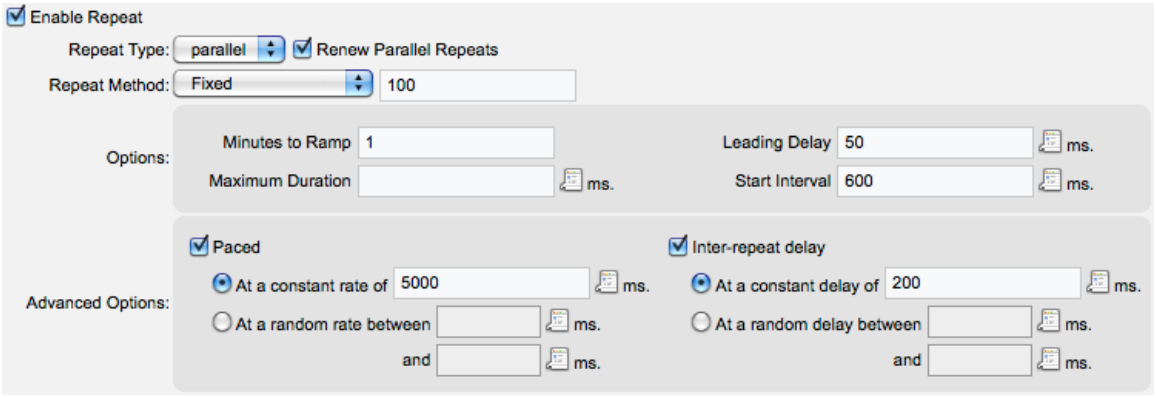
Repeating in CloudTest

CloudTest version

This document applies to CloudTest build 5060 and later.

Introduction

Repeating for an item in a Composition or a Clip is specified in a "Repeat" pane at the bottom of the Composition or Clip Editor. The following example is for a Track in the Composition Editor.



Repeat Type

There are two basic types of repeating:

- Parallel repeating – All of the repeats play simultaneously in parallel.
- Serial repeating – The repeats play serially, one at a time, one after the other.

Serial repeating is allowed for any item type, although for certain item types (such as Checkpoints) the repeat count is only permitted to be either zero or one.

Parallel repeating is allowed in the following situations:

- All Tracks.
- Clips in a Timed Band.
- Any item inside a Timed Clip, as long as the item is not inside a Chain or Page.

There are numerous sub-features that apply to repeating. Not all features apply to both types of repeating.

Repeat Method

The following features control how many times the repeat occurs:

- Fixed

  A constant is given at definition time, and the repeat occurs that many times.

- Property: Fixed

  A Custom or Global Property is specified, and the repeat occurs as many times as specified by the value of the property at runtime.

The property value is obtained when repeating starts. Changing the property value after repeating has started has no effect on the number of repeats. The Scripting function "`endRepeat`" can be used to terminate repeating.

- Property: For Each

  A Custom or Global Property is specified, and the repeat occurs as many times as there are array values in the property at runtime, if the property value is an array. If the property value is a single value, the repeat occurs once. The items inside each repeat have access to the individual value that the repeat was initiated for through the Scripting property "`forEachValue`".

  The list of property values is obtained when repeating starts. Changing the property value after repeating has started has no effect on the number of repeats. The Scripting function "`endRepeat`" can be used to terminate repeating.

- Property: While

  A Custom or Global Property is specified, and the repeat continues as long as the property value evaluates to "true". Permitted for serial repeating only.

- Continuous

  The repeating occurs forever, until explicitly stopped by a Script or there is an error, or until the Composition is stopped. Permitted for serial repeating only.

Parallel Repeat Renewal

Parallel repeat renewal applies only to parallel repeats and is either on or off. If it is on, each parallel repeat, when it has completed, will be automatically replaced with a new, replacement repeat. Thus, the number of instances of the repeating item is kept at the specified number indefinitely (subject to any ramp-up).

Parallel repeat renewal is the recommended way to implement "Virtual Users", as it keeps the number of Virtual Users at the desired level throughout the life of the Composition.

Although parallel repeat renewal is commonly associated with Tracks, it can be used for any item that repeats in parallel.

Start Interval and Minutes to Ramp

Start Interval only applies to parallel repeating, and indicates that each parallel repeat is to be started at an offset from the start of the prior repeat by the specified number of milliseconds. This is how "ramp-up" is achieved for Tracks (a.k.a. Virtual Users), although Start Interval can be used for any item that repeats in parallel, not just Tracks.

The Start Interval value can be a constant specified at definition time, or can come from a Custom or Global Property whose value is used at runtime. If the value comes from a property, the property value is obtained when repeating starts. Changing the property value after repeating has started has no effect on the currently-active repeat process.

Minutes to Ramp is a convenience value that can be entered instead of Start Interval. If Minutes to Ramp is entered, a Start Interval is computed such that the ramp-up lasts for

the specified number of minutes. Entering a value in Minutes to Ramp causes Start Interval to be recomputed and vice versa.

Leading Delay

Leading Delay specifies that there is to be a delay (in milliseconds) before any of the repeating starts.

The leading delay value can be a constant specified at definition time, or can come from a Custom or Global Property whose value is used at runtime.

Leading Delay is often used with Tracks (Virtual Users) to cause ramp-up for a certain Track to not begin until some point within the Composition play. However, leading delay can be used for any type of item.

Maximum Duration

Maximum Duration puts an upper limit on the amount of time for which a repeat will continue repeating, over and above any count or other limitation. The limit is specified in milliseconds.

Maximum Duration can be specified for serial repeats and parallel repeat renewal only.

The maximum duration value can be a constant specified at definition time, or can come from a Custom or Global Property whose value is used at runtime. If the value comes from a property, the property value is obtained when repeating starts. Changing the property value after repeating has started has no effect on the currently-active repeat process.

Maximum Duration will not interrupt or stop any individual repeat in the middle of play. It is examined each time a new individual repeat would ordinarily be started, in order to determine whether or not to continue repeating.

Pacing

Pacing specifies that delays are to be automatically inserted between repeats when necessary in order to cause the repeats to occur at a certain rate. The rate is specified in milliseconds, as in “every n milliseconds”.

Pacing can be specified for serial repeats and parallel repeat renewal only.

There are two basic types of pacing:

- Even

  The repeats are to occur at a regular, constant rate.

- Random

  The repeats are to occur at a random rate between a minimum and maximum.

The pacing rate information can be specified via constant(s) at definition time, or via Custom or Global Property values at runtime. Therefore, the possible combinations are:

- Even, constant

  Even pacing with a constant rate specified at definition time.

- Even, property

  Even pacing with the rate coming from the value of a property at runtime.

- Random, constant

  Random pacing with the minimum and maximum rates specified as constants at definition time.

- Random, property

  Random pacing with the minimum and maximum rates coming from Custom or Global Properties at runtime. The minimum and maximum values are taken from the properties when repeat first starts; they are not re-evaluated for each delay.

Inter-repeat delay

Inter-repeat delay specifies that there is to be a delay (gap) between the end of one repeat and the start of the next repeat. It applies to serial repeats and parallel repeat renewal only.

In the case of serial repeat, the delay occurs between the end of each serial repeat and the start of the next one.

In the case of parallel repeat renewal, the gap occurs between the end of each parallel repeat and the replacement ("renewal") repeat that it is replaced with.

The length of the inter-repeat delay can be specified in the following ways:

- Constant length

  A constant number of milliseconds is given at definition time.

- From a property

  A Custom or Global Property is specified, and the duration of the delay comes from the value of the property at runtime. The value is taken from the property when repeating first starts; it is not re-evaluated for each delay.

- Random length between two constants

  A constant minimum and maximum length is specified at definition time, and each delay will be a random value selected within that range.

- Random length between two property values

  Custom or Global Properties are specified for the minimum and maximum lengths. Each delay will be a random value selected within the range specified by the property values. The minimum and maximum values are taken from the properties when repeat first starts; they are not re-evaluated for each delay.

Access to the repeat process from Scripts

Scripts have access to various aspects of the repeat process, and can change some aspects of the repeat behavior. Refer to the Scripting documentation for further information on the following Script properties and methods:

- `forEachValue`
- `playNumber`
- `playNumberBeforeRenewal`
- `playNumberWithinRenewal`
- `repeatIndex`
- `endRepeat()`
- `clearRepeat()`
- `setRepeat()`

Repeat isolation

When an item repeats, copies are made of the item and its child objects to play for that repeat. This is to keep each repeat isolated, so that any changes made to the items during repeat play do not affect other repeats.

For example, suppose a Chain repeated and contained several Messages. For each repeat of the Chain, copies are made of the Chain and the Messages it contains. Therefore, if items are modified during a repeat of the Chain, such as changes to Custom Property values, those changes will only affect that repeat. Other repeats of the Chain and its contents will not "see" those changes.

However, if a Script were to make a change to a parent of the Chain, all of the repeats of the Chain will have the same parent, and thus all repeats would see those changes.

Repeat isolation exception for repeat count one

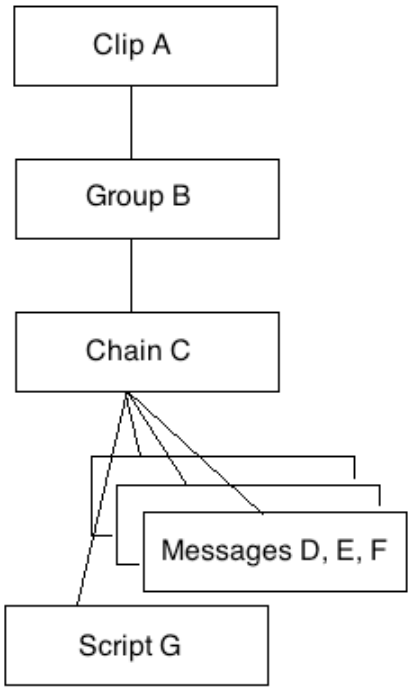
If it becomes time to play an item that repeats, but the repeat count evaluates to one, meaning that the item is to play only one time, then as an optimization CloudTest does not make a copy of the item and its children, but instead plays the item directly, as if it did not repeat.

In this case the values of the Scripting properties "`playNumber`" and "`repeatCount`" will diverge. In this case, "`playNumber`" will be zero to indicate the first play of the item, but "`repeatCount`" will be `-1`, since the item is not actually repeating.

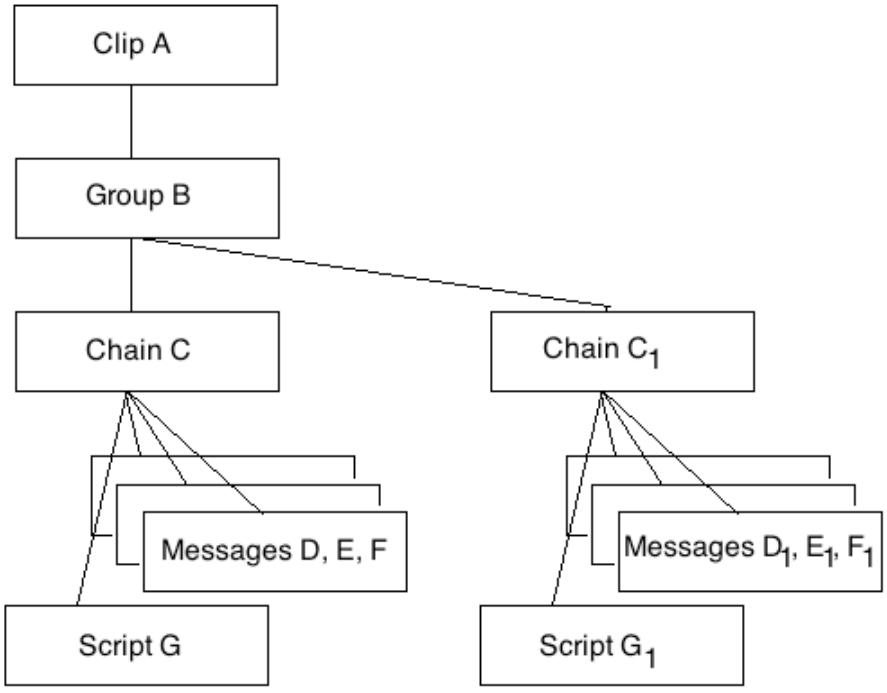
Repeat isolation ramifications for Scripts

A Script that traverses the Composition object hierarchy must sometimes be aware of the copies that are made for repeating items. Depending upon the method used to traverse the hierarchy, the Script may sometimes encounter the original item while at other times it may encounter a copy.

Consider the following Clip A, which contains a Group B, which contains a Chain C. Chain C contains Messages D, E, and F as well as Script G.



Now consider that Chain C repeats. Each repeat play of Chain C will play a copy of Chain C, thus the hierarchy when a copy of Chain C plays will be:



Clip A and Group B have not been changed. However, a copy of Chain C, Chain C1, has been created to play. All of the contents of the Chain have been copied as well.

When Script G1 traverses the Composition object hierarchy, which items it encounters will depend upon the situation. The "current item" properties of the `$context` object

will refer to the copies. Likewise, any traversals that start out from Script G1 itself or any contents of Chain C1 will refer to the copies.

However, any traversals that start from the top, such as Clip A or Group B, will always encounter the original Chain C and contents, not the copy.

To illustrate this, the following table shows examples of various scripting object hierarchy traversal calls or properties, and what object they would reference, if Script G1 made the reference.

Property or method used in Script G1	**Object referenced**
`$context.currentChain`	Chain C1
`$context.currentGroup`	Group B
`$context.currentItem.parent`	Chain C1
`$context.currentItem.parent.parent`	Group B
`$context.currentItem.parent.parent.children`	Chain C
`$context.currentGroup.children`	Chain C
`$context.currentGroup.getChild("Chain C")`	Chain C
`$context.currentChain.children`	Messages D1, E1, F1, Script G1
`$context.currentChain.parent.children[0].children`	Messages D, E, F, Script G
`$context.currentChain.getChild("Message E")`	Message E1
`$context.currentGroup.getChild( "Chain C").getChild("Message E")`	Message E
`$context.currentClip.getItemViaPath( "Group B/Chain C")`	Chain C
`$context.currentClip.getItemViaPath( "Group B/Chain C/Message D")`	Message D
`$context.currentChain.getItemViaPath( "Message D")`	Message D1
`$context.currentChain.getItemViaPath( "../Chain C/Message D")`	Message D