = header.indexOf(stringToFind);`

If the value does not exist in the header, output an error.	`if (searchIndex == -1)`
	`{`
	`var details = "Response:\n" + header + "\n\nValue searched`
	`for:\n" + stringToFind;`
	
	`$context.result.postMessage($context.result.LEVEL_ERROR,`
	`"Expected value not found in '" + msg.name + "' message`
	`header", details);`
	` }`
	`}`
	`msg.clearResponse();`

Script 4: Validate If a String Is Numeric

The Validate if a string is numeric script will validate that a variable (`newText`, in this case) has only numeric characters in it.

Set `newText` equal to a range of substring values.	`newText = newText.substring(0, 7);`

Check for numeric values	`var strValidChars = "0123456789";`
	`   var strChar;`
	`   var blnResult = true;`

Test the variable, `strString`, for the characters in Line 1.	`for (i = 0; i < newText.length && blnResult == true; i++)`
	`{`
	`strChar = newText.charAt(i);`
	`if (strValidChars.indexOf(strChar) == -1)`
	`{`
	`blnResult = false;`

Do something to the non-numeric variable here	`$context.result.postMessage($context.result.LEVEL_INFO, "found not numeric");`
	` }`
	`}`

Error Detection and Handling Scripts

This section presents a variety of SOASTA CloudTest error detection scripts. Scripts can be copied and pasted directly into the SOASTA CloudTest Script Editor using Central > Scripts. Longer scripts have an accompanying image that can be clicked to pop out the full example script. Shorter scripts are presented inline (in the right column).

Script 1: Set All Play counts = 1

If an error is detected, Set all play counts = 1.

Set play count for all messages in a clip to 1 (note that this ignores any messages that might have other repeats set on them).	`var nextItem = $context.currentItem.nextItem;`
	`while (nextItem != null)`
	`{`
	`  nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL,`
	`nextItem.REPEAT_TYPE_COUNT_CONSTANT, 1,`
	`nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);`
	`  nextItem = nextItem.nextItem;`
	`}`

Script 2: Set Play Count for Subsequent Clip Elements to 0 If an Error Is Detected

If an error is detected, Set play count for all subsequent clip elements in a clip to 0.

If serial repeats are used, this requires that all clip elements have a play count set to 1 at the beginning of the clip. Refer to the prior script example for an example.

This example assumes errorDetected is true if there was an error. Disable all following items in the current Clip.	`var errorDetected = true;`
	`if (errorDetected)`
	`{`
	`var nextItem = $context.currentItem.nextItem;`
	`while (nextItem != null)`
	`{`
	`nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL,`
	`nextItem.REPEAT_TYPE_COUNT_CONSTANT, 0,`
	`nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);`
	`nextItem = nextItem.nextItem;`
	` }`
	`}`

Script 3: Stop Current Clip If an Error Is Detected

The following script detects whether an error occurs in a test clip and, if so, stops the clip. If an error is detected, stop the clip. This is a variant on the script above.

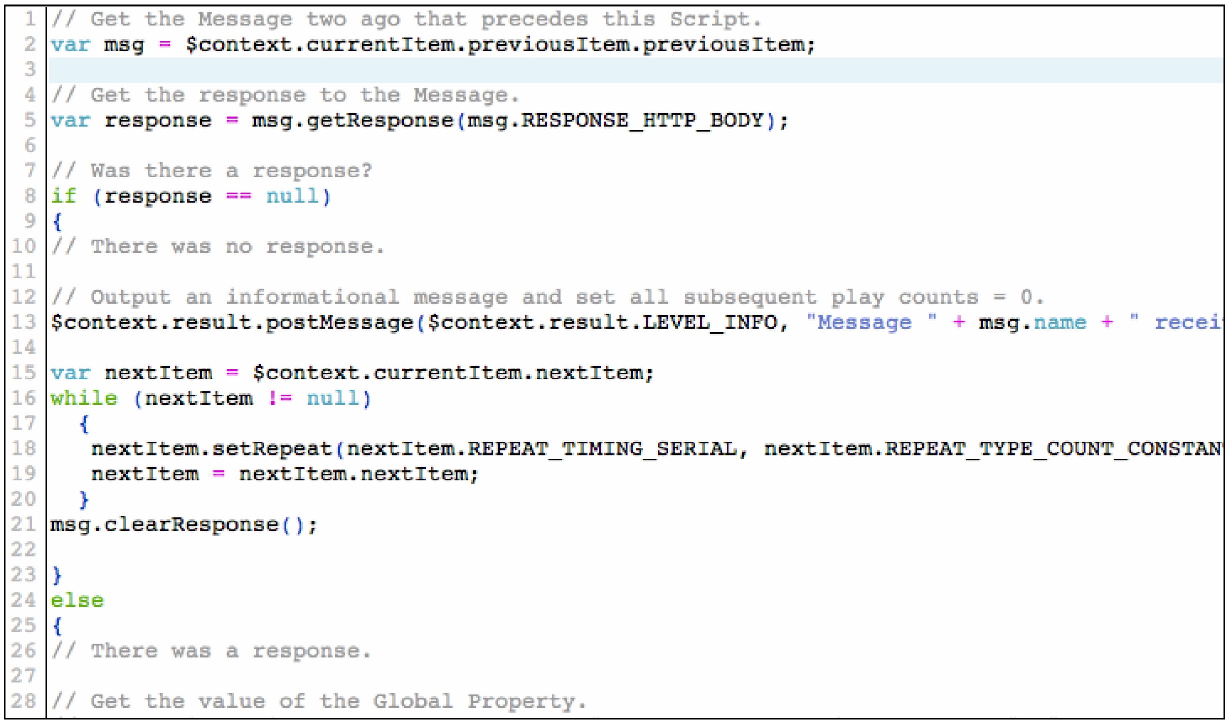
This example assumes "errorDetected" is true if there was an error.	`var errorDetected = true;`

If an error is detected.	`if (errorDetected)`
	`{`

Stop the current test clip.	`$context.currentClip.end();`
	`}`

Script 4: Set All play Counts = 0 if “authid” Is Not returned

Set all play counts in clip = 0 if “authid” is not returned from the login POST.


```
// Get the Message two ago that precedes this Script.
var msg = $context.currentItem.previousItem.previousItem;

// Get the response to the Message.
var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);

// Was there a response?
if (response == null)
{
// There was no response.

// Output an informational message and set all subsequent play counts = 0.
$context.result.postMessage($context.result.LEVEL_INFO, "Message " + msg.name + " recei

var nextItem = $context.currentItem.nextItem;
while (nextItem != null)
  {
   nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL, nextItem.REPEAT_TYPE_COUNT_CONSTAN
   nextItem = nextItem.nextItem;
  }
msg.clearResponse();

}
else
{
// There was a response.

// Get the value of the Global Property.
```

Get the Message two ago that precedes this Script.	`var msg = $context.currentItem.previousItem.previousItem;`
Get the response from HTTP BODY.	`var response = msg.getResponse(msg.RESPONSE_HTTP_BODY);`

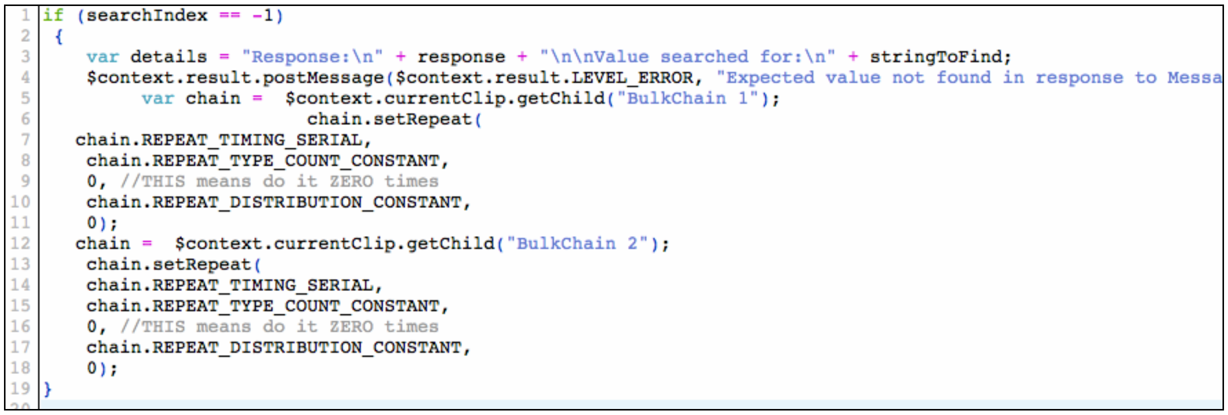
If there was no response, post debug information to results, and set all play counts to 0.	`if (response == null)`
	`{`
	`$context.result.postMessage($context.result.LEVEL_INFO, "Message " + msg.name + " received no response to validate.");`
	
	`var nextItem = $context.currentItem.nextItem;`
	`while (nextItem != null)`
	`  {`
	`   nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL, nextItem.REPEAT_TYPE_COUNT_CONSTANT, 0, nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);`
	`   nextItem = nextItem.nextItem;`
	`  }`
	`msg.clearResponse();`
	
	`}`
	`else`
	`{`

If there was a response, specify the hard-coded string to find.	var stringToFind = "authid";
Determine if the response string exists.	var searchIndex = response.indexOf(stringToFind);

If the value doesn't exist in the response, post an error to results, and set all subsequent requests to play count=0.	if (searchIndex == -1)
	{
	var details = "Response:\n" + response + "\n\nValue searched for:\n" + stringToFind;
	$context.result.postMessage($context.result.LEVEL_ERROR, "Login failed because authid not found " + msg.name, details);
	var nextItem = $context.currentItem.nextItem;
	while (nextItem != null)
	{
	nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL, nextItem.REPEAT_TYPE_COUNT_CONSTANT, 0, nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);
	nextItem = nextItem.nextItem;
	}
	msg.clearResponse();
	}
	}

Script 5: Disable Chain If Value Not Found in Response

This script can be added to the typical validation script. "BulkChain 1" is the name of the chain. There are 2 chains in the test, and they are named BulkChain 1 and 2. The script can find them by name and set them to a specific value. 0 = it will not play.



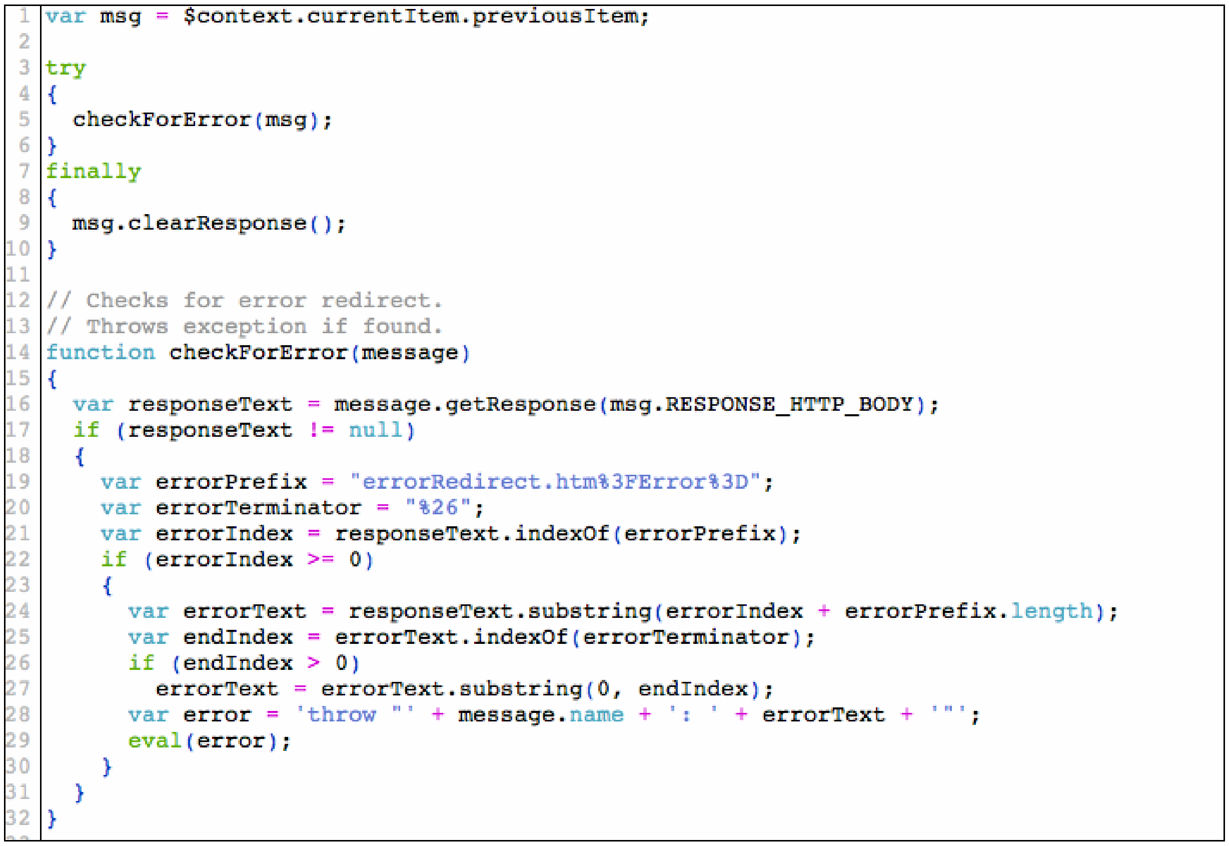
```
if (searchIndex == -1)
 {
    var details = "Response:\n" + response + "\n\nValue searched for:\n" + stringToFind;
    $context.result.postMessage($context.result.LEVEL_ERROR, "Expected value not found in response to Messa
         var chain =  $context.currentClip.getChild("BulkChain 1");
                        chain.setRepeat(
   chain.REPEAT_TIMING_SERIAL,
    chain.REPEAT_TYPE_COUNT_CONSTANT,
    0, //THIS means do it ZERO times
    chain.REPEAT_DISTRIBUTION_CONSTANT,
    0);
   chain =  $context.currentClip.getChild("BulkChain 2");
    chain.setRepeat(
    chain.REPEAT_TIMING_SERIAL,
    chain.REPEAT_TYPE_COUNT_CONSTANT,
    0, //THIS means do it ZERO times
    chain.REPEAT_DISTRIBUTION_CONSTANT,
    0);
}
```

This example assumes `errorDetected` is true if there was an error. Disable all following items in the current Clip.

```
var errorDetected = true;
if (errorDetected)
{
var nextItem = $context.currentItem.nextItem;
while (nextItem != null)
{
nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL,
nextItem.REPEAT_TYPE_COUNT_CONSTANT, 0,
nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);
nextItem = nextItem.nextItem;
 }
}
```

Script 6: Check for ErrorRedirect Response

The Check for ErrorRedirect response script checks the body of a response for errors and throws the error and message associated with the error.



```
var msg = $context.currentItem.previousItem;

try
{
  checkForError(msg);
}
finally
{
  msg.clearResponse();
}

// Checks for error redirect.
// Throws exception if found.
function checkForError(message)
{
  var responseText = message.getResponse(msg.RESPONSE_HTTP_BODY);
  if (responseText != null)
  {
    var errorPrefix = "errorRedirect.htm%3FError%3D";
    var errorTerminator = "%26";
    var errorIndex = responseText.indexOf(errorPrefix);
    if (errorIndex >= 0)
    {
      var errorText = responseText.substring(errorIndex + errorPrefix.length);
      var endIndex = errorText.indexOf(errorTerminator);
      if (endIndex > 0)
        errorText = errorText.substring(0, endIndex);
      var error = 'throw "' + message.name + ': ' + errorText + '"';
      eval(error);
    }
  }
}
```

Set the context to the preceding message.	`var msg = $context.currentItem.previousItem;`
Check for errors using try/catch and clear the response.	`try { checkForError(msg); } finally { msg.clearResponse(); }`

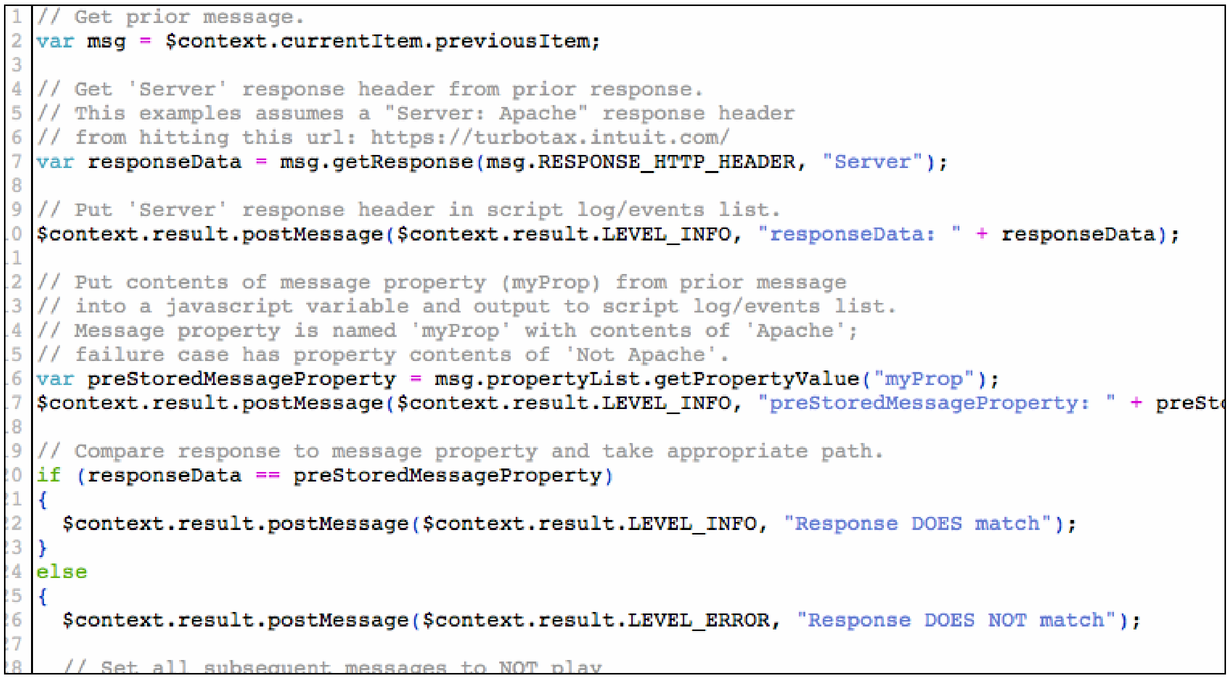
Check for error redirect and throw an exception if found.

```
function checkForError(message)
{
  var responseText =
message.getResponse(msg.RESPONSE_HTTP_BODY);
  if (responseText != null)
  {
    var errorPrefix = "errorRedirect.htm%3FError%3D";
    var errorTerminator = "%26";
    var errorIndex = responseText.indexOf(errorPrefix);
    if (errorIndex >= 0)
    {
      var errorText = responseText.substring(errorIndex +
errorPrefix.length);
      var endIndex = errorText.indexOf(errorTerminator);
      if (endIndex > 0)
        errorText = errorText.substring(0, endIndex);
      var error = 'throw "' + message.name + ': ' + errorText +
'"';
      eval(error);
    }
  }
}
```

Script 7: Compare Response to Message Property and Take Different Actions on Result (Enable ErrorHandling Chain If Error Occurs)

The Compare response to message property and take different actions on result (enable ErrorHandling chain if error occurs) script checks for data that has been previously entered into a message property (in this case something from a message header). This script structure is useful for error handling. If the response matches, continue with the test. If the response doesn't match the expected response, output an error or stop the test. This approach for error handling is powerful because it doesn't rely on hard-coded values in the script. Instead, the matching is done on variables placed in individual messages.

This approach for error handling is powerful because it doesn't rely on hard-coded values in the script. Instead, the matching is done on variables placed in individual messages.



```
// Get prior message.
var msg = $context.currentItem.previousItem;

// Get 'Server' response header from prior response.
// This examples assumes a "Server: Apache" response header
// from hitting this url: https://turbotax.intuit.com/
var responseData = msg.getResponse(msg.RESPONSE_HTTP_HEADER, "Server");

// Put 'Server' response header in script log/events list.
$context.result.postMessage($context.result.LEVEL_INFO, "responseData: " + responseData);

// Put contents of message property (myProp) from prior message
// into a javascript variable and output to script log/events list.
// Message property is named 'myProp' with contents of 'Apache';
// failure case has property contents of 'Not Apache'.
var preStoredMessageProperty = msg.propertyList.getPropertyValue("myProp");
$context.result.postMessage($context.result.LEVEL_INFO, "preStoredMessageProperty: " + preSt

// Compare response to message property and take appropriate path.
if (responseData == preStoredMessageProperty)
{
  $context.result.postMessage($context.result.LEVEL_INFO, "Response DOES match");
}
else
{
  $context.result.postMessage($context.result.LEVEL_ERROR, "Response DOES NOT match");

  // Set all subsequent messages to NOT play
```

Set the context to the prior message.	`var msg = $context.currentItem.previousItem;`

Get 'Server' response header from prior response. This example assumes a "Server: Apache" response header	`var responseData = msg.getResponse(msg.RESPONSE_HTTP_HEADER,`
	`"Server");`

Put the 'Server' response header in script log/events list.	`$context.result.postMessage($context.result.LEVEL_INFO,`
	`"responseData: " + responseData);`

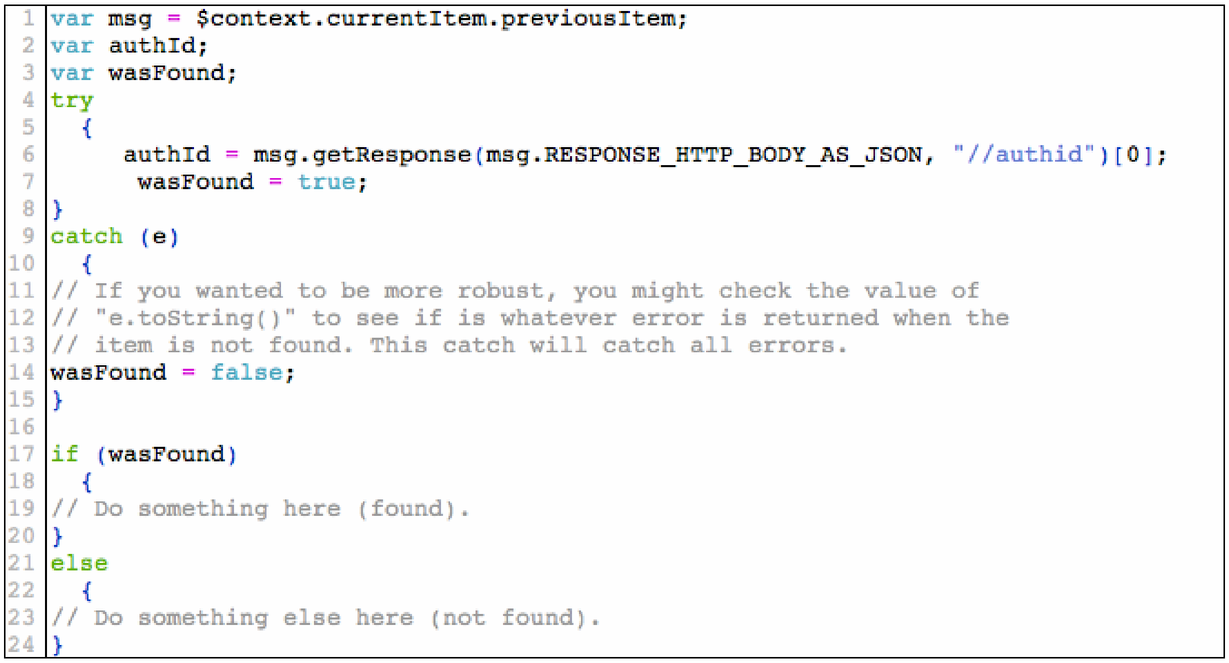
Put contents of message property (`myProp`) from prior message into a variable and output to script log/events list. Message property is named '`myProp`' with contents of 'Apache'; failure case has property contents of 'Not Apache'.	`var preStoredMessageProperty =`
	`msg.propertyList.getPropertyValue("myProp");`
	
	`$context.result.postMessage($context.result.LEVEL_INFO,`
	`"preStoredMessageProperty: " + preStoredMessageProperty);`

Compare response to message property and take appropriate path.	`if (responseData == preStoredMessageProperty)`
	
	`{`
	
	`  $context.result.postMessage($context.result.LEVEL_INFO,`
	`"Response DOES match");`
	
	`}`
	
	`else`
	
	`{`
	
	`  $context.result.postMessage($context.result.LEVEL_ERROR,`
	`"Response DOES NOT match");`

Set all subsequent messages to NOT play.	`var nextItem = $context.currentItem.nextItem;` `while (nextItem != null)` `{` `nextItem.setRepeat(nextItem.REPEAT_TIMING_SERIAL, nextItem.REPEAT_TYPE_COUNT_CONSTANT, 0, nextItem.REPEAT_DISTRIBUTION_CONSTANT, 0);` `nextItem = nextItem.nextItem;` `}`
Enable an "`ErrorHandling`" chain. **Note:** The `ErrorHandling` chain was disabled in the clip by default (i.e. serial repeat = 0) Clear the message response from memory since it is no longer needed.	`chain =  $context.currentClip.getChild("ErrorHandling");` `chain.setRepeat(` `chain.REPEAT_TIMING_SERIAL,` `chain.REPEAT_TYPE_COUNT_CONSTANT,` `1, // this means do it ONE time` `chain.REPEAT_DISTRIBUTION_CONSTANT,` `0);` `}` `msg.clearResponse();`

Script 8: Try/Catch Example

Use this brief example script to develop an example of try/catch logic.



```
var msg = $context.currentItem.previousItem;
var authId;
var wasFound;
try
  {
     authId = msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_JSON, "//authid")[0];
      wasFound = true;
}
catch (e)
  {
// If you wanted to be more robust, you might check the value of
// "e.toString()" to see if is whatever error is returned when the
// item is not found. This catch will catch all errors.
wasFound = false;
}

if (wasFound)
  {
// Do something here (found).
}
else
  {
// Do something else here (not found).
}
```

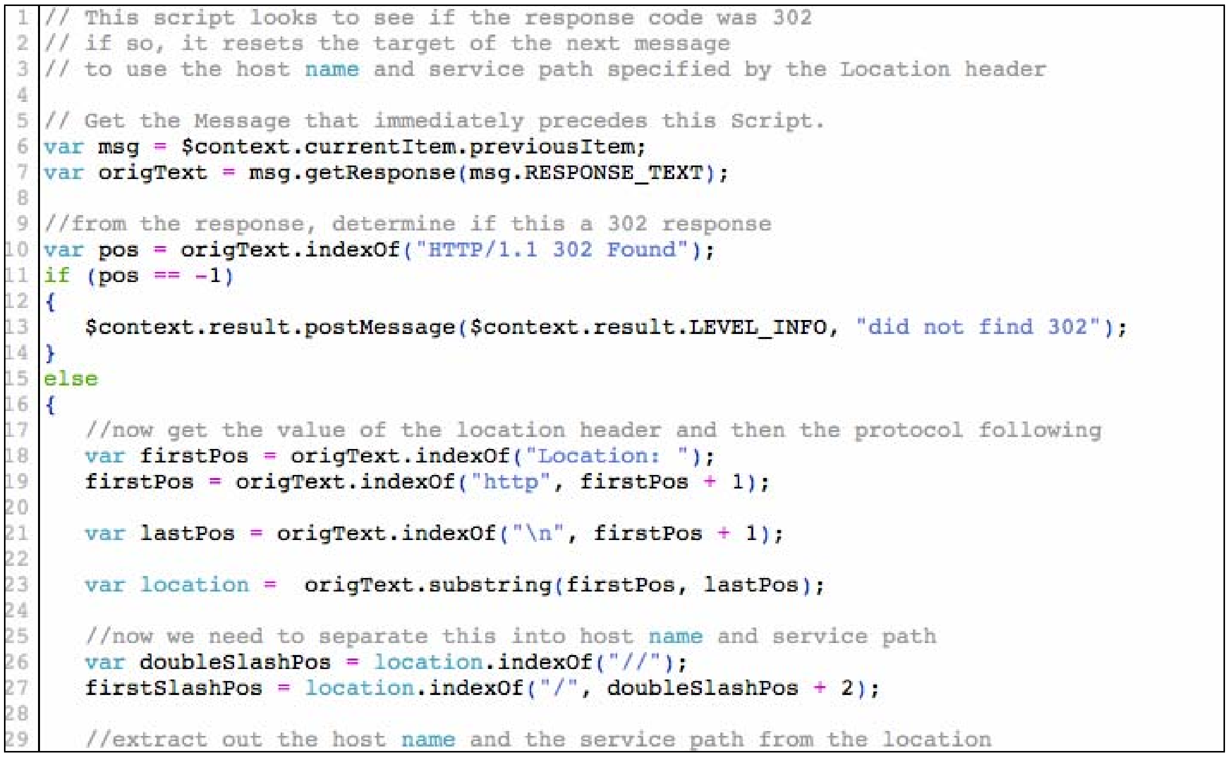
Use try/catch to search a JSON message response searching for authid.	`var msg = $context.currentItem.previousItem;`
	`var authId;`
	`var wasFound;`
	`try`
	`   {`
	`        authId =`
	`msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_JSON,`
	`"//authid")[0];`
	`         wasFound = true;`
	`}`
	`catch (e)`
	`   {`

Optionally, check the value of "e.toString()" to see if is whatever error is returned when the item is not found. This catch will catch all errors.	`wasFound = false;`
	`}`
	
	`if (wasFound)`
	`   {`
	`// Do something here (found).`
	`}`
	`else`
	`   {`
	`// Do something else here (not found).`
	`}`

Target/Hostname Modification Scripts

Script 1: Handle Dynamic HTTP 302 Redirects

The Handle dynamic HTTP 302 redirects script is used to detect a redirect (HTTP 302) in the test and set the next message's target so subsequent requests go to that new hostname. This script looks to see if the response code was 302. If so, it resets the target of the next message to use the host name and service path specified by the Location header



```
// This script looks to see if the response code was 302
// if so, it resets the target of the next message
// to use the host name and service path specified by the Location header

// Get the Message that immediately precedes this Script.
var msg = $context.currentItem.previousItem;
var origText = msg.getResponse(msg.RESPONSE_TEXT);

//from the response, determine if this a 302 response
var pos = origText.indexOf("HTTP/1.1 302 Found");
if (pos == -1)
{
   $context.result.postMessage($context.result.LEVEL_INFO, "did not find 302");
}
else
{
   //now get the value of the location header and then the protocol following
   var firstPos = origText.indexOf("Location: ");
   firstPos = origText.indexOf("http", firstPos + 1);

   var lastPos = origText.indexOf("\n", firstPos + 1);

   var location =  origText.substring(firstPos, lastPos);

   //now we need to separate this into host name and service path
   var doubleSlashPos = location.indexOf("//");
   firstSlashPos = location.indexOf("/", doubleSlashPos + 2);

   //extract out the host name and the service path from the location
```

Get the message that precedes this script.	`var origText = msg.getResponse(msg.RESPONSE_TEXT);`

Determine if the response is an HTTP 302 redirect.	`var pos = origText.indexOf("HTTP/1.1 302 Found");`
	`if (pos == -1)`
	`{`
	`   $context.result.postMessage($context.result.LEVEL_INFO, "did not find 302");`
	`}`
	`else`
	`{`

Get the value of the Location header and the following protocol.	`var firstPos = origText.indexOf("Location: ");` `firstPos = origText.indexOf("http", firstPos + 1);` `var lastPos = origText.indexOf("\n", firstPos + 1);` `var location =  origText.substring(firstPos, lastPos);`
Separate the host name and service path.	`var doubleSlashPos = location.indexOf("//");` `firstSlashPos = location.indexOf("/", doubleSlashPos + 2);`
Extract the host name and the service path from the location.	`var hostName    = location.substring(doubleSlashPos + 2, firstSlashPos);` `var servicePath = location.substring(firstSlashPos);`
Set the host name and service path for the target of the next message.	`$context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("HostName", hostName);` `$context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("ServicePath", servicePath);` `}`

Script 2: Override a Target’s URL for the Instance of a Clip

The change this script makes persists across serial repeats of a clip. This script sets the target for the message immediately after this script. Since it is setting the target’s URL, all messages that use that target use this new URL.

```
var hostname = 'new_hostname.company.com';
$context.result.postMessage($context.result.LEVEL_INFO, "hostname: " + hostname);
$context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("HostName",
hostname);
```

Script 3: Override a target’s use of HTTP/HTTPS for the instance of a clip

This script overrides a target to either use or not use SSL (i.e. HTTP to HTTPS or vice versa).

```
$context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("UseSSL",
“true”);
```

Script 4: Host Override in Header

In some situations when an application is deployed to a non-production environment, an IP address must be used to reach that environment (i.e. the target location is an IP address). However, the application may only respond to requests where the HTTP Request Headers include the correct hostname in the host header. This script alters the host header to include the appropriate hostname, but sends the message to the IP address. The alternative to using this script would be entering a HOSTS file entry manually on each load server.

Put the one-line script in the test clip just before the first message that utilizes the target that needs to have the host override header in it. More information can be found in the FAQ, How do I override the value of the HTTP host header in requests?

```
$context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("HttpHostOv
erride", "www.optimizedennys.com");
```

Miscellaneous Scripts

Script 1: Stop Test Clips (and Optionally Clip Repeats) via Script

In those cases where you need a script to skip playing the remainder of the current repeat of the current test clip, you would do:

```
$context.currentClip.end();
```

If the clip was serially repeating and you want to end it completely, including stopping it from repeating any further, you would do:

```
$context.currentClip.end();
$context.currentClip.endRepeat();
```

If the clip was serially repeating and you wanted the current repeat to finish, but not repeat any more after that, you would do:

```
$context.currentClip.endRepeat();
```

Optionally, if you want to cause the container to error out, supply error text like in the following example:

```
$context.currentClip.end("There was an Error");
```

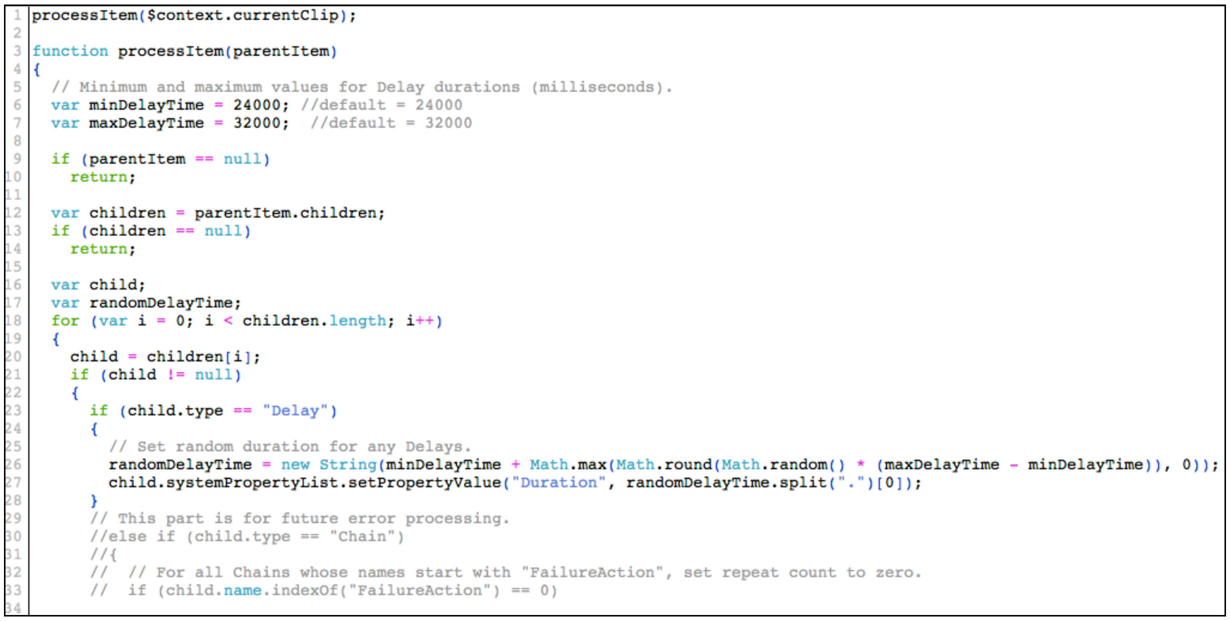
Script 2: Clear Response from Prior Message

The last line of this script (`msg.clearResponse();`) is the main part of this script. It is useful for ensuring that large responses do not stay in memory if they are no longer needed.

```
var msg = $context.currentItem.previousItem;
msg.clearResponse();
```

Script 3: Random Think Times

The Random Think Times script resets all subsequent delays to have a random think time between the interval specified in the script below. If you want to set all delays to zero, just set both the minimum and maximum delay times to zero.



```
 1 processItem($context.currentClip);
 2
 3 function processItem(parentItem)
 4 {
 5   // Minimum and maximum values for Delay durations (milliseconds).
 6   var minDelayTime = 24000; //default = 24000
 7   var maxDelayTime = 32000;  //default = 32000
 8
 9   if (parentItem == null)
10     return;
11
12   var children = parentItem.children;
13   if (children == null)
14     return;
15
16   var child;
17   var randomDelayTime;
18   for (var i = 0; i < children.length; i++)
19   {
20     child = children[i];
21     if (child != null)
22     {
23       if (child.type == "Delay")
24       {
25         // Set random duration for any Delays.
26         randomDelayTime = new String(minDelayTime + Math.max(Math.round(Math.random() * (maxDelayTime - minDelayTime)), 0));
27         child.systemPropertyList.setPropertyValue("Duration", randomDelayTime.split(".")[0]);
28       }
29       // This part is for future error processing.
30       //else if (child.type == "Chain")
31       //{
32       //  // For all Chains whose names start with "FailureAction", set repeat count to zero.
33       //  if (child.name.indexOf("FailureAction") == 0)
34
```

Line 1 creates a currentClip context for a function, `processItem`, that is defined in the subsequent lines of the script.	`processItem($context.currentClip);`
Line 3 creates the function, `processItem`, which is then defined in the remainder of this script.	`function processItem(parentItem)`

Lines 6-7 define the minimum and maximum delay times that bound the think times.	`var minDelayTime = 24000; //default = 24000`
	`var maxDelayTime = 32000;  //default = 32000`

Lines 9-24 specify that if the `parentItem` is not null then the variables, `children` and `randomDelayTime`, will be em-ployed according to the formula on line 21.	`if (parentItem == null)`
	`    return;`
	
	`  var children = parentItem.children;`
	`  if (children == null)`
	`    return;`
	
	`  var child;`
	`  var randomDelayTime;`
	`  for (var i = 0; i < children.length; i++)`
	`  {`
	`    child = children[i];`
	`    if (child != null)`
	`    {`
	`      if (child.type == "Delay")`
	`      {`

Lines 26-27 set the random duration for any delays.	`randomDelayTime = new String(minDelayTime + Math.max(Math.round(Math.random() * (maxDelayTime - minDelayTime)), 0));`
	`        child.systemPropertyList.setPropertyValue("Duration", randomDelayTime.split(".")[0]);`

Script 4: Abort a Script and Consider It an Error with Custom Error Text

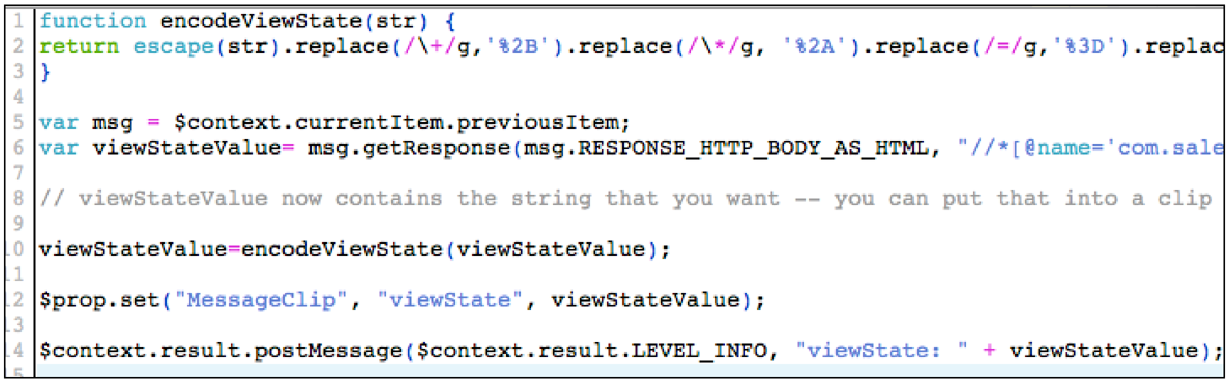
This short script immediately aborts play of the current Script, and considers the Script to have failed with the given error text.

If no error text is provided, the text "Script aborted." is used. So, for example, in those cases where the eval/throw statement is used (such as in the Validation Scripts above), you could instead do:

```
$context.abortScript("Error text here");
```

Script 5: Encoding Text

The Encoding Text script exemplifies how to URL encode text. This is useful in cases where certain characters like a plus sign (+) need to be encoded to go on a URL – as %2B. This script is also useful because it shows how to create a function that performs the encoding. This function can be called multiple times in the script.



```
function encodeViewState(str) {
return escape(str).replace(/\+/g,'%2B').replace(/\*/g, '%2A').replace(/=/g,'%3D').replac
}

var msg = $context.currentItem.previousItem;
var viewStateValue= msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='com.sale

// viewStateValue now contains the string that you want -- you can put that into a clip

viewStateValue=encodeViewState(viewStateValue);

$prop.set("MessageClip", "viewState", viewStateValue);

$context.result.postMessage($context.result.LEVEL_INFO, "viewState: " + viewStateValue);
```

Lines 1-3 create a function, `encodeViewStart(str)`, and then uses a regular expression to perform that encoding.	`function encodeViewState(str) {`
	`return escape(str).replace(/\+/g,'%2B').replace(/\*/g,'%2A').replace(/=/g,'%3D').replace(/:/g,'%3A').replace(/;/g,'%3B').replace(/\//g, '%2F');`
	`}`

Line 5 sets the context of msg to the `previousItem`, while line 6 defines the extraction path from the HTML response.	`var msg = $context.currentItem.previousItem;`
	`var viewStateValue= msg.getResponse(msg.RESPONSE_HTTP_BODY_AS_HTML, "//*[@name='com.salesforce.visualforce.ViewState']/@value")[0];`

Line 10 specifies that `viewStateValue` is equal on the encoded string.	`viewStateValue=encodeViewState(viewStateValue);`

Line 12 sets a property to store the encoded value and line 14 posts a message to results.	`$prop.set("MessageClip", "viewState", viewStateValue);`
	
	`$context.result.postMessage($context.result.LEVEL_INFO, "viewState: " + viewStateValue);`

Another example of using the replace function to do encoding is presented below. This second script uses different statements for each replacement rather than combining them together (as the first example shows).

```
var msg = $context.currentItem.previousItem;
var origText = msg.getResponse(msg.RESPONSE_TEXT);
var newText = origText.substring(pos + 6);

newText = newText.replace("%2F","/","g");
newText = newText.replace("+","%2B","g");
newText = newText.replace("%3D","=","g");

$prop.set("MessageClip", "ID", newText);
```

Script 6: Math Calculations using "ISSE" Expressions

The example script below is not run in a separate script object – it is run directly in the query string parameter of a message. In this example, it is adding the VUNumber property value to 500. Note: all properties are a string, so the property needs to be converted to a number before it can be used in math calculations.

```
{%%expr:
Math.floor(Number($context.currentTrack.systemPropertyList.getPropertyValue(
"VUNumber")) + 500).toString() %%}
```

Script 7: Extract All Links from a Given Response

This brief script creates an array with all of the values of the HREF tags in the response.

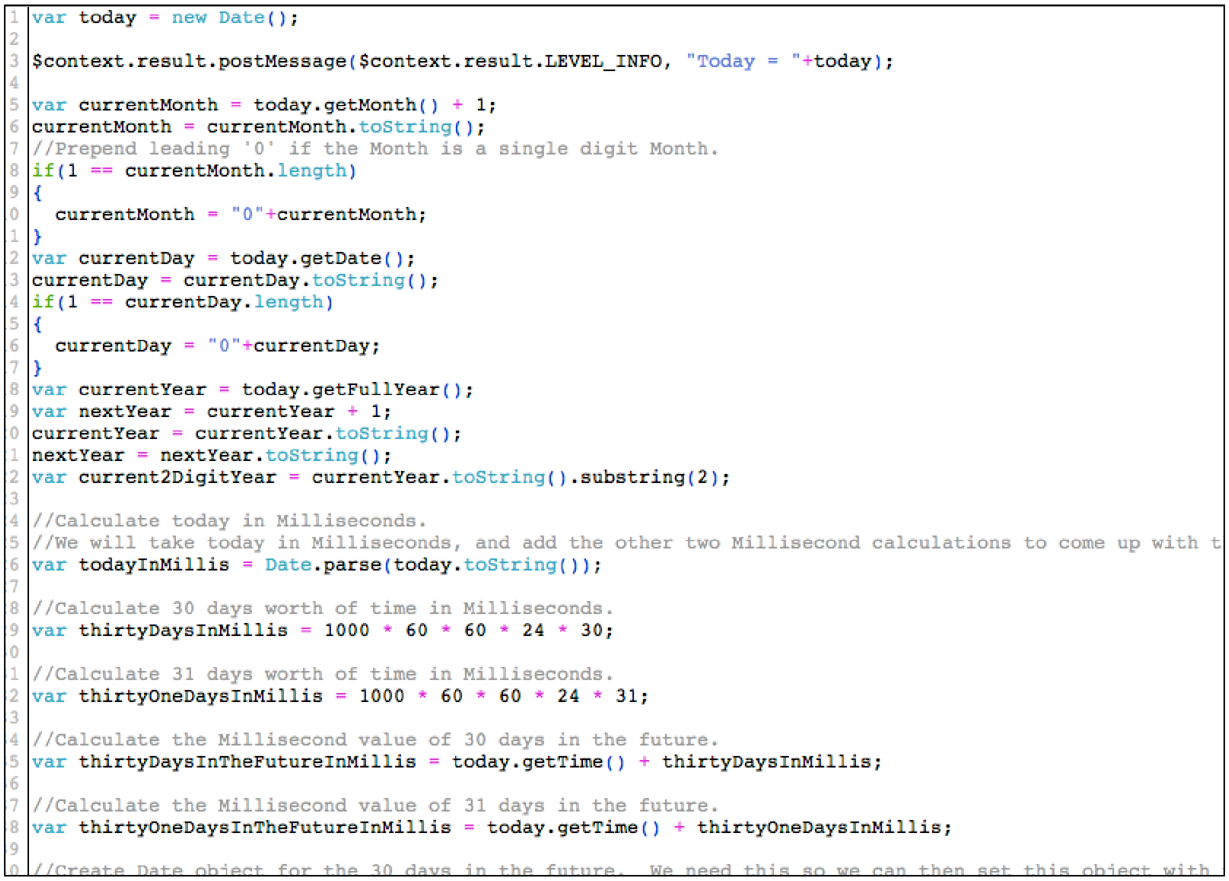
```
var links = msg.getResponse(msg.RESPONSE_TEXT_AS_HTML, "//@href");
var details = links.length +" "+ links.join();
$context.result.postMessage($context.result.LEVEL_INFO, "Links", details);
```

Note that Line 2 above prints the values in the result as a comma-separated list. Alternately, this code could be used in place of line 2:

```
for(var i = 0; i < links.length; i++)
{
  var text = links[i];
  if(text.charAt(0) != '#')
  {
    $context.result.postMessage($context.result.LEVEL_INFO, text);
  }
}
```

Script 8: Determine Dates 30 and 31 Days from Now

The Determine dates 30 and 31 days from now script calculates today's date in the individual pieces of month, day, and year in both 2 and 4 digits. It then calculates two other dates in individual parts based on today's date. These two dates are 30 and 31 days into the future from today's date.



```
1 var today = new Date();
2
3 $context.result.postMessage($context.result.LEVEL_INFO, "Today = "+today);
4
5 var currentMonth = today.getMonth() + 1;
6 currentMonth = currentMonth.toString();
7 //Prepend leading '0' if the Month is a single digit Month.
8 if(1 == currentMonth.length)
9 {
0   currentMonth = "0"+currentMonth;
1 }
2 var currentDay = today.getDate();
3 currentDay = currentDay.toString();
4 if(1 == currentDay.length)
5 {
6   currentDay = "0"+currentDay;
7 }
8 var currentYear = today.getFullYear();
9 var nextYear = currentYear + 1;
0 currentYear = currentYear.toString();
1 nextYear = nextYear.toString();
2 var current2DigitYear = currentYear.toString().substring(2);
3
4 //Calculate today in Milliseconds.
5 //We will take today in Milliseconds, and add the other two Millisecond calculations to come up with t
6 var todayInMillis = Date.parse(today.toString());
7
8 //Calculate 30 days worth of time in Milliseconds.
9 var thirtyDaysInMillis = 1000 * 60 * 60 * 24 * 30;
0
1 //Calculate 31 days worth of time in Milliseconds.
2 var thirtyOneDaysInMillis = 1000 * 60 * 60 * 24 * 31;
3
4 //Calculate the Millisecond value of 30 days in the future.
5 var thirtyDaysInTheFutureInMillis = today.getTime() + thirtyDaysInMillis;
6
7 //Calculate the Millisecond value of 31 days in the future.
8 var thirtyOneDaysInTheFutureInMillis = today.getTime() + thirtyOneDaysInMillis;
9
0 //Create Date object for the 30 days in the future.  We need this so we can then set this object with
```

Line 1 establishes the variable, `today`, which gets the current date.	`viewStateValue=encodeViewState(viewStateValue);`

Line 5 establishes the variable, currentMonth, which gets the current month.	`var currentMonth = today.getMonth() + 1;`
	`currentMonth = currentMonth.toString()`

Script 9: Reading a Clip Property into a Test

The following brief script reads a clip property into a test using a variable.

```
var auth_id = $prop.value("MessageClip", "auth_id");
```

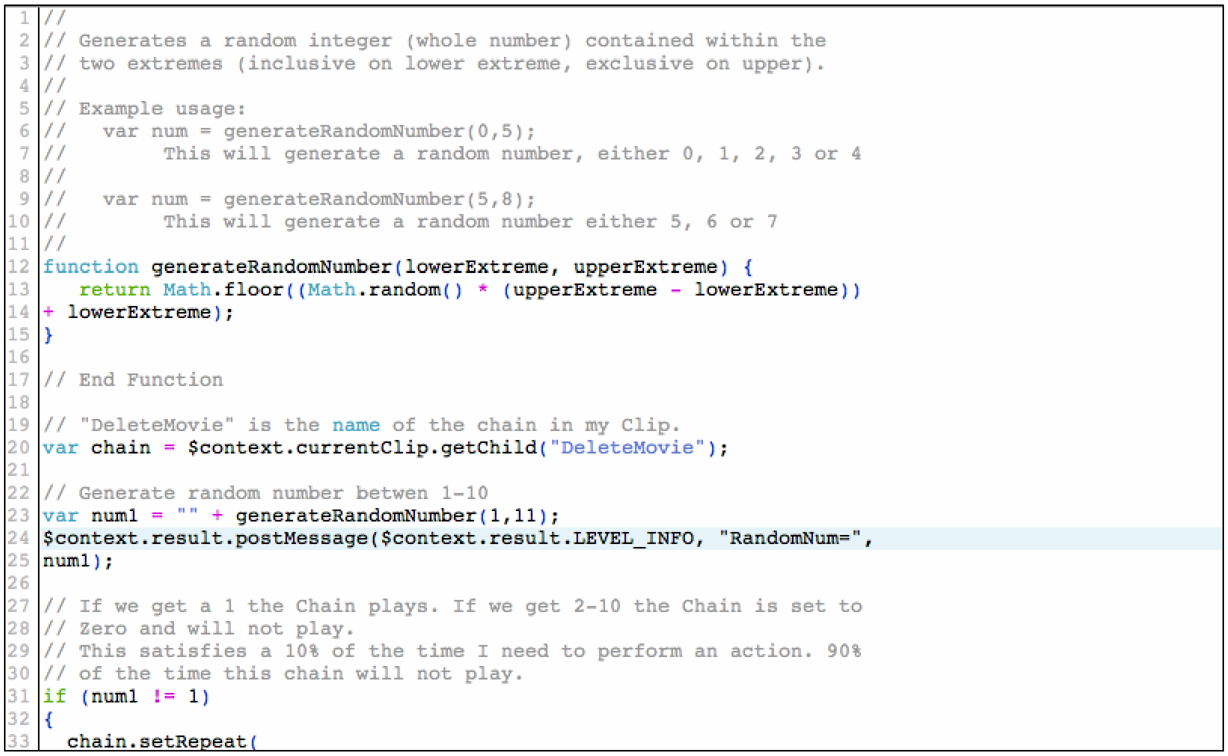
Script 10: Trim Spaces in a String

The following brief script is used to trim spaces from a string.

```
function trim(stringToTrim) {
                return stringToTrim.replace(/^\s+|\s+$/g,"");
}
function ltrim(stringToTrim) {
                return stringToTrim.replace(/^\s+/,"");
}
function rtrim(stringToTrim) {
                return stringToTrim.replace(/\s+$/,"");
}
```

Script 11: Conditional Logic using Chains and Random Numbers

The Conditional Logic using Chains and Random Numbers script plays a chain 10% of the time and turns it off the remaining 90% of the time.



```
//
// Generates a random integer (whole number) contained within the
// two extremes (inclusive on lower extreme, exclusive on upper).
//
// Example usage:
//   var num = generateRandomNumber(0,5);
//        This will generate a random number, either 0, 1, 2, 3 or 4
//
//   var num = generateRandomNumber(5,8);
//        This will generate a random number either 5, 6 or 7
//
function generateRandomNumber(lowerExtreme, upperExtreme) {
   return Math.floor((Math.random() * (upperExtreme - lowerExtreme))
+ lowerExtreme);
}

// End Function

// "DeleteMovie" is the name of the chain in my Clip.
var chain = $context.currentClip.getChild("DeleteMovie");

// Generate random number betwen 1-10
var num1 = "" + generateRandomNumber(1,11);
$context.result.postMessage($context.result.LEVEL_INFO, "RandomNum=",
num1);

// If we get a 1 the Chain plays. If we get 2-10 the Chain is set to
// Zero and will not play.
// This satisfies a 10% of the time I need to perform an action. 90%
// of the time this chain will not play.
if (num1 != 1)
{
  chain.setRepeat(
```

Lines 12-14 generate a random integer contained within the two extremes.	`function generateRandomNumber(lowerExtreme, upperExtreme) {`
	`   return Math.floor((Math.random() * (upperExtreme - lowerExtreme))`
	`+ lowerExtreme);`
	`}`

Line 20 creates a variable, chain, that gets the child of the name specified.	`var chain = $context.currentClip.getChild("DeleteMovie");`

Lines 23-25 generate a random number between 1-10.	`var num1 = "" + generateRandomNumber(1,11);`
	`$context.result.postMessage($context.result.LEVEL_INFO, "RandomNum=",`
	`num1);`

Use the var, num1, from above to play the clip, otherwise set it to 0 using setRepeat.	if (num1 != 1)
	{
	chain.setRepeat(
	chain.REPEAT_TIMING_SERIAL,
	chain.REPEAT_TYPE_COUNT_CONSTANT,
	0, //THIS means do it ZERO times
	chain.REPEAT_DISTRIBUTION_CONSTANT,
	0);
	}

Script 12: Replace Spaces with Plus (+) Signs

This simple script replaces any spaces in the `school_id` variable with `+` signs.

```
var school_id = school_id.replace(/\s+/g,'+');
```

Such a script is used in cases where parameterized data (i.e. from a CSV file) has spaces in the field, but POST data requires that these spaces be converted to plus signs (+).

Script 13: Dynamically Set Chain Repeats

This script sets the number of chain repeatsto 10 dynamically during test execution.

```
// This Script sets the repeat specifications
// for the chain that follows it in the Clip.

var chain = $context.currentItem.nextItem;
chain.setRepeat(chain.REPEAT_TIMING_SERIAL,
chain.REPEAT_TYPE_COUNT_CONSTANT, 10, chain.REPEAT_DISTRIBUTION_CONSTANT,
0);
$context.result.postMessage($context.result.LEVEL_INFO, "Set repeat spec for
item: " + chain.name);
```

Script 14: Retrieve the Current Cookie Values

This Script retrieves the current cookie values and displays them in the Result:

```
var cookies = $context.currentClip.targets[0].cookies;
if (cookies == null)
{
  $context.result.postMessage($context.result.LEVEL_INFO, "No cookies.");
}
else
{
  var text = "";
  for each (var cookie in cookies))
  {
    text += "name=" + cookie.name + ", ";
    text += "domain=" + cookie.domain + ", ";
    text += "path=" + cookie.path + ", ";
    text += "value=" + cookie.value + ", ";
    text += "expirationDate=" + cookie.expirationDate + ", ";
    text += "secure=" + cookie.secure + "\n";
  }

  $context.result.postMessage($context.result.LEVEL_INFO, cookies.length + "
cookies.", text);
}
```

Script 15: Replace the Cookie List with a New List

This script that replaces the entire current cookie list with a new list that contains two cookies named “MyCookie1” and “MyCookie2”:

```
var newList = new Array();

var cookie = new Object();
cookie.name = "MyCookie1";
cookie.domain = "myhostname";
cookie.path = "/some/path";
cookie.value = "Value of MyCookie1";
cookie.expirationDate = new Date("05 Aug 2030 00:00:00 GMT");
cookie.secure = false;
newList[0] = cookie;

cookie = new Object();
cookie.name = "MyCookie2";
cookie.domain = "myhostname";
cookie.path = "/some/path";
cookie.value = "Value of MyCookie2";
cookie.expirationDate = new Date("05 Aug 2030 00:00:00 GMT");
cookie.secure = false;
newList[1] = cookie;
$context.currentClip.targets[0].cookies = newList;
```

Script 16: Find the Current Cookie and Change Its Value

This script that finds the current cookie named “ChocolateChip” and changes it’s value to “42”:

```
var list = $context.currentClip.targets[0].cookies;

if (list != null)
{
  for each (var cookie in list)
  {
    if (cookie.name == "ChocolateChip")
    {
      cookie.value = "42";
      $context.result.postMessage($context.result.LEVEL_INFO, "Cookie value 
replaced.");
      break;
    }
  }
}

$context.currentClip.targets[0].cookies = list;
```

Script 17: Add a New Cookie to the Cookie List

This script that adds new cookie named “ChocolateChip” to the current cookie list:

```
var list = $context.currentClip.targets[0].cookies;

if (list == null)
  list = new Array();

var cookie = new Object();
cookie.name = "ChocolateChip";
cookie.domain = "myhostname";
cookie.path = "/my/path";
cookie.value = "Cookie value";
cookie.expirationDate = new Date("05 Aug 2030 00:00:00 GMT");
cookie.secure = false;

list[list.length] = cookie;

$context.currentClip.targets[0].cookies = list;
```

Script 18: Set only the next delay to a random value

This script is a variation of the Random Think Times script. The Random Think Times script resets ALL delays. This script resets JUST the next one.

```
var nextItem = $context.currentItem.nextItem;

var minDelayTime = 24000; //default = 24000
var maxDelayTime = 36000;  //default = 32000

randomDelayTime = new String(minDelayTime + Math.max(Math.round(Math.random() * 
(maxDelayTime - minDelayTime)), 0));
nextItem.systemPropertyList.setPropertyValue("Duration", 
randomDelayTime.split(".")[0]);
```

Script 19: Check for 200 status code; stop clip if not found

This script detects an HTTP 200 status code and stops the test clip if that code is not found.

```
var msg = $context.currentItem.previousItem;

var StatusCode = msg.getResponse(msg.RESPONSE_HTTP_STATUSCODE);
if (StatusCode != 200){
```

```
$context.result.postMessage($context.result.LEVEL_INFO, "Not an HTTP 200
response; clip end");
$context.currentClip.end();
}
```

SOASTA Extension Reference

This reference is provided as a convenience for script authors using this guide.

- Refer to the Script Reference API (Latest Build) for a complete up-to-date reference to the latest script options in CloudTest.

- Refer to the Cloudlink Documentation page, Release Notes section for a given build—the reference for the specific build is in the row for any given release.

Context Object ($context)

Contains the object model that represents the Composition.

Context Properties

All of the following Context properties are read only.

`composition`

The "Composition" object that represents the top of the Composition object model.

`result`

The "Result" object that represents the Result being produced for the currently playing Composition.

`currentItem`

The object that represents the current context in which the current Script is executing. For Script objects in Clips, Chains, Groups or Pages, this will always be the Script object itself. When scripting is used in an "In Situ Substitution Expression" (ISSE), this will be the object containing the ISSE (such as the Message).

`currentBand`

A "Band" object that represents the current Band according to the context in which the current script is executing. Null if there is no current Band.

`currentBandIndex`

An integer value that represents the "repeat index" of the current Band if the current Band repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Band or the current Band does not repeat or has a repeat count of one.

`currentBandPlayNumber`

An integer value that represents the "play ordinal number" of the current Band. Starting with the number 0, each play of the Band is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Band.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentTrack`

A "Track" object that represents the current Track according to the context in which the current script is executing. Null if there is no current Track.

`currentTrackIndex`

An integer value that represents the "repeat index" of the current Track if the current Track repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Track or if the current Track does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`currentTrackPlayNumber`

An integer value that represents the "play ordinal number" of the current Track. Starting with the number 0, each play of the Track is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Track.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentClip`

A “Clip” object that represents the current Clip according to the context in which the current script is executing. Null if there is no current Clip.

`currentClipIndex`

An integer value that represents the “repeat index” of the current Clip if the current Clip repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Clip or if the current Clip does not repeat or has a repeat count of one (in other words, it is -1 if it doesn’t actually repeat).

`currentClipPlayNumber`

An integer value that represents the “play ordinal number” of the current Clip. Starting with the number 0, each play of the Clip is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Clip.

Play number sequences are maintained within the item’s parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don’t repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentChain`

A “Chain” object that represents the current Chain according to the context in which the current script is executing. Null if there is no current Chain.

`currentChainIndex`

An integer value that represents the “repeat index” of the current Chain if the current Chain repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Chain or if the current Chain does not repeat or has a repeat count of one (in other words, it is -1 if it doesn’t actually repeat).

`currentChainPlayNumber`

An integer value that represents the “play ordinal number” of the current Chain. Starting with the number 0, each play of the Chain is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Chain.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentGroup`

A "Group" object that represents the current Group according to the context in which the current script is executing. Null if there is no current Group.

`currentGroupIndex`

An integer value that represents the "repeat index" of the current Group if the current Group repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Group or if the current Group does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`currentGroupPlayNumber`

An integer value that represents the "play ordinal number" of the current Group. Starting with the number 0, each play of the Group is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Group.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentPage`

A "Page" object that represents the current Page according to the context in which the current script is executing. Null if there is no current Page.

`currentPageIndex`

An integer value that represents the "repeat index" of the current Page if the current Page repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Page or if the current Page does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`currentPagePlayNumber`

An integer value that represents the "play ordinal number" of the current Page. Starting with the number 0, each play of the Page is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Page.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentIf`

An "If" object that represents the current If according to the context in which the current script is executing. Null if there is no current If. If nested within multiple Ifs, this is the lowest-level If (the If "nearest to" the Script in terms of the parentage hierarchy).

`currentIfIndex`

An integer value that represents the "repeat index" of the current If if the current If repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current If or if the current If does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`currentIfPlayNumber`

An integer value that represents the "play ordinal number" of the current If. Starting with the number 0, each play of the If is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current If.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentSwitch`

A “Switch” object that represents the current Switch according to the context in which the current script is executing. Null if there is no current Switch. If nested within multiple Switches, this is the lowest-level Switch (the Switch “nearest to” the Script in terms of the parentage hierarchy).

`currentSwitchIndex`

An integer value that represents the “repeat index” of the current Switch if the current Switch repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Switch or if the current Switch does not repeat or has a repeat count of one (in other words, it is -1 if it doesn’t actually repeat).

`currentSwitchPlayNumber`

An integer value that represents the “play ordinal number” of the current Switch. Starting with the number 0, each play of the Switch is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Switch.

Play number sequences are maintained within the item’s parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don’t repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentTransaction`

A “Transaction” object that represents the current Transaction according to the context in which the current script is executing. Null if there is no current Transaction. If nested within multiple Transactions, this is the lowest-level Transaction (the Transaction “nearest to” the Script in terms of the parentage hierarchy).

`currentTransactionIndex`

An integer value that represents the “repeat index” of the current Transaction if the current Transaction repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Transaction or if the current Transaction does not repeat or has a repeat count of one (in other words, it is -1 if it doesn’t actually repeat).

`currentTransactionPlayNumber`

An integer value that represents the "play ordinal number" of the current Transaction. Starting with the number 0, each play of the Transaction is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Transaction.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentMessage`

A "Message" object that represents the current Message according to the context in which the current script is executing. Null if there is no current Message.

`currentMessageIndex`

An integer value that represents the "repeat index" of the current Message if the current Message repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Message, or if the current Message does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`currentMessagePlayNumber`

An integer value that represents the "play ordinal number" of the current Message. Starting with the number 0, each play of the Message is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Message.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentBrowserAction`

A “BrowserAction” object that represents the current Browser Action according to the context in which the current script is executing. Null if there is no current Browser Action.

`currentBrowserActionIndex`

An integer value that represents the “repeat index” of the current Browser Action if the current Browser Action repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Browser Action or if the current Browser Action does not repeat or has a repeat count of one (in other words, it is -1 if it doesn’t actually repeat).

`currentBrowserActionPlayNumber`

An integer value that represents the “play ordinal number” of the current Browser Action. Starting with the number 0, each play of the Browser Action is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Browser Action.

Play number sequences are maintained within the item’s parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don’t repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentCheckpoint`

A “Checkpoint” object that represents the current Checkpoint according to the context in which the current script is executing. Null if there is no current Checkpoint.

`currentDelay`

A “Delay” object that represents the current Delay component according to the context in which the current script is executing. Null if there is no current Delay component.

`currentScript`

A “Script” object that represents the current Script component in the Composition according to the context in which the current script is executing. Null if there is no current Script component.

`currentScriptIndex`

An integer value that represents the "repeat index" of the current Script component if the current Script component repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if there is no current Script or if the current Script does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`currentScriptPlayNumber`

An integer value that represents the "play ordinal number" of the current Script. Starting with the number 0, each play of the Script is assigned a unique number. The numbers are contiguous (no gaps). The value is -1 if there is no current Script.

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`currentTarget`

A "Target" object that represents the current Target according to the context in which the current script is executing. Null if there is no current Target.

`validationValue`

If the current Script was called to perform validation for Message or Browser Action response, this property contains the value to be validated. If the Script was called to perform overall validation, this will be the entire response, otherwise it will be the specific portion of the response that is to be validated.

Note that the Script also has access to the response through the methods of the Message or Browser Action object.

If the current Script was not called to perform validation, this value will be null.

`userName`

The user ID of the user that started the current Composition playing.

`locationName`

The name of the location (from the Server List) on which the Composition is playing. If the Composition is playing on multiple Maestro servers, this will be the name of the location of the particular server that this Script is playing on.

`serverName`

The name of the server (from the Server List) on which the Composition is playing. If the Composition is playing on multiple Maestro servers, this will be the name of the particular server that this Script is playing on.

`serverType`

The type of the server (from the Server List) on which the Composition is playing ("General" or "Load"). If the Composition is playing on multiple servers, this will be the type of the server on which the current Script is playing.

Note that this is not necessarily the same as the "Load mode" setting in the Composition Editor. It is possible to play a Composition that is in "Load mode" on a General server, and it is also possible to play a Composition that is not in "Load mode" on a Load server. Those two things are not the same.

Context Methods

```
void abortScript(string errorText)
```

Immediately aborts play of the current Script, and considers the Script to have failed with the given error text. If no error text is provided, the text "Script aborted." is used.

```
Array readFromURL(string url, string format,
                  boolean useCache)
```

Note: This function has been replaced by the "readFromURL" function on the **SystemUtilities** object, and is maintained here only for backwards compatibility.

CustomProperties ($prop)

Provides an easy shortcut for accessing the Custom Properties in the Composition without having to traverse the object model.

CustomProperty Methods

```
string value(string pathType, string propertyPath)
```

Returns the value of the specified Custom Property.

The "pathType" parameter value specifies the "starting point" of the path, relative to the current context in which the current Script is running. It can be any of the following values (case is not significant):

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to the current Band in which the Script is executing.

- "`Track`"

  The path is relative to the current Track in which the Script is executing.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to the current Clip in which the Script is executing. If the Script is nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip "nearest to" the Script in terms of the parentage hierarchy).

- "`Chain`"

  The path is relative to the current Chain in which the Script is executing. If the Script is nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain "nearest to" the Script in terms of the parentage hierarchy).

- "`Group`"

  The path is relative to the current Group in which the Script is executing. If the Script is nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the Script in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the current Transaction in which the Script is executing. If the Script is nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the Script in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the current If in which the Script is executing. If the Script is nested within multiple Ifs, the path is relative to the lowest-level containing If (the If “nearest to” the Script in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the current Switch in which the Script is executing. If the Script is nested within multiple Switch, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the Script in terms of the parentage hierarchy).

- “`Page`”

  The path is relative to the current Page in which the Script is executing.

- “`Message`”

  The path is relative to the current Message in which the Script is executing.

- “`Browser Action`”

  The path is relative to the current Browser Action in which the Script is executing.

- “`Destination`”

  The path is relative to the current Message or Browser Action in which the Script is executing. Once the item is found, the result is to be the item’s Target.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void set(string pathType, string propertyPath, var
         newValue)
```

Sets a new value into the specified Custom Property.

The “pathType” and “propertyPath” parameters are the same as for the “value” method, above.

The “newValue” parameter is the new value for the property. It can be null. (The “undefined” value is treated as null.)

SystemProperties ($sysprop)

Provides an easy shortcut for accessing the System Properties in the Composition without having to traverse the object model.

System Property Methods

```
string value(string pathType, string propertyPath)
```

Returns the value of the specified System Property.

The “pathType” parameter value specifies the “starting point” of the path, relative to the current context in which the current Script is running. It can be any of the following values (case is not significant):

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to the current Band in which the Script is executing.

- “`Track`”

  The path is relative to the current Track in which the Script is executing.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the current Clip in which the Script is executing. If the Script is nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the Script in terms of the parentage hierarchy).

- “`Chain`”

The path is relative to the current Chain in which the Script is executing. If the Script is nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the Script in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the current Group in which the Script is executing. If the Script is nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the Script in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the current Transaction in which the Script is executing. If the Script is nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the Script in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the current If in which the Script is executing. If the Script is nested within multiple Ifs, the path is relative to the lowest-level containing If (the If “nearest to” the Script in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the current Switch in which the Script is executing. If the Script is nested within multiple Switch, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the Script in terms of the parentage hierarchy).

- “`Page`”

  The path is relative to the current Page in which the Script is executing.

- “`Message`”

  The path is relative to the current Message in which the Script is executing.

- “`Browser Action`”

  The path is relative to the current Browser Action in which the Script is executing.

- `“Destination”`

  The path is relative to the current Message or Browser Action in which the Script is executing. Once the item is found, the result is to be the item’s Target.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void set(string pathType, string propertyPath, var
        newValue)
```

Sets a new value into the specified System Property.

The “pathType” and “propertyPath” parameters are the same as for the “value” method, above.

The “newValue” parameter is the new value for the property. It can be null. (The “undefined” value is treated as null.) The value will be converted to a string.

System Properties

Composition:

`StartTimeMillis` - Composition start time, expressed as the difference, in milliseconds, between the start time and midnight, January 1, 1970 UTC

`ServerNumber` - The number of the current server for a multi-server Composition. Ranges from zero to n-1, where n is the number of servers. Always zero when the Composition runs all on one server.

Track:

`VUNumber` - The virtual user number represented by the Track

Message:

`WSA-MessageID` - For WSA (Web Services Addressing) messages, an IRI that uniquely identifies this message in time and space. If this value is not explicitly set, the system will create a unique ID for the message.

`WSA-To` - For WSA (Web Services Addressing) messages, an IRI for the destination. If this value is not explicitly set, the system will create a value for the message.

Browser Action:

`Param1` - The value of the first Browser Action parameter.

`Param2` - The value of the second Browser Action parameter.

`Param3` - The value of the third Browser Action parameter.

`Param4` - The value of the fourth Browser Action parameter.

Delay:

`Type` - Either “Constant” to indicate that this Delay has a fixed duration, or “Random” to indicate that this Delay’s duration is chosen at random on each play.

`Duration` - If type is “Constant”, the duration for this Delay, in milliseconds. Not used if type is “Random”.

`Minimum` - If type is “Random”, the minimum of the range from which durations are to be randomly chosen, in milliseconds. Not used if type is “Constant”.

`Duration` - If type is “Random”, the maximum of the range from which durations are to be randomly chosen, in milliseconds. Not used if type is “Constant”.

Target (HTTP or SOAP):

`URL` - Full URL. Setting this property also changes the HostName, ServicePath, Port, and UseSSL properties.

`HostName` - Host name

`ServicePath` - Service path

`Port` - HTTP Port

`UseSSL` - True if SSL (https) is to be used.

`UserName` - HTTP Basic Authentication User name

`Password` - HTTP Basic Authentication User password

`MaximumConnectionsPerHost` - Maximum number of connections that will be created for any particular host URI

`MaximumTotalConnections` - Maximum number of active connections for this Target

`ConnectionTimeout` - Connection timeout, in milliseconds, zero means no timeout

`SocketReadTimeout` - Socket read timeout, in milliseconds, zero means no timeout

`HttpHostOverride` - String value to override a target's host value. See How do I override host files for a load test?

`HttpBinaryConversion` – Integer to specify a binary conversion, such as in the conversion of WCF info.

`connectionRefresh` – Maximum lifetime of a connection for reuse, in seconds. If -1, no lifetime refresh is enforced.

`connectionIdleRefresh` – Maximum idle time for connection for reuse, in seconds. If -1, no idle refresh is enforced.

`UseOAuth` – True if target uses OAuth authentication.

`oauth_consumer_key` – Customer key for OAuth authentication.

`oauth_consumer_secret` – For OAuth authentication, the current Consumer Secret used to authenticate the Consumer to the Service Provider.

`oauth_token_secret` – Current token value for OAuth authentication.

`oauth_signature_method` – Signature method for OAuth authentication.

`oauth_signature` – Current OAuth signature.

`oauth_timestamp` – Current OAuth timestamp.

oauth_nonce - Current OAuth nonce.

oauth_callback - HTTP address for OAuth authentication callback.

oauth_callback_confirmed - Current value for OAuth authentication callback confirmation.

oauth_verified - Current value for OAuth authentication callback verifier.

oauth_token - Current value for OAuth authentication callback token.

Target (JMS):

`URL` - JNDI URL.

`ProviderName` - JMS Provider name

`DestinationName` - Topic or Queue name

`ConnectionFactoryName` - Connection Factory name

`JNDIUserName` - JNDI user name

`JNDIPassword` - JNDI password

`ConnectionFactoryUserName` - Connection Factory user name

`ConnectionFactoryPassword` - Connection Factory password

Target (Browser):

`StartingURL` - Starting URL

`BrowserType` - Browser type

`Conductor` - Conductor Name

`WaitTimeout` - Wait timeout for a single wait condition in milliseconds

`AllowNativeXPath` - Whether to use the browser's native XPath for evaluating locators

`MouseSpeed` - The number of pixels to move the mouse in drag and drop or move events

`WaitInterval` - The interval on which to check for the wait condition

`ActionTimeout` - Action timeout in milliseconds

`FirefoxProfle` – The Firefox browser profile to set

System Utilities ($util)

A singleton object that provides various utilities for use by the Script.

Accessible through the $util constant.

System Utilities Methods

`string decodeString(stringToDecode, decodingTypeIndicator)`

Returns the decoded string. Null if the input string was null or undefined.

The “stringToDecode” specifies the string to be decoded.

The “decodingTypeIndicator” specfies the type of decoding to be done.

`string encodeString(stringToEncode, encodingTypeIndicator)`

Returns the encoded string. Null if the input string was null or undefined.

The “stringToEncode” specifies the string to be decoded.

The “EncodingTypeIndicator” specfies the type of decoding to be done.

`string generateRandomString(length, characterList)`

Returns the generated string.

The “length” of the string to generate.

The “characterList” is a String containing the pool of characters from which to generate the String. Normally each character would appear exactly once in the list. However, if a character appears multiple times that will give it greater weighting in the random selection and thus that character will tend to appear more often.

`string generateRandomStringAlpha(length)`

Returns the generated string of alphabetic characters of the specified length. The returned string will contain characters in the ranges A-Z and a-z.

The “length” of the string to generate.

`string generateRandomStringAlphanumeric(length)`

Returns the generated string of alphanumeric characters of the specified length. The returned string will contain characters in the ranges A-Z, a-z, 0-9.

The “length” of the string to generate.

`string generateRandomStringDecimalDigits(length)`

Returns the generated string of decimal digits of the specified length. The returned string will contain characters in the range 0-9.

The “length” of the string to generate.

`string generateRandomStringHexDigits(length)`

Returns the generated string of of hexadecimal digits of the specified length. The returned string will contain characters in the ranges A-F and 0-9.

The “length” of the string to generate.

```
Array readFromURL(string url, string format,
                  boolean useCache, connectionTimeout,
                  readTimeout)
```

Reads data from a CSV file (resource) at the specified URL, and returns that data as an array of strings.

Each element in the returned array represents a row in the file. For any row that contains multiple values, the value for that row is a nested array of the values in that row.

The "url" parameter is the URL to read from.

The "format" parameter is optional, but if specified must be the string "CSV", which is the only value currently supported.

The "useCache" parameter is optional. If the value is true, the data that is read is cached in memory, so that subsequent reads from the same URL in the same play of the Composition will retrieve the data from memory rather than re-reading from the URL. If the value is false, the data will be read from the URL on every call. If this parameter is omitted or null, true is assumed.

Example 1

If you use the following form:

var dataList =
$util.readFromURL("http://www.myhost.com/directory/file.csv", "CSV", true);

It will be read once on each server.

If you pass in false for the third parameter (or leave off the third parameter), it will be read every time.

Example 2

To read from a file at “http://host/files/Locations.csv”, the following call would be made:

```
var locationList =
$util.readFromURL(“http://host/files/Locations.csv”);
```

If the file contained the following lines:

```
San Francisco,CA,94103

Timbuktu

Caribou,Aroostook County,Maine,USA

Hill Valley,CA,91905
```

The following array of strings would be returned:

`locationList[0][0]`	`San Francisco`
`locationList[0][1]`	`CA`
`locationList[0][2]`	`94103`
`locationList[1]`	`Timbuktu`
`locationList[2][0]`	`Caribou`
`locationList[2][1]`	`Aroostook County`
`locationList[2][2]`	`Maine`
`locationList[2][3]`	`USA`
`locationList[3][0]`	`Hill Valley`
`locationList[3][1]`	`CA`
`locationList[3][2]`	`91905`

GlobalProperties ($globalprop)

Provides access to all Global Custom Properties.

Global Property Methods

```
string value(string listName, string propertyName, boolean
             suppressIncrement)
```

Returns the value of the specified Global Custom Property.

The “listName” parameter specifies the name of the Global Custom Property List. If the value is null, the name “Default” is used.

The “propertyName” parameter specifies the name of the individual Global Custom Property within the list.

If the property is a counter, it may be incremented before the value is returned, unless the “suppressIncrement” parameter is true. (“Message-level” counters are incremented only when accessed from within a Message, and then only on the first access within that Message.)

If the “suppressIncrement parameter is omitted, “false” is assumed.

```
string valueUsingPath(string propertyPath,

                      boolean suppressIncrement)
```

Returns the value of the specified Global Custom Property.

The “propertyPath” parameter specifies which property is to be accessed. It can be as simple as only a property name, in which case the property will be taken from the default Global Custom Property List (named “Default”), or it can be a “path” that specifies a Global Custom Property List and a property.

A “path” to a property is given by using the name of the Global Custom Property List, a slash, and then the name of the Global Custom Property in that list.

For example, the following path:

```
My list/Some property
```

Refers to the property named “`Some property`” in the Global Custom Property List named “`My list`”.

The following paths are equivalent:

```
Default/Property 6

Property 6
```

Both of the above paths refer to the property named "`Property 6`" in the default Global Custom Property List.

If the property is a counter, it may be incremented before the value is returned, unless the "suppressIncrement" parameter is true. ("Message-level" counters are incremented only when accessed from within a Message, and then only on the first access within that Message.)

If the "suppressIncrement parameter is omitted, "false" is assumed.

```
string set(string listName, string propertyName, var
           newValue)
```

Sets a new value into the specified Global Custom Property.

The "listName" parameter specifies the name of the Global Custom Property List. If the value is null, the name "Default" is used.

The "propertyName" parameter specifies the name of the individual Global Custom Property within the list.

The "newValue" parameter is the new value for the property. It can be null. (The "undefined" value is treated as null.) The value will be converted to a string.

```
string setUsingPath(string propertyPath, var newValue)
```

Sets a new value into the specified Global Custom Property.

The "propertyPath" parameter specifies which property is to be accessed. It can be as simple as only a property name, in which case the property will be taken from the default Global Custom Property List (named "Default"), or it can be a "path" that specifies a Global Custom Property List and a property.

See the description of property paths for the "valueUsingPath" method.

The "newValue" parameter is the new value for the property. It can be null. (The "undefined" value is treated as null.) The value will be converted to a string.

Result Object

Represents the Result being produced for the currently playing Composition.

Result Properties

`LEVEL_ERROR` (read only - integer)

An indicator to be passed in for the “level” parameter of the “postMessage” method.

`LEVEL_STATISTICS` (read only - integer)

An indicator to be passed in for the “level” parameter of the “postMessage” method.

`LEVEL_INFO` (read only - integer)

An indicator to be passed in for the “level” parameter of the “postMessage” method.

`LEVEL_VERBOSE` (read only - integer)

An indicator to be passed in for the “level” parameter of the “postMessage” method.

`summary` (read/write - string)

The “summary” text that will be shown in the Result for the Composition.

`name` (read only - string)

The name of the Result in the Repository (the base name only, does not include the path). Null if no Result is being written.

Result Methods

```
void postMessage(int level, string message, string
                 details)
```

Adds a custom message to the Result object with the given message and detail text. The “details” parameter is optional and may be omitted. isAdopted

Composition Object

Represents the Composition itself.

Composition Properties

`Name` (read only – string)

The name of the Composition

`Parent` (read only – object)

Always null.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Composition”.

`children` (read only – array of objects)

An array of Band objects representing all of the Bands in the Composition.

`index` (read only – integer)

Always returns -1 for Compositions.

`iterationNumber` (read only – integer)

If the Composition is repeating, this is the iteration number of the current repeat of the Composition. The iteration number starts at one. Compositions can be set to repeat via the “Repeat play” option in the “Play with options” dialog in Central, or in the “Play Options” dialog in the Composition Editor.

`nextItem` (read only – object)

Always returns null for Compositions.

`previousItem` (read only – object)

Always returns null for Compositions.

`forEachValue` (read only – string, number, date/time, or null)

Always returns null for Compositions.

`repeatIndex` (read only – integer)

Always -1 for Compositions.

`playNumber` (read only – integer)

Always 0 for Compositions.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

Always 0 for Compositions.`isPreviewMode` (read only – boolean)

True if the Composition is being played in Preview mode.

Composition Methods

`void abort(Object message, Object details)`

Terminates the Composition as if there were an error. The message and detail text given in the parameters are inserted into the Result object.

Example:

```
$context.composition.abort("Message text from script",
                              "Details\nfrom\nscript");
```

`void stop()`

Stops the Composition, as if the user had pressed the “stop” button.

`void pause()`

Pauses the Composition, as if the user had pressed the “pause” button. Note that this merely starts the Composition pausing. The Composition will not actually be completely paused until all portions of the Composition have paused, which cannot happen until the current Script ends.

`object getChild(string childName)`

Returns a specific Band within the Composition by name, or null if there is no Band with the specified name.

`object getItemViaPath(string pathType, string path)`

Given the path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter value specifies the “starting point” of the path. For Compositions the only allowable value is “Composition”, to indicate that the path is relative to the Composition.

The “path” parameter contains a path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

`void rampPause()`

Pauses the Composition’s “ramp up”, if any, as if the user had pressed the “pause ramp” button. Note that this merely starts the ramp pausing. The ramp will not actually be completely paused until all portions of the Composition on all servers have paused the ramp.

This method may be called at any time, including when the ramp up is already paused or pausing, or when it is resuming.

“Ramp up” is defined as Track parallel repeating with an “interval”.

`void rampResume()`

Resumes the Composition’s “ramp up”, if any, as if the user had pressed the “resume ramp” button. Note that this merely starts the ramp resuming. The ramp will not actually be completely resumed until all portions of the Composition on all servers have resumed the ramp.

This method may be called at any time, including when the ramp up is already resuming or is not paused.

“Ramp up” is defined as Track parallel repeating with an “interval”.

Band Object

Represents a Band within the Composition.

Band Properties

`name` (string)

The name of this item.

An item’s name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (string)

The string “Band”.

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the “play ordinal number” of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item’s parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don’t repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the “Renew parallel repeats” option enabled, this is an integer value that is the “playNumber” value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the “Renew parallel repeats” option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If “Renew parallel repeats” is not enabled for this item, or it doesn’t repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Band Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given the path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter value specifies the "starting point" of the path, relative to this Band. For Bands, it can be any of the following values (case is not significant):

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to this Band.

The "path" parameter contains a path as specified for "In Situ Substitution Specifications" (see the separate document on ISSEs).

`void clearRepeat()`

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

```
void endRepeat()
```

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an “abort” – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the “renewal” option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with “parallel repeat renewal”, this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal “lines” that are proceeding in parallel (there will always be

3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Track

Represents a Track within the Composition.

Track Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Track".

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

Track Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter value specifies the “starting point” of the path, relative to this Track. For Tracks it can be any of the following values (case is not significant):

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to this Track’s parent Band.

- “`Track`”

  The path is relative this Track.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

```
void end(String optionalErrorText)
```

Requests that play of this Track be terminated. If an optional error text string is provided, the Track will be considered to have ended in error. If the Track is not currently playing, no action is taken.

Note that this is not an “abort” – play will be ended after any currently playing Clip(s) complete.

```
void endRepeat()
```

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an “abort” – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Target Object

Represents the target itself.

Target Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Target".

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

This property is always null for Targets.

`repeatIndex` (read only – integer)

Always -1 for Targets.

`playNumber` (read only – integer)

Always 0 for Targets.

`playNumberBeforeRenewal` (read only – integer)

Always 0 for Targets.

`playNumberWithinRenewal` (read only – integer)

Always 0 for Targets.

`cookies` (readable and settable – array of "structs" (objects with properties))

This contains the current list of cookies being maintained. This list changes as responses are received that contain cookies.

The value is null if there are currently no cookies being maintained.

There is a single list of cookies maintained across all Targets within the same instance of the same Clip. Thus the value for this property will be the same for all Targets in the same instance of the same Clip, and setting this property affects the Clip's cookie processing across all Targets within the Clip.

This property can be set to replace the entire list of cookies. This can be done by modifying the existing list, or by creating an entirely new list.

The value of the property is an array of Objects. Each object in the array has the following property values:

- "`name`" (String) – the name of the cookie.
- "`domain`" (String) – the domain name of the cookie.
- "`path`" (String) – the cookie's path.
- "`value`" (String) – the value of the cookie.
- "`expirationDate`" (Date object) – the expiration date of the cookie (null if none).
- "`secure`" (Boolean) – true if it is a secure cookie.

Here is an example Script that retrieves the current cookie values and displays them in the Result:

```
var cookies = $context.currentClip.targets[0].cookies;

if (cookies == null)

{

  $context.result.postMessage($context.result.LEVEL_INFO, "No
cookies.");

}

else

{

  var text = "";

  for each (var cookie in cookies))

  {

    text += "name=" + cookie.name + ", ";

    text += "domain=" + cookie.domain + ", ";

    text += "path=" + cookie.path + ", ";

    text += "value=" + cookie.value + ", ";

    text += "expirationDate=" + cookie.expirationDate + ", ";

    text += "secure=" + cookie.secure + "\n";

  }


  $context.result.postMessage($context.result.LEVEL_INFO,
cookies.length + " cookies.", text);

}
```

Here is an example Script that replaces the entire current cookie list with a new list that contains two cookies named “MyCookie1” and “MyCookie2”:

```
var newList = new Array();


var cookie = new Object();

cookie.name = "MyCookie1";

cookie.domain = "myhostname";

cookie.path = "/some/path";

cookie.value = "Value of MyCookie1";

cookie.expirationDate = new Date("05 Aug 2030 00:00:00 GMT");

cookie.secure = false;

newList[0] = cookie;


cookie = new Object();

cookie.name = "MyCookie2";

cookie.domain = "myhostname";

cookie.path = "/some/path";

cookie.value = "Value of MyCookie2";

cookie.expirationDate = new Date("05 Aug 2030 00:00:00 GMT");

cookie.secure = false;

newList[1] = cookie;

$context.currentClip.targets[0].cookies = newList;
```

Here is an example Script that finds the current cookie named “ChocolateChip” and changes it’s value to “42”:

```
var list = $context.currentClip.targets[0].cookies;


if (list != null)

{

  for each (var cookie in list)

  {

    if (cookie.name == "ChocolateChip")

    {

      cookie.value = "42";

      $context.result.postMessage($context.result.LEVEL_INFO,
"Cookie value replaced.");

      break;

    }

  }

}


$context.currentClip.targets[0].cookies = list;
```

Here is an example Script that adds new cookie named “ChocolateChip” to the current cookie list:

```
var list = $context.currentClip.targets[0].cookies;


if (list == null)

  list = new Array();


var cookie = new Object();

cookie.name = "ChocolateChip";

cookie.domain = "myhostname";

cookie.path = "/my/path";

cookie.value = "Cookie value";

cookie.expirationDate = new Date("05 Aug 2030 00:00:00 GMT");

cookie.secure = false;


list[list.length] = cookie;


$context.currentClip.targets[0].cookies = list;
```

Target Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter indicates the starting point of the path within this item’s container hierarchy. For Targets, it may have the following values:

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to the Band that contains this item.

- “`Track`”

  The path is relative to the Track that contains this item.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- `“Target”`

  The path is relative to this Target.

  The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

`string signURL(string url)`

  “Signs” a URL according to the OAuth (“Open Authorization”) protocol.

  The “url” parameter is a complete HTTP URL. The return value is a modified version of the URL, “signed” according to OAuth.

  The information needed to perform the signing operation is taken from the following System Property values of the Target:

  - “oauth_consumer_key”
  - “oauth_consumer_secret”
  - “oauth_token”
  - “oauth_token_secret”
  - “oauth_signature_method”

`boolean isAdopted()`

Returns true if this Target was originally in a child nested Clip but was adopted by the current Clip due to Target Merging.

Most operations on this Target will also affect the original Target in the child nested Clip. For example, changing a System Property for this Target will also change the same System Property of the original Target in the child Clip. One notable exception is Custom Properties – this Target will still maintain it’s own set of Custom Properties, separate from the original Target in the child Clip.

`boolean isIndirect()`

Returns true if this Target was merged into a parent Clip due to Target Merging.

Most operations on this Target will be deferred to the adoptive Target in the parent Clip. For example, changing a System Property for this Target will also change the same System Property of the adoptive Target in the parent Clip. One notable exception is Custom Properties – this Target will still maintain it’s own set of Custom Properties, separate from the adoptive Target in the parent Clip.

`boolean hasBeenMergedInto()`

Returns true if one or more Targets in one or more child nested Clips have been merged into this Target due to Target merging.

Most operations on this Target will also affect the original Targets in the child nested Clips. For example, changing a System Property for this Target will also change the same System Property of the original Targets in the child Clips. One notable exception is Custom Properties – this Target will still maintain it's own set of Custom Properties, separate from the original Targets in the child Clips.

`object getDirectTarget()`

If this Target was merged into a parent Clip due to Target Merging, this returns the Target object for the Target that it was merged into. If it was merged into several levels of nested Clips above, the highest-level Target is returned.

If this Target was not merged into a parent Clip, this Target object itself is returned.

`void stop()`

> Stops play of the current instance of the Track. Does not affect any other instances of the Track. Does not cause the Track to stop repeating if it repeats.
>
> The stop will occur immediately once the calling Script ends.
>
> Wherever this call is made within the Track hierarchy, all items in the current instance of the Track will be stopped. For example, the Clip containing the current Script will be stopped, no further items in the Clip will play and no new Clips will be started in the current Track instance.

`void stopServer()`

> Stops play of all instances of the Track. If the Track repeats, no further repeats will occur.
>
> The stop will occur for the current instance of the Track once the calling Script ends. The stop will occur immediately during the call for all other instances of the Track.
>
> All instances of the Track will be stopped and no further items will play. For example, containing Clips will be stopped, no further items in the Clip will play and no new Clips will be started.
>
> If the Track has been copied to multiple servers (for example by entering a value into the "Copy Count" field when "Dedicated Load Server" is selected in the "General" tab in the Composition Editor), this call will only affect instances of the Track on the individual server on which the current Composition is playing. It will not affect copies of the Track that are playing on other servers.

```
Void drainServer()
```

Prohibits any further instances of the Track from being created on the current server. If the Track repeats, no new repeats will be started. If the Track uses “Parallel repeat renewal”, no new renewals will occur.

Any existing instances of the Track that have already started will be allowed to finish normally.

The prohibition on new instances of the Track takes effect as soon as the call is made. Once the prohibition is in effect, it cannot be cancelled.

If the Track has been copied to multiple servers (for example by entering a value into the “Copy Count” field when “Dedicated Load Server” is selected in the “General” tab in the Composition Editor), this call will only affect instances of the Track on the individual server on which the current Composition is playing. It will not affect copies of the Track that are playing on other servers.

Clip Object

Represents a Test Clip within the Composition.

Clip Properties

All of the following Clip properties are read-only.

`name` (string)

The name of this item.

An item’s name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent Track.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this Clip.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Clip”.

`children` (read only – array of objects)

An array of objects representing the children of this Clip, if any. Null if the Clip has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`targets` (array of objects)

An array of Target objects representing all of the Targets referenced by this Clip. Null if there are no Targets.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`dynamicResourceCache` (readable and settable – array of Strings)

If Dynamic-resource Caching is enabled for the Clip, this property contains the list of URLs currently in the cache. This list changes as Pages are played and dynamic resources are extracted from responses to the Main Message.

The value is null if the Dynamic-Resource Cache is not enabled, or if the cache is empty.

This property can be set to replace the entire cache. This can be done to modify the existing list of URLs, or to create an entirely new list of URLs.

The value of the property is an array of Strings. Each String is the fully-qualified URL of a Page dynamic resource. The list that is returned is sorted. Any new list that is provided need not be sorted. Null entries in any new list provided are ignored.

Here is an example Script that retrieves the current cache and displays it in the Result:

```
var urls = $context.currentClip.dynamicResourceCache;

if (urls == null)

{

  $context.result.postMessage($context.result.LEVEL_INFO, "Cache
is empty.");

}
```

```
else

{

  var text = "";

  for each (var url in urls))

  {

    text += url + "\n";

  }



  $context.result.postMessage($context.result.LEVEL_INFO,
urls.length + " URLs", text);

}
```

Here is an example Script that replaces the entire current cache with a new list that contains two URLs:

```
var newList = new Array();



newList[0] = "http://somewhere.com/somepage.html";

newList[1] = "http://someplace.com/index.html";



$context.currentClip.dynamicResourceCache = newList;
```

Here is an example Script that removes a specific URL from the cache, if it is there:

```
var list = $context.currentClip.dynamicResourceCache;


if (list != null)

{

  for (var i = 0; i < list.length; i++)

  {

    if (list[i] == "http://somewhere.com/somepage.html")

    {

      list[i] = null;

      $context.currentClip.dynamicResourceCache = list;

      $context.result.postMessage($context.result.LEVEL_INFO,
"URL removed.");

      break;

    }

  }

}
```

Here is an example Script that adds a new URL to the current cache:

```
var list = $context.currentClip.dynamicResourceCache;


if (list == null)

  list = new Array();


list[list.length] = "http://somewhere.com/somepage.html";


$context.currentClip.dynamicResourceCache = list;
```

Clip Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getTarget(string targetName)`

Returns a specific Target object by name, or null if there is no Target with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter value specifies the "starting point" of the path, relative to this Clip. For Clips it can be any of the following values (case is not significant):

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to this Clips parent Band.

- "`Track`"

  The path is relative this Clip's parent Track.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to this Clip.

- "`Chain`"

The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain "nearest to" the item in terms of the parentage hierarchy).

- "`Group`"

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group "nearest to" the item in terms of the parentage hierarchy).

- `“Transaction”`

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- `“If”`

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- `“Switch”`

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item. See the description of this method in the description of the Band object.

```
void end(String optionalErrorText)
```

Requests that play of this Clip be terminated. If an optional error text string is provided, the Track will be considered to have ended in error. If the Clip is not currently playing, no action is taken.

Note that this is not an “abort” – play will be ended after any currently playing item(s) in the Clip complete.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an “abort” – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the “renewal” option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with “parallel repeat renewal”, this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal “lines” that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Chain Object

Represents a Chain within the Composition.

Chain Properties

`name` (string)

The name of this item.

An item’s name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Chain".

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Chain Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter value specifies the "starting point" of the path, relative to this Chain. For Chains it can be any of the following values (case is not significant):

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to this Chain's parent Band.

- "`Track`"

  The path is relative this Chain's parent Track.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to this Chain's parent Clip.

- "`Chain`"

  The path is relative to this Chain.

The "propertyPath" parameter contains a property path as specified for "In Situ Substitution Specifications" (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The "timingType" parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The "repeatType" parameter indicates what sort of value is contained in the "control" parameter. Currently the "repeatType" parameter must always be set to "REPEAT_TYPE_COUNT_CONSTANT".

The "control" parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

```
void end(String optionalErrorText)
```

Requests that play of this Chain be terminated. If an optional error text string is provided, the Chain will be considered to have ended in error. If the Chain is not currently playing, no action is taken.

Note that this is not an “abort” – play will be ended after any currently playing item in the Chain completes.

```
void endRepeat()
```

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an “abort” – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the “renewal” option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with “parallel repeat renewal”, this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal “lines” that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Group Object

Represents a Group within the Composition.

Group Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Group".

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

playNumberBeforeRenewal (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of "playNumberNumberBeforeRenew" will be equal to the value of "playNumber". If this item does not repeat in parallel, the value will be zero.

See also "playNumber", above.

For example, if parallel repeat number 5 of the item ends, but the "Renew parallel repeats" option is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new "repeatIndex" and "playNumber" values (according to how many other repeats have already occurred). However, the "playNumberBeforeRenewal" value will still be 5 in this example.`REPEAT_TIMING_PARALLEL` (read only – integer)

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

REPEAT_TIMING_PARALLEL (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Group Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter value specifies the "starting point" of the path, relative to this Group. For Groups it can be any of the following values (case is not significant):

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to this Group's parent Band.

- "`Track`"

  The path is relative this Group's parent Track.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to this Group's parent Clip. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip "nearest to" the item in terms of the parentage hierarchy).

- "`Group`"

  The path is relative to this Group.

- "`Transaction`"

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction "nearest to" the item in terms of the parentage hierarchy).

- "`If`"

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If "nearest to" the item in terms of the parentage hierarchy).

- "`Switch`"

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch "nearest to" the item in terms of the parentage hierarchy).

The "propertyPath" parameter contains a property path as specified for "In Situ Substitution Specifications" (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The "timingType" parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The "repeatType" parameter indicates what sort of value is contained in the "control" parameter. Currently the "repeatType" parameter must always be set to "REPEAT_TYPE_COUNT_CONSTANT".

The "control" parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The "distributionType" parameter indicates what sort of value is contained in the "distribution" parameter. Currently the "distributionType" parameter must always be set to "REPEAT_DISTRIBUTION_CONSTANT".

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

`void end(String optionalErrorText)`

Requests that play of this Group be terminated. If an optional error text string is provided, the Group will be considered to have ended in error. If the Group is not currently playing, no action is taken.

Note that this is not an “abort” – play will be ended after any currently playing item in the Group completes.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an “abort” – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the “renewal” option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with “parallel repeat renewal”, this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal “lines” that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Transaction Object

Represents a Transaction within the Composition.

Transaction Properties

`name` (string)

The name of this item.

An item’s name can be changed by setting this property, with the following restrictions:

> The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).
>
> The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Transaction”.

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item’s parent’s array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item’s parent’s array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of "playNumberNumberBeforeRenew" will be equal to the value of "playNumber". If this item does not repeat in parallel, the value will be zero.

See also "playNumber", above.

For example, if parallel repeat number 5 of the item ends, but the "Renew parallel repeats" option is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new "repeatIndex" and "playNumber" values (according to how many other repeats have already occurred). However, the "playNumberBeforeRenewal" value will still be 5 in this example.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Transaction Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

```
object getItemViaPath(string pathType, string path)
```

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter indicates the starting point of the path within this item’s container hierarchy. For Transactions, it may have the following values:

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to the Band that contains this item.

- “`Track`”

  The path is relative to the Track that contains this item.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to this Transaction.

- “`If`”

The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If "nearest to" the item in terms of the parentage hierarchy).

- `"Switch"`

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch "nearest to" the item in terms of the parentage hierarchy).

- The "propertyPath" parameter contains a property path as specified for "In Situ Substitution Specifications" (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The "timingType" parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The "repeatType" parameter indicates what sort of value is contained in the "control" parameter. Currently the "repeatType" parameter must always be set to "REPEAT_TYPE_COUNT_CONSTANT".

The "control" parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The "distributionType" parameter indicates what sort of value is contained in the "distribution" parameter. Currently the "distributionType" parameter must always be set to "REPEAT_DISTRIBUTION_CONSTANT".

The "distribution" parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

`void end(String optionalErrorText)`

Requests that play of this Transaction be terminated. If an optional error text string is provided, the Transaction will be considered to have ended in error. If the Transaction is not currently playing, no action is taken.

Note that this is not an "abort" – play will be ended after any currently playing item in the Transaction completes.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an "abort" – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

If Object

Represents an If within the Composition.

If Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "If".

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of "playNumberNumberBeforeRenew" will be equal to the value of "playNumber". If this item does not repeat in parallel, the value will be zero.

See also “playNumber”, above.

For example, if parallel repeat number 5 of the item ends, but the “Renew parallel repeats” option is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new “repeatIndex” and “playNumber” values (according to how many other repeats have already occurred). However, the “playNumberBeforeRenewal” value will still be 5 in this example.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the “Renew parallel repeats” option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If “Renew parallel repeats” is not enabled for this item, or it doesn’t repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

If Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter indicates the starting point of the path within this item's container hierarchy. For Ifs, it may have the following values:

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to the Band that contains this item.

- "`Track`"

  The path is relative to the Track that contains this item.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip "nearest to" the item in terms of the parentage hierarchy).

- "`Chain`"

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain "nearest to" the item in terms of the parentage hierarchy).

- "`Group`"

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group "nearest to" the item in terms of the parentage hierarchy).

- "`Transaction`"

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction "nearest to" the item in terms of the parentage hierarchy).

- "`If`"

  The path is relative to this If.

- `“Switch”`

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

```
void end(String optionalErrorText)
```

Requests that play of this If be terminated. If an optional error text string is provided, the If will be considered to have ended in error. If the If is not currently playing, no action is taken.

Note that this is not an "abort" – play will be ended after any currently playing item in the If completes.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an "abort" – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Switch Object

Represents a Switch within the Composition.

Switch Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Switch”.

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item’s parent’s array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item’s parent’s array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where “`item`” is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item’s parent’s array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where “`item`” is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a “for-each” repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a “for-each” repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the “repeat index” of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn’t actually repeat).

`playNumber` (read only – integer)

An integer value that represents the “play ordinal number” of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item’s parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don’t repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the “Renew parallel repeats” option enabled, this is an integer value that is the “playNumber” value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of “playNumberNumberBeforeRenew” will be equal to the value of “playNumber”. If this item does not repeat in parallel, the value will be zero.

See also “playNumber”, above.

For example, if parallel repeat number 5 of the item ends, but the “Renew parallel repeats” option is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new “repeatIndex” and "playNumber" values (according to how many other repeats have already occurred). However, the "playNumberBeforeRenewal" value will still be 5 in this example.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Switch Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter indicates the starting point of the path within this item's container hierarchy. For Switches, it may have the following values:

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to the Band that contains this item.

- "`Track`"

  The path is relative to the Track that contains this item.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip "nearest to" the item in terms of the parentage hierarchy).

- "`Chain`"

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain "nearest to" the item in terms of the parentage hierarchy).

- "`Group`"

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group "nearest to" the item in terms of the parentage hierarchy).

- "`Transaction`"

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction "nearest to" the item in terms of the parentage hierarchy).

- "`If`"

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If "nearest to" the item in terms of the parentage hierarchy).

- "`Switch`"

The path is relative to this Switch.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

```
void end(String optionalErrorText)
```

Requests that play of this Switch be terminated. If an optional error text string is provided, the Switch will be considered to have ended in error. If the Switch is not currently playing, no action is taken.

Note that this is not an “abort” – play will be ended after any currently playing item in the Switch completes.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an "abort" – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Page Object

Represents a Page within the Composition.

Page Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Page”.

`children` (read only – array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item’s parent’s array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item’s parent’s array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where “`item`” is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item’s parent’s array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where “`item`” is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`dynamicResourceLinks` (array of strings)

If this Page has dynamic resources enabled, this property contains the list of resource links for the dynamic resources that will be retrieved. The value is an array of strings, with each string being the full URL to the resource.

This array can be modified by Script to change the dynamic resources that will be retrieved. URLs in the array can be modified or removed, and new URLs can be added to the array. Null and undefined entries in the array will be ignored (therefore an entry in the array can be "removed" by setting it to null).

The value of this property is available only after the Page's "main Message" ("HTML Document") has played. Any changes made to this property must be made after that point (any changes made to this property before the main Message plays will be overwritten). In other words, this property should generally only be used by Scripts that appear after the main Message in the Page.

If dynamic resource retrieval has been disabled for the Page, this property is ignored and will have no value, and any value set into it will be ignored.

Here's an example of a Script that uses the "dynamicResourceLinks" property to process the list of links that were extracted from the response to the Page's main Message, and do the following: (1) remove any resource links that contain the string ".css" anywhere in them, and (2) change the first occurrence of "https" to "http" in all links, and (3) add an additional hard-coded link to "http://somewhere.com/addedlink" to the list.

```
var links = $context.currentPage.dynamicResourceLinks;

if (links != null)

{

  for (var i = 0; i < links.length; i++)

  {

    if (links[i].indexof(".css") >= 0)

      links[i] = null;

    else

      links[i].replace(/https/, "http");

  }


  links[links.length] = "http://somewhere.com/addedlink";

}

else

{

  links = new Array();

  links[0] = "http://somewhere.com/addedlink";

}


$context.currentPage.dynamicResourceLinks = links;
```

`dynamicResourceHeaders` (string)

If this Page has dynamic resources enabled, this property contains the list of HTTP Headers that will be added for the dynamic resources that will be retrieved. The value is a string formatted as HTTP Headers are sent in an HTTP request (one line per header, with each line containing the header name, a colon and then the header value).

This string can be modified by Script to change the headers that will be sent. By default, the "User-Agent" header is copied from the Main Message's request.

The value of this property is available only after the Page's "main Message" ("HTML Document") has played. Any changes made to this property must be made after that point (any changes made to this property before the main Message plays will be overwritten). In other words, this property should generally only be used by Scripts that appear after the main Message in the Page.

If dynamic resource retrieval has been disabled for the Page, this property is ignored and will have no value, and any value set into it will be ignored.

Here's an example of a Script that adds the header "MyHeader", with value "123" to every dynamic resource Message in the Page.

```
var headers = $context.currentPage.dynamicResourceHeaders;

if (headers == null)

  headers = "MyHeader: 123";

else

  headers += "\nMyHeader: 123";

$context.currentPage.dynamicResourceHeaders = headers;
```

Page Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter indicates the starting point of the path within this item’s container hierarchy. For Pages, it may have the following values:

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to the Band that contains this item.

- “`Track`”

  The path is relative to the Track that contains this item.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- `“Switch”`

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- `“Page”`

  The path is relative to this Page.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

`void end(String optionalErrorText)`

Requests that play of this Page be terminated. If an optional error text string is provided, the Page will be considered to have ended in error. If the Page is not currently playing, no action is taken.

Note that this is not an “abort” – play will be ended after any currently playing item in the Page completes.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an “abort” – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the “renewal” option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with “parallel repeat renewal”, this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal “lines” that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Checkpoint Object

Represents a Checkpoint within the Composition.

Checkpoint Properties

`name` (string)

The name of this item.

An item’s name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Checkpoint”.

`children` (array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item’s parent’s array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item’s parent’s array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where “`item`” is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

Always -1 for a Checkpoint.

`playNumber` (read only – integer)

Always 0 for a Checkpoint.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Checkpoint Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter indicates the starting point of the path within this item's container hierarchy. For Checkpoints, it may have the following values:

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to the Band that contains this item.

- "`Track`"

  The path is relative to the Track that contains this item.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip "nearest to" the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- “`Checkpoint`”

  The path is relative to this Checkpoint.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- REPEAT_TIMING_PARALLEL
- REPEAT_TIMING_SERIAL

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

Message Object

Represents a Message within the Composition.

Message Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent Clip.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this Message.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Message".

`children` (read only – array of objects)

Always null

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`target` (read only – Target object)

The Target object for this Message. Can be null.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`RESPONSE_TEXT` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_HTTP_BODY` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_HTTP_HEADER` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_HTTP_HEADER_LIST` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_HTTP_PROTOCOL` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_HTTP_STATUSCODE` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_HTTP_STATUSTEXT` (read only – integer)

A constant that can be passed as the “indicator” parameter in calls to the “getResponse” method. (See the description of the “getResponse” method.)

`RESPONSE_TEXT_AS_XML` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getResponse" method. (See the description of the "getResponse" method.)

`RESPONSE_TEXT_AS_HTML` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getResponse" method. (See the description of the "getResponse" method.)

`RESPONSE_TEXT_AS_JSON` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getResponse" method. (See the description of the "getResponse" method.)

`RESPONSE_HTTP_BODY_AS_XML` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getResponse" method. (See the description of the "getResponse" method.)

`RESPONSE_HTTP_BODY_AS_HTML` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getResponse" method. (See the description of the "getResponse" method.)

`RESPONSE_HTTP_BODY_AS_JSON` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getResponse" method. (See the description of the "getResponse" method.)

`MESSAGE_TEXT` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getMessage" and "setMessage" methods. (See the descriptions of those method.)

`MESSAGE_HTTP_BODY` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getMessage" and "setMessage" methods. (See the descriptions of those method.)

`MESSAGE_HTTP_HEADERS` (read only – integer)

> A constant that can be passed as the "indicator" parameter in calls to the "getMessage" and "setMessage" methods. (See the descriptions of those method.)

`responseTime` (read only – long)

The total amount of time, in milliseconds, that it took to send the message and receive the response. Will be null if the message has not yet been sent or if in "preview" mode.

`bytesSentCount` (read only – long)

The number of bytes that were sent. Will be null if the message has not yet been sent or if in "preview" mode. Note: This is currently the number of Unicode characters, which may not be the same as the number of actual bytes.

`bytesReceivedCount` (read only – long)

The number of bytes that were received. Will be null if the message has not yet been sent or if in "preview" mode. Note: This is currently the number of Unicode characters, which may not be the same as the number of actual bytes.

`retryCount` (read only – long)

The number of retries that were required due to unavailable local ports when sending the message. Will be null if the message has not yet been sent or if in "preview" mode.

`retryTotalTime` (read only – long)

The total amount of time, in milliseconds, that was spent retrying due to unavailable local ports when sending the message. Will be null if the message has not yet been sent or if in "preview" mode.

`timeToFirstByte` (read only – long)

The "time to first byte" measurement for this message. Will be null if the message has not yet been sent or if in "preview" mode.

`timeToLastByte` (read only – long)

The "time to last byte" measurement for this message. Will be null if the message has not yet been sent or if in "preview" mode.

Message Methods

`object getChild(string childName)`

Always returns null.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter indicates the starting point of the path within this item's container hierarchy. For Messages, it may have the following values:

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to the Band that contains this item.

- "`Track`"

  The path is relative to the Track that contains this item.

- "`MessageClip`" or "`Clip`" (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip "nearest to" the item in terms of the parentage hierarchy).

- "`Chain`"

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain "nearest to" the item in terms of the parentage hierarchy).

- "`Group`"

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group "nearest to" the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- “`Page`”

  The path is relative to the Page that contains this item.

- “`Message`”

  The path is relative to this Message

- “`Destination`”

  The path is relative to this Message’s Target.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item. See the description of this method in the description of the Band object.

```
String or String[] getResponse(int indicator, string
     param)
```

Returns all or the specified portion of the response, if a response has been received. Can return null. Depending upon the situation, either null, String or array of String is returned.

The meaning of the "param" parameter depends upon the value of the "indicator" parameter.

The "indicator" parameter must be one of the following:

- `RESPONSE_TEXT`

  Returns the entire text of the response received as a string.

  The "param" parameter is not used.

- `RESPONSE_HTTP_BODY`

  Returns the body of the HTTP response received as a string.

  The "param" parameter is not used.

  This indicator value is valid only for responses that are HTTP-based.

- `RESPONSE_HTTP_STATUSTEXT`

  Returns the status text from the HTTP response received as a string.

  The "param" parameter is not used.

  This indicator value is valid only for responses that are HTTP-based.

- `RESPONSE_HTTP_HEADER`

  Returns one HTTP header from the response received as a string.

  The "param" parameter is the name of the HTTP header to be returned. If the response contains no HTTP header with the specified name, null is returned.

  This indicator value is valid only for responses that are HTTP-based.

- `RESPONSE_HTTP_HEADER_LIST`

  Returns a list of all HTTP headers in the response.

  The "param" parameter is not used.

  The return value is an array of Objects. Each object has two String property values: "name" (the name of the HTTP header) and "value" (the value of the HTTP header).

If an HTTP header appears more than once in the response, it will appear that many times in the array.

The items in the array are not in any particular order.

Here's an example of a Script that gets all HTTP headers from the response to the Message before the Script and outputs the list to the Result:

```
var msg = $context.currentItem.previousItem;

var headerList =
msg.getResponse(msg.RESPONSE_HTTP_HEADER_LIST);

if (headerList == null)

{


$context.result.postMessage($context.result.LEVEL_INFO,
"No headers.");

}

else

{

  var listAsText = "";

  for each (var header in headerList)

  {

    listAsText += "Name: " + header.name + ", Value: " +
header.value + "\n";

  }



$context.result.postMessage($context.result.LEVEL_INFO,

                              headerList.length + " HTTP
headers",

                              listAsText);

}
```

- `RESPONSE_HTTP_PROTOCOL`

  Returns the protocol value from the HTTP response received as a string.

The “param” parameter is not used.

This indicator value is valid only for responses that are HTTP-based.

- `RESPONSE_HTTP_STATUSCODE`

  Returns the status code from the HTTP response received as a string.

  The “param” parameter is not used.

  This indicator value is valid only for responses that are HTTP-based.

- `RESPONSE_TEXT_AS_XML`

  Parses the entire response text as XML, evaluates an XPath expression for that XML, and returns the result of that XPath expression.

  The “param” parameter contains the XPath expression.

  This indicator value is valid only for responses that are not HTTP-based.

  If no nodes in the XML match, null is returned. If there are one or more matching nodes, a string array is returned. (An array is returned even if there is only one match.)

- `RESPONSE_TEXT_AS_HTML`

  Parses the entire response text as HTML, evaluates an XPath expression for that HTML, and returns the result of that XPath expression.

  The “param” parameter contains the XPath expression.

  This indicator value is valid only for responses that are not HTTP-based.

  If no nodes in the HTML match, null is returned. If there are one or more matching nodes, a string array is returned. (An array is returned even if there is only one match.)

- `RESPONSE_TEXT_AS_JSON`

  Parses the entire response text as JSON, evaluates an XPath expression for that XML, and returns the result of that XPath expression.

  The “param” parameter contains the XPath expression.

  This indicator value is valid only for responses that are not HTTP-based.

If no item in the JSON match, null is returned. If there are one or more matching nodes, a string array is returned. (An array is returned even if there is only one match.)

- `RESPONSE_HTTP_BODY_AS_XML`

  Parses the body of the HTTP response received as XML, evaluates an XPath expression for that XML, and returns the result of that XPath expression.

  The “param” parameter contains the XPath expression.

  This indicator value is valid only for responses that are HTTP-based.

  If no nodes in the XML match, null is returned. If there are one or more matching nodes, a string array is returned. (An array is returned even if there is only one match.)

- `RESPONSE_HTTP_BODY_AS_HTML`

  Parses the body of the HTTP response received as HTML, evaluates an XPath expression for that HTML, and returns the result of that XPath expression.

  The “param” parameter contains the XPath expression.

  This indicator value is valid only for responses that are HTTP-based.

  If no nodes in the HTML match, null is returned. If there are one or more matching nodes, a string array is returned. (An array is returned even if there is only one match.)

- `RESPONSE_HTTP_BODY_AS_JSON`

  Parses the body of the HTTP response received as JSON, evaluates an XPath expression for that JSON, and returns the result of that XPath expression.

  The “param” parameter contains the XPath expression.

  This indicator value is valid only for responses that are HTTP-based.

  If no items in the JSON match, null is returned. If there are one or more matching nodes, a string array is returned. (An array is returned even if there is only one match.)

If the “indicator” parameter is omitted, null or undefined, “RESPONSE_TEXT” will be assumed.

`String getMessage(int indicator)`

Returns all or the specified portion of the message that is to be sent.

The "indicator" parameter must be one of the following:

- `MESSAGE_TEXT`

  Returns the entire text of the message that is to be sent.

  This indicator value is valid only for messages that are not HTTP-based.

- `MESSAGE_HTTP_BODY`

  Returns the body of the HTTP message that is to be sent.

  This indicator value is valid only for messages that are HTTP-based.

- `MESSAGE_HTTP_HEADERS`

  The current value of the (optional) HTTP headers to be sent with the message is returned. Can be null. A text string is returned with new-lines between the HTTP headers.

  This indicator value is valid only for messages that are HTTP-based.

If the "indicator" parameter is omitted, null or undefined, "MESSAGE_HTTP_BODY" will be assumed for HTTP-based messages, or "MESSAGE_TEXT" will be assumed for non-HTTP-based messages.

`String setMessage(string value, int indicator)`

Sets all or the specified portion of the message that is to be sent.

The "value" parameter is the value to be placed into the message.

The "indicator" parameter must be one of the following:

- `MESSAGE_TEXT`

  Sets the entire text of the message that is to be sent from the "value" parameter.

  This indicator value is valid only for messages that are not HTTP-based.

- `MESSAGE_HTTP_BODY`

  Sets the body of the HTTP message that is to be sent from the "value" parameter.

  This indicator value is valid only for messages that are HTTP-based.

- `MESSAGE_HTTP_HEADERS`

  The "value" parameter contains the (optional) HTTP headers to be sent with the message. Can be null. The value should be a text string with new-lines between the HTTP headers.

  This indicator value is valid only for messages that are HTTP-based.

If the "indicator" parameter is omitted, null or undefined, "MESSAGE_HTTP_BODY" will be assumed for HTTP-based messages, or "MESSAGE_TEXT" will be assumed for non-HTTP-based messages.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an "abort" – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

`String[] getResourceLinksFromHTMLResponse()`

If the response to this Message is HTML, the HTML is parsed and any links on the HTML page to external resources are extracted and returned as an array of URLs.

If there is no response, or the response is not HTML, null is returned.

BrowserAction Object

Represents a Browser Action within the Composition.

Browser Action Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent Clip.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this Browser Action.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Browser Action".

`children` (read only – array of objects)

Always null.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If "Renew parallel repeats" is not enabled for this item, or it doesn't repeat in parallel, this value will always be zero.

`target` (read only – Target object)

The Target object for this Browser Action. Can be null.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REQUEST_OPERATION_PARAMS` (read only – integer)

A constant that can be passed in calls to the “getRequest” and “setRequest” methods. (See the descriptions of the “getRequest” and “setRequest” methods.)

`RESPONSE_OUTPUTS` (read only – integer)

A constant that can be passed in calls to the “getResponse” method. (See the description of the “getResponse” method.)

BrowserAction Methods

`object getChild(string childName)`

Always returns null.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter indicates the starting point of the path within this item’s container hierarchy. For BrowserActions, it may have the following values:

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to the Band that contains this item.

- “`Track`”

  The path is relative to the Track that contains this item.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- “`Page`”

  The path is relative to the Page that contains this item.

- “`Browser Action`”

  The path is relative to this Browser Action.

- “Destination”

  The path is relative to this Browser Action’s Target.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item. See the description of this method in the description of the Band object.

```
void getRequest(int indicator)
```

Returns the specified portion of the Browser Action request that will be performed when the Browser Action is executed.

The “indicator” parameter must be the following value (only one possible value is currently supported):

- `REQUEST_OPERATION_PARAMS`

  Returns the “parameter” values for the Browser Action. An array of up to three String values is returned. The array will always have a size of three elements. If the Browser Action has less than three parameters, the extra elements of the returned array will contain nulls.

```
void setRequest(Object value, int indicator)
```

Sets the specified portion of the Browser Action request that will be performed when the Browser Action is executed.

The “value” parameter contains the value(s) to be placed into the request.

The “indicator” parameter must be the following value (only one possible value is currently supported):

- `REQUEST_OPERATION_PARAMS`

  Sets the “parameter” values for the Browser Action. The “value” parameter must be an array of up to three String values, representing up to three “parameter” values to be placed into the request. Any and all existing parameteres in the request will be removed and replaced with these new values.

`void getResponse(int indicator)`

Returns the specified portion of the results of the Browser Action that was performed.

The "indicator" parameter must be the following value (only one possible value is currently supported):

- `RESPONSE_OUTPUTS`

  Returns a JavaScript associative array containing the "output" values from the Browser Action. The "keys" of the array are the names of the output values, and the values associated with the "keys" are the output values themselves.

  If the Browser Action did not return any "output" values, null is returned.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an "abort" – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

Delay Object

Represents a Delay object within the Composition.

Delay Properties

`name` (string)

The name of this item.

An item's name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string "Delay".

`children` (array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item's parent's array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item's parent's array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where "`item`" is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item's parent's array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

Always -1 for a Delay.

`playNumber` (read only – integer)

Always 0 for a Delay.

`playNumberBeforeRenewal` (read only – integer)

Always 0 for a Delay.

`playNumberWithinRenewal` (read only – integer)

Always 0 for a Delay.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the "setRepeat" method. (See the description of the "setRepeat" method.)

Delay Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The "pathType" parameter indicates the starting point of the path within this item's container hierarchy. For Delays, it may have the following values:

- "`Composition`"

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- "`Band`"

  The path is relative to the Band that contains this item.

- "`Track`"

  The path is relative to the Track that contains this item.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- “`Switch`”

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- “`Delay`”

  The path is relative to this Delay.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
               control, int distributionType, long
               distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- `REPEAT_TIMING_PARALLEL`
- `REPEAT_TIMING_SERIAL`

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

Script Object

Represents a Script object within the Composition.

Script Properties

`name` (string)

The name of this item.

An item’s name can be changed by setting this property, with the following restrictions:

The name can only be changed before any activity has occurred for this item. It cannot have been started playing yet, cannot have started repeating yet, no other actions for it can have occurred yet (no other properties of it can have been set).

The name must be a legal item name (255 chars max, no square brackets or slashes), and must not be the same name as another item in the same container.

`parent` (read only – object)

The parent object of this item.

`propertyList` (read only – object)

A PropertyList object that allows access to all of the Custom Properties contained in this object.

`systemPropertyList` (read only – object)

A SystemPropertyList object that allows access to all of the System Properties contained in this object.

`type` (read only – string)

The string “Script”.

`children` (array of objects)

An array of objects representing the children of this object, if any. Null if the object has no children.

`index` (read only – integer)

Returns the position (zero-based index) of this item’s parent’s array of children.

`nextItem` (read only – object)

Returns the next item after this one in this item’s parent’s array of children, or null of this is the last item. In other words, returns the next sibling in the object hierarchy. Equivalent to:

```
item.parent.children(item.index + 1)
```

where “`item`” is the current item.

`previousItem` (read only – object)

Returns the previous item before this one in this item’s parent’s array of children, or null of this is the first item. In other words, returns the next sibling in the object hierarchy. . Equivalent to:

```
item.parent.children(item.index - 1)
```

where "`item`" is the current item.

`forEachValue` (read only – string, number, date/time, or null)

If this item is repeating due to a "for-each" repeat, this property contains the for-each value associated with this instance of the object. This will be a value in the array used for the for-each.

If this item is not repeating due to a "for-each" repeat, this property will be null.

`repeatIndex` (read only – integer)

An integer value that represents the "repeat index" of this item if it repeats, according to the context in which the current script is executing. The first repeat starts at index zero. The value is -1 if the current item does not repeat or has a repeat count of one (in other words, it is -1 if it doesn't actually repeat).

`playNumber` (read only – integer)

An integer value that represents the "play ordinal number" of this item. Starting with the number 0, each play is assigned a unique number. The numbers are contiguous (no gaps).

Play number sequences are maintained within the item's parent only. For example, if a parent item repeats, then the child items inside each repeat of the parent will have their own play number sequence starting at 0.

An item will only have a non-zero play number if it repeats. The play number is equivalent to the repeat index, except that the play number is 0 for items that don't repeat or have a repeat count of one, whereas the repeat index would be -1 in those cases.

`playNumberBeforeRenewal` (read only – integer)

If this item repeats in parallel with the "Renew parallel repeats" option enabled, this is an integer value that is the "playNumber" value of the original parallel repeat for this item if it has been renewed because a prior parallel repeat ended. If this is the original parallel repeat, the value of `playNumberBeforeRenewal` will be equal to the value of `playNumber`. If this item does not repeat in parallel, the value will be zero.

For example, if parallel repeat number 5 of the item ends, but the Renew parallel repeats checkbox is enabled, the ending repeat will be replaced with a new, replacement repeat. The new repeat will have new `repeatIndex` and `playNumber` values (according to how many other repeats have already occurred). However, the `playNumberBeforeRenewal` value will still be 5 in this example.

This property is always 0 for Composition, Checkpoint, Delay, and Target.

`playNumberWithinRenewal` (read only – integer)

If this item repeats in parallel with the “Renew parallel repeats” option enabled, this is an integer value that is the ordinal of the current repeat within the sequence of repeat renewals. For example, if this is the original repeat, this value will be zero, but if this is the third renewal of the original repeat this value will be 3.

If “Renew parallel repeats” is not enabled for this item, or it doesn’t repeat in parallel, this value will always be zero.

`REPEAT_TIMING_PARALLEL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TIMING_SERIAL` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_TYPE_COUNT_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

`REPEAT_DISTRIBUTION_CONSTANT` (read only – integer)

A constant that can be passed in calls to the “setRepeat” method. (See the description of the “setRepeat” method.)

Script Methods

`object getChild(string childName)`

Returns a specific child object by name, or null if there is no child with the specified name.

`object getItemViaPath(string pathType, string path)`

Given a path to an item in the Composition object hierarchy, returns the object in the hierarchy that represents that item.

The “pathType” parameter indicates the starting point of the path within this item’s container hierarchy. For Scripts, it may have the following values:

- “`Composition`”

  The path is relative to the Composition as a whole. (Therefore, the path must start with the name of a Band.)

- “`Band`”

  The path is relative to the Band that contains this item.

- “`Track`”

  The path is relative to the Track that contains this item.

- “`MessageClip`” or “`Clip`” (either one is accepted)

  The path is relative to the Clip that contains this item. If nested within multiple Clips, the path is relative to the lowest-level containing Clip (the Clip “nearest to” the item in terms of the parentage hierarchy).

- “`Chain`”

  The path is relative to the Chain that contains this item. If nested within multiple Chains, the path is relative to the lowest-level containing Chain (the Chain “nearest to” the item in terms of the parentage hierarchy).

- “`Group`”

  The path is relative to the Group that contains this item. If nested within multiple Groups, the path is relative to the lowest-level containing Group (the Group “nearest to” the item in terms of the parentage hierarchy).

- “`Transaction`”

  The path is relative to the Transaction that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing Transaction (the Transaction “nearest to” the item in terms of the parentage hierarchy).

- “`If`”

  The path is relative to the If that contains this item. If nested within multiple Transactions, the path is relative to the lowest-level containing If (the If “nearest to” the item in terms of the parentage hierarchy).

- `“Switch”`

  The path is relative to the Switch that contains this item. If nested within multiple Switches, the path is relative to the lowest-level containing Switch (the Switch “nearest to” the item in terms of the parentage hierarchy).

- `“Page”`

  The path is relative to the Page that contains this item.

- `“Script”`

  The path is relative to this Script.

The “propertyPath” parameter contains a property path as specified for “In Situ Substitution Specifications” (see the separate document on ISSEs).

```
void clearRepeat()
```

Removes any current repeating specification from this item (so that it will play exactly once).

```
void setRepeat(int timingType, intRepeat Type, long
              control, int distributionType, long
              distribution)
```

Sets a new repeating specification for this item.

The “timingType” parameter indicates which type of repeat timing is to be used. It must be one of the following values:

- `REPEAT_TIMING_PARALLEL`
- `REPEAT_TIMING_SERIAL`

The “repeatType” parameter indicates what sort of value is contained in the “control” parameter. Currently the “repeatType” parameter must always be set to “REPEAT_TYPE_COUNT_CONSTANT”.

The “control” parameter is the count of the number of repeats to be performed. If the value is less than or equal to zero, the item will not be played at all.

The “distributionType” parameter indicates what sort of value is contained in the “distribution” parameter. Currently the “distributionType” parameter must always be set to “REPEAT_DISTRIBUTION_CONSTANT”.

The “distribution” parameter is a time length, in milliseconds, by which the start of each repeat is to be offset from the start of the prior repeat. This value only applies to parallel repeats, and must be set to zero for serial repeats.

`void endRepeat()`

Requests that repeating for this item be ended. If this item is not currently playing, no action is taken.

Note that this is not an "abort" – repeating will be ended after any currently playing individual repeat of the item completes.

This method is permitted only for serial repeating and for parallel repeating with the "renewal" option. It is not supported for parallel repeating without repeat renewal and an error will be generated if it is attempted to call this method for such a repeat.

For items that repeat serially, this call ends the serial repeating.

For items that repeat in parallel with "parallel repeat renewal", this call ends the renewals for the current sequence of parallel renewals, but does not affect other parallel renewal sequences. For example, consider the case of an item that repeats 3 times with parallel repeat renewal enabled. This item thus has 3 parallel renewal "lines" that are proceeding in parallel (there will always be 3 instances active at any given time). If this method is called from a Script that is playing from some repeat renewal that originated from the second repeat, then there will be no further renewals for the renewal line that started from the second repeat. However, renewals in the renewal lines that started from the first and third repeats will be unaffected.

PropertyList Object

An object that provides access to all of the Custom Properties contained within an item in the Composition. Contained in the "propertyList" property of objects that represent items in the Composition.

PropertyList Properties

All of the PropertyList properties are read-only.

`name` (string)

Always null.

`parent` (object)

The object who's properties are being accessed.

`propertyNames` (array of strings)

An array of all of the Custom Property names in the item. Can be null if the item has no Custom Properties.

`type` (string)

The string “PropertyList”.

`children` (array of objects)

Always null.

PropertyList Methods

`object getChild(string childName)`

Always returns null.

`void createProperty(string newPropertyName)`

Creates a new Custom Property within the item, with the specified name.

`var getPropertyValue(string propertyName)`

Returns the current value of the specified Custom Property. Can return null.

`void setPropertyValue(string propertyName, var newValue)`

Sets a new value into the specified Custom Property. The value can be set to null.

SystemPropertyList Object

An object that provides access to all of the System Properties contained within an item in the Composition. Contained in the “systemPropertyList” property of objects that represent items in the Composition.

SystemPropertyList Properties

All of the following properties are read-only.

`name` (string)

Always null.

`parent` (object)

The object whose properties are being accessed.

`type` (string)

The string “SystemPropertyList”.

`children` (array of objects)

Always null.

SystemPropertyList Methods

`object getChild(string childName)`

Always returns null.

`var getPropertyValue(string propertyName)`

Returns the current value of the specified System Property. Can return null.

`void setPropertyValue(string propertyName, var newValue)`

Sets a new value into the specified System Property. The value can be set to null.

Example

This example script outputs the following to the Result object:

1. A list of the names of all of the “current” items in the Composition.
2. A textual display of the entire Composition item tree, including the names, types, and properties of all items in the Composition.

```
var tree = "";


displayItem($context.composition, "");


$context.result.postMessage($context.result.LEVEL_INFO, "Composition object tree", tree);


var currentItems = "";

if ($context.currentBand == null)

  currentItems += "No Band";

else

  currentItems += "Band: " + $context.currentBand.name;

if ($context.currentTrack == null)

  currentItems += ", No Track";

else

  currentItems += ", Track: " + $context.currentTrack.name;

if ($context.currentClip == null)

  currentItems += ", No Clip";

else

  currentItems += ", Clip: " + $context.currentClip.name;

if ($context.currentChain == null)

  currentItems += ", No Chain";

else

  currentItems += ", Chain: " + $context.currentChain.name;

if ($context.currentPage == null)

  currentItems += ", No Page";

else
```

```
  currentItems += ", Page: " + $context.currentPage.name;

if ($context.currentGroup == null)

  currentItems += ", No Group";

else

  currentItems += ", Group: " + $context.currentGroup.name;

if ($context.currentMessage == null)

  currentItems += ", No Message";

else

  currentItems += ", Message: " + $context.currentMessage.name;

if ($context.currentBrowserAction == null)

  currentItems += ", No BrowserAction";

else

  currentItems += ", Browser Action: " + $context.currentBrowserAction.name;

if ($context.currentCheckpoint == null)

  currentItems += ", No Checkpoint";

else

  currentItems += ", Checkpoint: " + $context.currentCheckpoint.name;

if ($context.currentDelay == null)

  currentItems += ", No Delay";

else

  currentItems += ", Delay: " + $context.currentDelay.name;

if ($context.currentScript == null)

  currentItems += ", No Script";

else

  currentItems += ", Script: " + $context.currentScript.name;

if ($context.currentTarget == null)

  currentItems += ", No Target";

else

  currentItems += ", Target: " + $context.currentTarget.name;


$context.result.postMessage($context.result.LEVEL_INFO, "Current items", currentItems);
```

```
function displayItem(item, indent)
{
  if (item == null)
  {
    tree += indent + "(null)";
  }
  else
  {
    tree += indent + "\"" + item.name + "\", type: " + item.type + "\n";

    tree += indent + "  Parent: ";
    if (item.parent == null)
    {
      tree += "(none)\n";
    }
    else
    {
      tree += item.parent.name + "\n";

      var parentChild = item.parent.getChild(item.name);
      if (parentChild == null)
        $context.composition.abort("Unable to get child from parent, object \"" + item.name +
               "\".");
      else if (parentChild != item)
        $context.composition.abort("Parent's child does not match, object \"" + item.name +
               "\".");
    }

    if (item.type == "Message" || item.type="Browser Action")
    {
      var target = item.target;
  if (target != null)

  {

    tree += indent + "  Target:\n";

    tree += indent + "    \"" + target.name + "\", type: " + target.type + "\n";

  }

}

else if (item.type == "Clip")

{

  var targets = item.targets;

  if (targets != null)

  {

    tree += indent + "  Targets:\n";

    for (var i = 0; i < targets.length; i++)

    {

      tree += indent + "    \"" + targets[i].name + "\", type: " + targets[i].type + "\n";

    }

  }

}


var propertyList = item.propertyList;

if (propertyList != null)

{

  var propertyNames = propertyList.propertyNames;

  if (propertyNames != null)

  {

    tree += indent + "  Properties:\n";


    for (var i = 0; i < propertyNames.length; i++)

    {

      tree += indent + "    Name: \"" + propertyNames[i] + "\", Value: \"" +
           propertyList.getPropertyValue(propertyNames[i]) + "\"\n";
      propertyList.setPropertyValue(propertyNames[i], "A string value");

      if (propertyList.getPropertyValue(propertyNames[i]) != "A string value")

        $context.composition.abort("Unable to set value of property \"" + propertyNames[i]
           + "\" to a String.");



      propertyList.setPropertyValue(propertyNames[i], 123);

      if (propertyList.getPropertyValue(propertyNames[i]) != 123)

        $context.composition.abort("Unable to set value of property \"" + propertyNames[i]
           + "\" to an Integer.");



      propertyList.setPropertyValue(propertyNames[i], 123.456);

      if (propertyList.getPropertyValue(propertyNames[i]) != 123.456)

        $context.composition.abort("Unable to set value of property \"" + propertyNames[i]
           + "\" to a Float.");



      propertyList.setPropertyValue(propertyNames[i], new Date("Sun Sep 24 14:15:16 PDT
           2006"));

      if (propertyList.getPropertyValue(propertyNames[i]).toString() != "Sun Sep 24
           14:15:16 PDT 2006")

        $context.composition.abort("Unable to set value of property \"" + propertyNames[i]
           + "\" to a DateTime.");



      propertyList.setPropertyValue(propertyNames[i], null);

      if (propertyList.getPropertyValue(propertyNames[i]) != null)

        $context.composition.abort("Unable to set value of property \"" + propertyNames[i]
           + "\" to null.");

    }

  }

}



var children = item.children;

if (children != null)

{

  tree += indent + "  Children:\n";
```

```
      for (var i = 0; i < children.length; i++)
      {
        displayItem(children[i], indent + "   ");
      }
    }
  }
}
```

SOASTA, Inc.
444 Castro St.
Mountain View, CA 94041
866.344.8766
http://www.soasta.com