Advanced Distributed Learning Initiative

Sharable Content Object
Reference Model
(SCORM®) 2004 2nd Edition
Addendum

Version 1.2

April 15, 2005

This page intentionally left blank.

Advanced Distributed Learning

SCORM 2004 2nd Edition
Addendum

Version 1.2

Available at ADLNet.org
(http://www.adlnet.org/)

For questions and comments visit the ADL Help & Info
Center at ADLNet.org

This page intentionally left blank.

Table of Contents

This page intentionally left blank.

SECTION 1
Introduction

This page intentionally left blank.

1.1.   Purpose

The purpose of this document is to track all reported issues with SCORM 2004 2nd Edition that would require updates to the SCORM 2004 2nd Edition documentation suite. This document captures those issues and describes the corrections needed to address them. The information contained in this document supersedes information contained in the SCORM 2004 2nd Edition documentation suite. Vendors should adhere to all changes to SCORM 2004 2nd Edition as described in this document. The SCORM 2004 Conformance Test Suite will be updated to reflect the changes described in this document.

This document should be used in conjunction with the current SCORM 2004 2nd Edition documentation suite until a new edition of SCORM 2004 is published by ADL. This document will be updated to include additional corrections should they become known.

Please submit any additional known issues with SCORM 2004 2nd Edition to the ADL Technical Team via the Help & Info Center on ADLNet.org.

This document is divided into three major sections. Section 1 explains the purpose of the SCORM 2004 2nd Edition addendum. Section 2 describes addenda that are related to defects in the SCORM 2004 2nd Edition. These defects may affect conformance to SCORM 2004, depending on current implementations. Section 3 describes addenda that are clarifications or enhancements that do not affect SCORM 2004 conformance.

This page intentionally left blank.

SECTION 2
SCORM 2004 Defect Addenda

This page intentionally left blank.

2.1. Handling of Invalid SetValue() Requests for Data Model Element Collections

The SCORM 2004 RTE Version 1.3.1 does not clearly describe how to handle invalid `SetValue()` requests invoked against data model elements in a "to-be" created record of a data model element collection. More specifically, there is no specific statement of when a data model element record should be created in response to a `SetValue()` request invoked against one of its children, increasing the count of its containing collection.

For example, assume that the value for `cmi.comments_from_learner._count` is zero (the SCO has not set any comments from learner) and the SCO attempts to make the following `SetValue()` request:

```
SetValue("cmi.comments_from_learner.0.comment","{lang=}")
```

This `SetValue()` request is invalid because the value provided for the language delimiter is not a valid `language_type` – it cannot be an empty string (refer to the SCORM 2004 RTE Version 1.3.1, *Section 4.1.1.7: Data Types* for requirements of a valid `language_type`). In this case, the LMS is required to return `false` and set the API error code to `406 – Data Model Element Type Mismatch`.

This issue is experienced when continuing the example. SCORM does not clearly define how an LMS should respond to the following `GetValue()` requests when they are invoked immediately following the previous `SetValue()` request.

```
GetValue("cmi.comments_from_learner.0.comment)

GetValue("cmi.comments_from_learner._count)
```

2.1.1. Rationale for Change

Invalid `SetValue()` requests (regardless of the data model element) always result in an LMS returning `false` and setting the API error code. The LMS is required to not alter the state of the data model element's value. In the case where these `SetValue()` requests are invoked against data model elements in a "to-be" created record, not altering the state of the data model element implies that no record is created and that the containing collection's size does not change.

2.1.2. SCORM Update

Section 4.1.1.3: Handling Collections found in the SCORM RTE Version 1.3.1 will be updated to include the following required behavior:

Failure to set a data model element in a "to-be" created data model record does not result in any data being persisted and does not increase the containing collection size.

The following child data model elements are affected by this change. If any of these elements is set successfully, it would cause a newly created record to be added to the containing collection, increasing the collection's count by 1.

- `cmi.objectives.`*n*`.id`
- `cmi.interactions.`*n*`.id`
- `cmi.interactions.`*n*`.objectives.`*n*`.id`
- `cmi.comments_from_learner.`*n*`.comment`
- `cmi.comments_from_learner.n.location`
- `cmi.comments_from_learner.`*n*`.timestamp`

When an LMS receives an invalid `SetValue()` request against one of these data model elements, the LMS will return `false` and set the appropriate API error code. The size of the containing collection shall not be incremented. If a `_count` request is made prior to the invalid `SetValue()` request and a `_count` request is made immediately after the invalid `SetValue()` request, then the value returned by both calls shall be identical (i.e., the collection size has not changed).

An update will be made to define that if an LMS receives a GetValue() request immediately following an invalid SetValue() request, then the behavior shall be:

(1) Return an empty characterstring ("")
(2) Set the API Error Code to `301 – General Get Failure`

2.2. Ambiguous Pseudo-Code in Case #4 of Choice Sequencing Request Process

This addendum addresses a discrepancy found in Choice Sequencing Request Process (SB.2.9) of the SCORM 2004 Sequencing and Navigation (SN) Version 1.3.1. The discrepancy can be found in Case #4 which starts on line 11. Case #4 deals with the Choice Sequencing Request scenario where the target activity is an ancestor of the Current Activity. In this case, the Activity Tree traversal will be up the "active" path rather than across the tree; however, the pseudo code includes an evaluation of Constrained Choice attribute that requires an ambiguous traversal either Backward or Forward in the Activity Tree relative to the constrained activity.

2.2.1. Rationale for Change

When the target of a choice navigation request is an ancestor of the current activity (Case #4), the Constrained Choice attribute will have no effect. In Case #4, only the target or its immediate "next" activity (if the target is a cluster – SB.2.9, line 14), could possibly be identified for delivery, which is the intended effect of the Constrained Choice Attribute. In this case, the evaluation of Constrained Choice adds no value and potentially leads to an ambiguous traversal direction.

2.2.2. SCORM Update

The evaluation of the Constrained Choice attribute is unnecessary and ambiguous when performed during Case #4 of the Sequencing Choice Process. The Choice Sequencing Request Process (SB.2.9) will be updated to delete the pseudo-code that references the constrained activity in Case #4 (this includes lines 11.3, 11.4.2.*, and 11.5.*).

The updated Choice Sequencing Process is reproduced in whole for easy reference.

Choice Sequencing Request Process [SB.2.9] (for a target activity; may return a delivery request; may change the *Current Activity*; may return an exception code):		
Reference: Activity is Active AM.1.1; Activity is Suspended AM.1.1; Available Children AM.1.1; Check Activity Process UP.5; Choice Activity Traversal Subprocess SB.2.4; Current Activity AM.1.2; End Attempt Process UP.4; Flow Subprocess SB.2.3; Sequencing Control Mode Choice SM.1; Sequencing Control Choice Exit SM.1; Sequencing Rules Check Process UP.2; Terminate Descendent Attempts Process UP.3*; adlseq:constrainedChoice SCORM SN; adlseq:preventActivation SCORM SN*		
1.	**If** there is no target activity **Then**	There must be a target activity for choice
1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-1*)	Nothing to deliver
	End If	
2.	**If** the target activity is not the root of the activity tree **Then**	
2.1.	**If** the *Available Children* for the parent of the target activity does not	The activity is

	contain the target activity **Then**	currently not available
2.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-2*)	Nothing to deliver
	End If	
	End If	
3.	Form the activity path as the ordered series of activities from the root of the activity tree to the target activity, inclusive	
4.	**For** each activity in the activity path	
4.1.	Apply the *Sequencing Rules Check Process* to the activity and the *Hide from Choice sequencing* rules	Cannot choose something that is hidden
4.2.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	
4.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-3*)	Nothing to deliver
	End If	
	End For	
5.	**If** the target activity is not the root of the activity tree **Then**	
5.1.	**If** the *Sequencing Control Mode Choice* for the parent of the target activity is *False* **Then**	Confirm that control mode allow 'choice' of the target
5.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-4*)	Nothing to deliver
	End If	
	End If	
6.	**If** the *Current Activity* is *Defined* **Then**	Has the sequencing session already begun?
6.1.	Find the common ancestor of the *Current Activity* and the target activity	
7.	**Else**	
7.1.	Set common ancestor is the root of the activity tree	No, choosing the target will start the sequencing session
	End If	
8.	**Case:** *Current Activity* and target activity are identical	Case #1 - select the current activity
8.1.	**Break All** Cases	Nothing to do in this case
	End Case	
9.	**Case:** *Current Activity* and the target activity are siblings	Case #2 - same cluster; move toward the target activity
9.1.	Form the activity list as the ordered sequence of activities from the *Current Activity* to the target activity, exclusive of the target activity	We are attempted to walk toward the target activity. Once we reach the target activity, we don't need to test it.
9.2.	**If** the activity list is *Empty* **Then**	Nothing to choose

9.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)	Nothing to deliver
	End If	
9.3.	**If** the target activity occurs after the *Current Activity* in preorder traversal of the activity tree **Then**	
9.3.1.	traverse is *Forward*	
9.4.	**Else**	
9.4.1.	traverse is *Backward*	
	End If	
9.5.	**For** each activity on the activity list	
9.5.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the traverse direction	
9.5.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**	
9.5.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Choice Activity Traversal Subprocess*)	Nothing to deliver
	End If	
	End For	
9.6.	**Break All** Cases	
	End Case	
10.	**Case:** *Current Activity* and common ancestor are the same **Or** *Current Activity* is **Not** D*efined*	Case #3 - path to the target is forward in the activity tree
10.1.	Form the activity path as the ordered series of activities from the common ancestor to the target activity, exclusive of the target activity	
10.2.	**If** the activity path is *Empty* **Then**	
10.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)	Nothing to deliver
	End If	
10.3.	**For** each activity on the activity path	
10.3.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the *Forward* direction	
10.3.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**	
10.3.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Choice Activity Traversal Subprocess*)	Nothing to deliver
	End If	
10.3.3.	**If** *Activity is Active* for the activity is *False* **And** (the activity is **Not** the common ancestor **And** *adlseq:preventActivation* for the activity is *True*) **Then**	If the activity being considered is not already active, make sure we are allowed to activate it
10.3.3.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-6*)	Nothing to deliver
	End If	
	End For	
10.4.	**Break All** Cases	
	End Case	
11.	**Case:** Target activity is the common ancestor of the *Current Activity*	Case #4 - path to the target is backward in the activity tree

11.1.	Form the activity path as the ordered series of activities from the *Current Activity* to the target activity, inclusive	
11.2.	**If** the activity path is *Empty* **Then**	
11.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5)*	Nothing to deliver
	End If	
11.3.	**For** each activity on the activity path	
11.3.1.	**If** the activity is not the last activity in the activity path **Then**	
11.3.1.1.	**If** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	Make sure an activity that should not exit will exit if the target is delivered.
11.3.1.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-7*)	Nothing to deliver
	End If	
	End If	
	End For	
11.4.	**Break All** Cases	
	End Case	
12.	**Case:** Target activity is forward from the common ancestor activity	Case #5 - target is a descendent activity of the common ancestor
12.1.	Form the activity path as the ordered series of activities from the *Current Activity* to the common ancestor, inclusive	
12.2.	**If** the activity path is *Empty* **Then**	
12.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5)*	Nothing to deliver
	End If	
12.3.	Set constrained activity to *Undefined*	
12.4.	**For** each activity on the activity path	Walk up the tree to the common ancestor
12.4.1.	**If** the activity is not the last activity in the activity path **Then**	
12.4.1.1.	**If** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	Make sure an activity that should not exit will exit if the target is delivered
12.4.1.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-7*)	Nothing to deliver
	End If	
	End If	
12.4.2.	**If** constrained activity is *Undefined* **Then**	Find the closest constrained activity to the current activity
12.4.2.1.	**If** *adlseq:constrainedChoice* for the activity is *True* **Then**	
12.4.2.1.1.	Set constrained activity to activity	
	End If	
	End If	
	End For	
12.5.	**If** constrained activity is *Defined* **Then**	

12.5.1.	**If** the target activity is *Forward* in the activity tree relative to the constrained activity **Then**	
12.5.1.1.	traverse is *Forward*	'Flow' in a forward direction to see what activity comes next
12.5.2.	**Else**	
12.5.2.1.	traverse is *Backward*	'Flow' in a backward direction to see what activity comes next
	End If	
12.5.3.	Apply the *Choice Flow Subprocess* to the constrained activity in the traverse direction	
12.5.4.	Set activity to consider to the activity identified by the *Choice Flow Subprocess*	
12.5.5.	**If** the target activity is **Not** an available descendent of the activity to consider **And** (the target activity is **Not** the constrained activity **Or** the target activity is **Not** the activity to consider) **Then**	Make sure the target activity is within the set of 'flow' constrained choices
12.5.5.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-8*)	
	End If	
	End If	
12.6.	Form the activity path as the ordered series of activities from the common ancestor to the target activity, exclusive of the target activity	
12.7.	**If** the activity path is *Empty* **Then**	
12.7.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)	Nothing to deliver
	End If	
12.8.	**If** the target activity is forward in the activity tree relative to the *Current Activity* **Then**	Walk toward the target activity
12.8.1.	**For** each activity on the activity path	
12.8.1.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the *Forward* direction	
12.8.1.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**	
12.8.1.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Choice Activity Traversal Subprocess*)	Nothing to deliver
	End If	
12.8.1.3.	**If** *Activity is Active* for the activity is *False* **And** (the activity is **Not** the common ancestor **And** *adlseq:preventActivation* for the activity is *True*) **Then**	If the activity being considered is not already active, make sure we are allowed to activate it
12.8.1.3.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-6*)	Nothing to deliver
	End If	
	End For	

12.9.	**Else**	
12.9.1.	**For** each activity on the activity path	
12.9.1.1.	**If** *Activity is Active* for the activity is *False* **And** (the activity is **Not** the common ancestor **And** *adlseq:preventActivation* for the activity is *True*) **Then**	If the activity being considered is not already active, make sure we are allowed to activate it
12.9.1.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-6*)	Nothing to deliver
	End If	
	End For	
	End If	
12.10.	**Break All Cases**	
	End Case	
13.	**If** the target activity is a leaf activity **Then**	
13.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: the target activity; **Exception**: *n/a*)	
	End If	
14.	Apply the *Flow Subprocess* to the target activity in the *Forward* direction with consider children equal to *True*	The identified activity is a cluster. Enter the cluster and attempt to find a descendent leaf to deliver
15.	**If** the *Flow Subprocess* returns *False* **Then**	Nothing to deliver, but we succeeded in reaching the target activity - move the current activity
15.1.	Apply the *Terminate Descendent Attempts Process* to the common ancestor	
15.2.	Apply the *End Attempt Process* to the common ancestor	
15.3.	Set the *Current Activity* to the target activity	
15.4.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-9*)	Nothing to deliver
16.	**Else**	
16.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: for the activity identified by the *Flow Subprocess*; **Exception**: *n/a*)	
	End If	

pseudo code for choice sequencing request process

2.3.   Misevaluation of Traversal Direction

This addendum addresses an error discovered in the Flow Activity Traversal Subprocess (SB.2.2) of the SCORM 2004 SN Version 1.3.1. The error prevents the proper reversal of the flow traversal direction during the evaluation of a *Previous* sequencing request. This case only occurs when a *Forward Only* cluster is encountered and all of its children are skipped during the evaluation of a *Previous* sequencing request. In this case, the Flow Activity Traversal Subprocess needs to switch direction twice, once to move forward through the *Forward Only* cluster and then once again to continue to move backward (honoring the original request) from the *Forward Only* cluster to its predecessor.

2.3.1.   Rationale for Change

This error is a bug in the SCORM Sequencing pseudo code. A strict implementation of the SCORM Sequencing pseudo code would prevent an LMS from recognizing a change in traversal direction and from successfully passing the SCORM Sequencing Conformance Test Case CM-3b.

2.3.2.   SCORM Update

Line 3.3.1 of the Flow Activity Traversal Subprocess (SB.2.2) will be updated. The currently referenced (local variable) "traversal direction" will be updated to "previous traversal direction."

The updated Flow Activity Traversal Subprocess is reproduced in whole for easy reference.

Flow Activity Traversal Subprocess [SB.2.2] (for an activity, a traversal direction, and a previous traversal direction; returns the 'next' activity in a directed traversal of the activity tree and True if the activity can be delivered; may return an exception code):		
Reference: Check Activity Process UP.5; Flow Activity Traversal Subprocess SB.2.2; Flow Tree Traversal Subprocess SB.2.1; Sequencing Control Flow SM.1; Sequencing Rules Check Process UP.2		
1.	**If** *Sequencing Control Flow* for the parent of the activity is *False* **Then**	Confirm that 'flow' is enabled
1.1.	**Exit** *Flow Activity Traversal Subprocess* (**Deliverable**: *False*; **Next Activity**: the activity; **Exception**: *SB.2.2-1*)	
	End If	
2.	Apply the *Sequencing Rules Check Process* to the activity and its *Skipped* sequencing rules	
3.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	Activity is skipped, try to go to the 'next' activity
3.1.	Apply the *Flow Tree Traversal Subprocess* to the activity in the traversal direction and the previous traversal direction with consider children equal to *False*	

3.2.	**If** the *Flow Tree Traversal Subprocess* does not identify an activity **Then**	
3.2.1.	**Exit** *Flow Activity Traversal Subprocess* (**Deliverable**: *False*; **Next Activity**: the activity; **Exception**: exception identified by the *Flow Tree Traversal Subprocess*)	
3.3.	**Else**	
3.3.1.	**If** the previous traversal direction is *Backward* **And** the Traversal Direction returned by the *Flow Tree Traversal Subprocess* is *Backward* **Then**	Make sure the recursive call is considers the correct direction
3.3.1.1.	Apply the *Flow Activity Traversal Subprocess* to the activity identified by the *Flow Tree Traversal Subprocess* in the traversal direction and a previous traversal direction of *n/a*	Recursive call – make sure the 'next' activity is OK
3.3.2.	**Else**	
3.3.2.1.	Apply the *Flow Activity Traversal Subprocess* to the activity identified by the *Flow Tree Traversal Subprocess* in the traversal direction and a previous traversal direction of previous traversal direction	Recursive call – make sure the 'next' activity is OK
	End If	
3.3.3.	**Exit** *Flow Activity Traversal Subprocess* - (Return the results of the recursive *Flow Activity Traversal Subprocess*)	Possible exit from recursion
	End If	
	End If	
4.	Apply the *Check Activity Process* to the activity	Make sure the activity is allowed
5.	**If** the *Check Activity Process* returns *True* **Then**	
5.1.	**Exit** *Flow Activity Traversal Subprocess* (**Deliverable**: *False*; **Next Activity**: the activity; **Exception**: *SB.2.2-2*)	
	End If	
6.	**If** the activity is not a leaf node in the activity tree **Then**	Cannot deliver a non-leaf activity; enter the cluster looking for a leaf
6.1.	Apply the *Flow Tree Traversal Subprocess* to the activity in the traversal direction and a previous traversal direction of *n/a* with consider children equal to *True*	
6.2.	**If** the *Flow Tree Traversal Subprocess* does not identify an activity **Then**	
6.2.1.	**Exit** *Flow Activity Traversal Subprocess* (**Deliverable**: *False*; **Next Activity**: the activity; **Exception**: exception identified by the *Flow Tree Traversal Subprocess*)	
6.3.	**Else**	
6.3.1.	**If** the traversal direction is *Backward* **And** the traversal direction returned by the *Flow Tree Traversal Subprocess* is *Forward* **Then**	Check if we are flowing backward through a forward only cluster - must move forward instead
6.3.1.1.	Apply the *Flow Activity Traversal Subprocess* to the activity identified by the *Flow Tree Traversal Subprocess* in the *Forward* direction with the previous traversal direction of *Backward*	Recursive call – Make sure the identified activity is OK
6.3.2.	**Else**	

6.3.2.1.		Apply the *Flow Activity Traversal Subprocess* to the activity identified by the *Flow Tree Traversal Subprocess* in the traversal direction and a previous traversal direction of *n/a*	Recursive call – Make sure the identified activity is OK
		End If	
6.3.3.		**Exit** *Flow Activity Traversal Subprocess* - (Return the results of the recursive *Flow Activity Traversal Subprocess*)	Possible exit from recursion
	End If		
	End If		
7.	**Exit** *Flow Activity Traversal Subprocess* (**Deliverable**: *True*; **Next Activity**: the activity; **Exception**: *n/a*)		Found a leaf

pseudo code for flow activity traversal subprocess.

2.4. Measure Rollup Should not be Applied to Leaf Activities

This addendum addresses a side effect discovered in the Overall Rollup Process (RB.1.5) found in the SCORM 2004 SN Version 1.3.1. The side effect discovered prevents scores reported by SCOs from being utilized during sequencing evaluations. *Section 4.6.1: Overall Rollup Process* of the SCORM 2004 SN Version 1.3.1 defines when the Overall Rollup Process is applied through its extended rollup process and some rules constraining how that rollup is evaluated. Each time the Overall Rollup Process is invoked, it applies the various rollup subprocesses to each activity along the "active path" – the path from the Current Activity (typically a leaf) and the root of the Activity Tree. For each activity, even a Current Activity that happens to be a leaf, the Measure Rollup Process (RB.1.1) is applied. However, the purpose of the Measure Rollup Process is to aggregate (rollup) the measures of the target activity's children. If none of the activity's children have a measure or if the activity has no children, the resulting measure is "unknown."

2.4.1. Rationale for Change

A strict implementation of the SCORM Sequencing pseudo code would result in all delivered leaf activities having a measure of "unknown" because the Measure Rollup Process would interpret the lack of *counted measures* as an indication that the rolled up measure should be "unknown" (RB.1.1, line 5.2.1). For sequencing purposes, the "unknown" value would be used instead of any measure provided by a SCO (cmi.score.scaled), defeating the intent of mapping SCORM Run-Time Data to the associated activity's tracking data. The net result would prevent an LMS from successfully passing several of the measure-based SCORM Sequencing Conformance Test Cases (MS-*).

2.4.2. SCORM Update

Although *Section 4.6.1: Overall Rollup Process* of the SCORM 2004 SN Version 1.3.1 states that "Rollup rules have no effect if defined on a leaf activity – there is nothing to rollup", this instruction could be interpreted as applying only to Rollup Rule Descriptions, as defined in the Sequencing Definition Model, and not to general measure rollup. To ensure that there is no ambiguity in defined sequencing behaviors, the following bullet will be added to the SCORM 2004 SN Version 1.3.1 in Section 4.6.1:

- Measure rollup is not applied to leaf activities.

In addition, a clause will be added to the Overall Rollup Process (RB.1.5) that applies to line 3.1. The clause will ensure that the Measure Rollup Process (RB.1.1) is not applied to leaf activities. This change is normative behavior and will enforce that all implementation will utilize the measure reported by a SCO for sequencing purposes.

The updated Overall Rollup Process is reproduced in whole for easy reference.

Overall Rollup Process [RB.1.5] (for an activity; may change the tracking information for the activity and its ancestors):		
Reference: Activity Progress Rollup Process RB.1.3; Measure Rollup Process RB.1.1; Objective Rollup Process RB.1.2; Tracked SM.11; Tracking Model TM		
1.	Form the activity path as the ordered series of activities from the root of the activity tree to the activity, inclusive, in reverse order.	
2.	**If** the activity path is *Empty* **Then**	
2.1.	**Exit** *Overall Rollup Process*	Nothing to rollup
	End If	
3.	**For** each activity in the activity path	
3.1.	**If** the activity has children **Then**	Only apply Measure Rollup to non-leaf activities
3.1.1.	Apply the *Measure Rollup Process* to the activity	Rollup the activity's measure
	End If	
3.2.	Apply the appropriate *Objective Rollup Process* to the activity	Apply the appropriate behavior described in section RB.1.2, based on the activity's defined sequencing information
3.3.	Apply the *Activity Progress Rollup Process* to the activity	Apply the appropriate behavior described in section RB.1.3, based on the activity's defined sequencing information
	End For	
4.	**Exit** *Overall Rollup Process*	

pseudo code for overall rollup process

2.5. Invalid Default Value Defined for the measureSatisfactionIfActive attribute

This addendum address an error found in the SCORM Content Aggregation Model (CAM) Version 1.3.1. *Section 5.1.11: <rollupConsiderations> Element* contains an optional attribute named `measureSatisfactionIfActive`. This attribute is defined with an incorrect default value. The value defined in the CAM is `false`. The default value of the `measureSatisfactionIfActive` should be `true`. The `adlseq_v1p3.xsd` correctly identifies the default value as `true`.

2.5.1. Rationale for Change

This change is being made to correctly identify the default value of the `measureSatisfactionIfActive` attribute.

2.5.2. SCORM Update

The `measureSatisfactionIfActive` attribute defined for the `<rollupConsiderations>` element found in *Section 5.1.11: <rollupConsiderations> Element* will be updated to change the default value of the `measureSatisfactionIfValid` to `true`. The section will be updated as follows:

From:

> - `measureSatisfactionIfActive` (optional, default value = `false`). This attribute indicates if the measure should be used to determine satisfaction during rollup when the activity is active. XML Data Type: `xs:boolean`.

To:

> - `measureSatisfactionIfActive` (optional, default value = `true`). This attribute indicates if the measure should be used to determine satisfaction during rollup when the activity is active. XML Data Type: `xs:boolean`.

2.6.   Incorrect SPM for the <dataFromLMS> Element

This addendum addresses and error in the definition of the Smallest Permitted Maximum (SPM) for the ADL Content Packaging extension element `<adlcp:dataFromLMS>`. The SCORM 2004 CAM Version 1.3.1 incorrectly defines the SPM for the `<adlcp:dataFromLMS>` element's value as 4096 characters.

2.6.1.   Rationale for Change

The ADL Content Packaging Extension element, `<adlcp:dataFromLMS>`, is used to initialize the `cmi.launch_data` SCORM 2004 RTE Data Model element. The `cmi.launch_data` data model element has a defined SPM of 4000 characters. The IEEE Data Model Standard defines the SPM of the Launch Data element (i.e., maps to the `cmi.launch_data` model element) as 4000 characters. Since SCORM defines that the `<adlcp:dataFromLMS>` element is used to initialize the `cmi.launch_data` data model element, then the SPMs should match.

2.6.2.   SCORM Update

The Data Type section of the <dataFromLMS> element found in *Section 3.4.1.14: <dataFromLMS> Element* will be updated to change the SPM for the element's value from 4096 characters to 4000 characters. The section will be updated as follows:

From:

Data Type: The `<dataFromLMS>` element is represented as a characterstring element. The characterstring has an SPM of 4096 characters.

To:

Data Type: The `<dataFromLMS>` element is represented as a characterstring element. The characterstring has an SPM of 4000 characters.

2.7.   Handling of Reserved Delimiters

This addendum addresses various issues involving a discrepancy between the types of delimiters and the requirements on the syntax and placement of delimiters used by the SCORM 2004 RTE Data Model dot-notation binding in the SCORM 2004 RTE Version 1.3.1.

2.7.1.   Rationale For Change

This change is being made to remove the discrepancies between the types of delimiters, syntax requirements for each type and placement requirements for each type. The current text is confusing and does not clearly address these issues.

2.7.2.   SCORM Update

The following updates will be made to *Section 4.1.1.6: Reserved Delimiters*:

- Table 4.1.1.6a will be broken up to resolve the ambiguity between the two types of delimiters (Property and Separator delimiters).

- The section will be updated to describe the Property Delimiter Syntax Requirements, Property Delimiter Placement Syntax, Separator Delimiter Syntax Requirements and Separator Placement Syntax.

As mentioned above, Table 4.1.1.6a, will be broken up into two tables. Table 4.1.1.6a will be updated to describe the Reserved Property Delimiters as follows:

Table 4.1.1.6a: Reserved Property Delimiters

Reserved Delimiter Syntax	Default Value	Example
{lang=<language_type>}	{lang=en}	{lang=en}
{case_matters=<boolean>}	{case_matters=false}	{case_matters=true} {case_matters=false}
{order_matters=<boolean>}	{order_matters=true}	{order_matters=true} {order_matters=false}

Because these delimiters are not required, the default value shall be assumed for those cases where the delimiter is not specified. If the delimiters are used in the characterstring, there are other requirements on placement of the delimiter and the delimiter syntax.

Property Delimiter Syntax Requirements: The delimiter shall be treated as a constant set of characters with the following format::

```
delimiter ::= "{" + name + "=" + value + "}"
```

NOTE: The "{" and "}" are required to indicate the beginning and ending portions of a delimiter. The "=" is required to separate the `name` and `value` pieces of the delimiter. The absences of these required characters will cause the delimiter to not be recognized by the system; instead, the set of characters will be treated as part of the underlying characterstring data value.

The `name` represents the identifier of the delimiter. The `name` is represented by a set of reserved tokens:

- `lang`
- `case_matters`
- `order_matters`

For the {lang=<language_type>} delimiter to be recognized as the language for the characterstring, it must be placed at the beginning of the characterstring being qualified. If the {lang=<language_type>} delimiter is not the first set of characters, then the default language shall be assumed.

The `name` token must be represented as-is (i.e., one of the following: `lang`, `case_matters` or `order_matters`). Any derivatives of these tokens (e.g., padding the token names with white space) will result in an unrecognized delimiter and the set of characters will be treated as part of the underlying characterstring.

The `value` indicates the value for the named delimiter. The `value` portion of the delimiter is restricted to the following:

- `lang`: Restricted to the value represented by a `language_type` (Refer to *Section 4.1.1.7: Data Types* for requirements of a language_type).
- `case_matters`: Restricted to either `true` or `false`
- `order_matters`: Restricted to either `true` or `false`

NOTE: If the value does not meet its named delimiter's type requirements, then the delimiter is improperly formed and the characterstring does not meet the requirements of its type (i.e., causing a `406 – Data Type Mismatch` error to occur).

Valid Examples:

- SetValue("cmi.comments_from_learner.0.comment",
  "{lang=en}Characterstring in the English language")

- SetValue("cmi.interactions.0.correct_response.0.pattern",
  " {lang=en}{case_matters=true}Characterstring in the English
  language where the case matters")

- SetValue("cmi.comments_from_learner.0.comment",
  "{lang =fr}Characterstring in the English language")

There is no delimiter qualifying this `Characterstring` – "{lang =fr}" is not considered a language delimiter because it contains white space, which is not lexically equivalent to the `lang` reserved delimiter. In this case, the overall `Characterstring` includes {lang =fr} and is still considered valid; its default language is English ("en"). If this `SetValue` call is invoked, the LMS shall set the data model element associated with the call to
"{lang =fr}Characterstring in the English language", set the error code to `0 – No error` and return `true`.

- `SetValue("cmi.comments_from_learner.0.comment",`
  `"{case_matters=invalid}Characterstring in the English language")`

  There are no delimiters qualifying this `Characterstring` – "{case_matters=invalid" is not part of the defined format for `cmi.comments_from_learner.n.comment`. In this case the overall `Characterstring` includes {case_matters=invalid} and is still considered valid; its default language is English ("en"). If this `SetValue` call is invoked, the LMS shall set the data model element associated with the call to
  "{case_matters=invalid}Characterstring in the English language", set the error code to `0 – No error` and return `true`.

Invalid Examples:

- `SetValue("cmi.interaction.0.correct_response.0.pattern",`
  `"{case_matters=invalid}{lang=en}Characterstring in the English`
  `language")`

  Assuming that the type of `cmi.interaction.0` is `fill-in`, the case matters delimiter is invalid because it requires a value of true or false. This example uses "invalid." Because the delimiter is improperly formed, an LMS should not set the data model element associated with the call, set the error code to `406 – Data Type Mismatch`, and return `false`.

- `SetValue("cmi.comments_from_learner.0.comment",`
  `"{lang= fr}Characterstring in the French language")`

  In this example, the lang delimiter is invalid because its value includes white space, which is not part of a valid language string. Because the delimiter is improperly formed an LMS should not set the data model element associated with the call, set the error code to `406 – Data Type Mismatch`, and return `false`.

Property Delimiter Placement Requirements: The delimiters are required to be placed in specific positions within the characterstring. In those cases where a combination of delimiters may be used, the order of the delimiters is described by the data model element. If a default value is used (implied by the absence of a delimiter) for one of the delimiters in the set of delimiters, then the order should still be preserved. The delimiters shall be concatenated together with no white space permitted between the delimiters. For example:

- `{case_matters=true}{order_matters=true}`

No white space or other characters are permitted prior to the first delimiter identified in the characterstring. If there are no delimiters, which implies that the default values are being used, then the value represents the characterstring used for the data model element.

NOTE: If any white space or other character is found at the beginning of the characterstring, then this will cause the set of characters to be treated as part of the underlying characterstring.

As mentioned above, Table 4.1.1.6a will be broken up into two tables. Table 4.1.1.6b will be updated to describe the Reserved Separator Delimiters as follows:

Table 4.1.1.6b: Reserved Separator Delimiter

Reserved Delimiter Syntax	Default Value	Example

`[.]`	`Not applicable, needs to be provided`	`Used to separate a pair of values that are related for an interaction:`
		
		`1`**`[.]`**`a`

`[,]`	`Not applicable, needs to be provided`	`Used to separate a set of values for an interaction's collection:`
		
		`1[.]a`**`[,]`**`2[.]c`**`[,]`**`3[.]b`

`[:]`	`Not applicable, needs to be provided`	`Used to represent a separator between a range of numeric values:`
		
		`1[:]100 – a range where the numeric value is between 1 and 100 (inclusive)`

<u>Separator Delimiter Syntax Requirements:</u> The delimiter shall be treated as a constant set of characters with the following format::

        `delimiter ::= "[" + reserved_character + "]"`

NOTE: The "`[`" and "`]`" are required to indicate the beginning and ending portions of the delimiter. The absences of these required characters will cause the delimiter to not be recognized by the system; instead, the set of characters will be treated as part of the underlying characterstring.

The `reserved_character` represents the defined separator. The `reserved_character` is represented by a set of reserved tokens:

- `.`
- `,`

- :

NOTE: The `reserved_character` token must be represented as-is (i.e., one of the following: `[.]`, `[,]` or `[:]`). Any derivatives of these tokens (e.g., padding the reserved_character tokens with white space) will result in an unrecognized delimiter and the set of characters will be treated as part of the underlying characterstring.

Separator Delimiter Placement Requirements: The delimiters are required to be placed in specific positions within the characterstring. The delimiter is used to separate various data values defined by a particular data model element. For more information on the data model elements that use these separator delimiters refer to *Section 4.2: SCORM Run-Time Environment Data Model*.

2.8.   Deprecating the adlcp:persistState Attribute

This addendum addresses the deprecation of the `adlcp:persistState` attribute and its associated run-time behavior.

2.8.1.   Rationale For Change

This change is being made to resolve potential run-time behavior discrepancies related to the initialization, management, and persistence of Run-Time Data associated with a new learner attempt on a SCO whose associated learning resource has been tagged with an `adlcp:persistState` attribute equal to `true`.

2.8.2.   SCORM Update

It has been determined through a review of the use cases related to the `adlcp:persistState` attribute, that the intention of the `adlcp:persistState` attribute has been superseded by the development of various e-learning standards and specifications. The removal of the `adlcp:persistState` attribute will ensure that complimentary emerging technologies and specifications can incorporate a more comprehensive solution to the existing use cases without burdening LMSs and content developers with, potentially complex and confusing, legacy support.

The following updates will be made to the SCORM 2004 CAM Version 1.3.1:

- *Section 3.4.1.21: <resource> Element* will be updated to remove the adlcp:persistState attribute definition and declaration.

- Code Illustration 3-19 will be updated to remove the use of the adlcp:persistState attribute.

- Table 3.5.3.a will be updated to remove the adlcp:persistState attribute from the table.

The following updates will be made to the SCORM 2004 RTE Version 1.3.1:

- *Section 2.1.1.2: Persisting Run-Time Data Across Learner Attempts and Activities* will be removed.

- References to using Persist State will be removed from *Section 4.2.17: Objectives* and *Section 4.2.23: Suspend Data*.

The following updates will be made to the ADL Content Packaging Extension XML Schema Definition (XSD) file – adlcp_v1p3.xsd:

- Upon the formal release of an updated version of the SCORM 2004 documentation suite, the adlcp_v1p3.xsd will be updated to remove the attribute declaration of the adlcp:persistState.

2.9. Language Type Syntax

This addendum addresses unclear language in *Section 4.1.1.7: Data Types* in the SCORM 2004 RTE Version 1.3.1 on the requirements for the language_type data type.

2.9.1. Rationale For Change

The syntax and length restrictions for a `langcode` and `subcode` are ambiguous. The underlying standards (See Internet Engineering Task Force [IETF] Request For Comment [RFC] 3066 <http://www.ietf.org/rfc/rfc3066.txt>) indicate that the length of the `langcode` and each `subcode` range between a minimum of 1 to a maximum of 8 characters. This information is currently not provided in SCORM.

2.9.2. SCORM Update

Section 4.1.1.7: Data Types of the SCORM 2004 RTE Version 1.3.1 will be updated to add language stating that the `langcode` and `subcode` shall be 1 to 8 characters in length. Supplemental information will also be provided in relation to RFC 3066.

2.10.   Objective Satisfied By Measure Evaluation Behavior Discrepancy

This addendum addresses a behavioral allocation change regarding the threshold evaluation of a mastery score and the effects of that evaluation on SCORM 2004 RTE Data Model and Sequencing Tracking Data.

2.10.1.   Rationale For Change

The SCORM Run-Time Environment Version 1.3.1 defines an LMS behavior that applies an evaluation of `cmi.score.scaled` against a defined threshold, `cmi.scaled_passing_score`. The SCORM 2004 RTE Version 1.3.1, in *Table 4.2.22.1a*, describes this evaluation applied to various sets of SCO Data Model elements.

In addition, SCORM 2004 SN Version 1.3.1 defines a similar behavior that conditionally, based on the value of *Objective Satisfied by Measure*, applies a threshold evaluation to an activity's reported measure against a known minimum measure. The SCORM 2004 SN Version 1.3.1, *Section 4.2.1.7: Tracking Behavior*, requirement 5, defines this behavior.

In some situations, the threshold evaluation applied to a SCO's RTE data by an LMS may produce a different result than the threshold evaluation applied to the SCO's associated activity by a sequencing implementation. This discrepancy may be confusing to both LMS implementers and content developers.

2.10.2.   SCORM Update

To remove the potential discrepancy in success (satisfaction) status determination, the SCO Run-Time behavior will be updated to produce results consistent with the mastery evaluation applied to the SCO's associated activity. This change will require the following updates to SCORM:

SCORM 2004 CAM Version 1.3.1

Section 5.1.7.1.1: \<minNormalizedMeasure> Element

From:

> The `<minNormalizedMeasure>` element identifies minimum satisfaction measure for the objective [5]. The value is normalized between −1 and 1 (inclusive). If this element is used to define a minimum satisfaction measure for the primary objective (i.e., `<primaryObjective>` element), then the LMS shall use this value to initialize the `cmi.scaled_passing_score` (See the SCORM RTE Book[2]).

To:

The `<minNormalizedMeasure>` element identifies minimum satisfaction measure for the objective [5]. The value is normalized between −1 and 1 (inclusive). If the primary objective (i.e., `<primaryObjective>` element) has `satisfiedByMeasure` equal to `true`, the LMS shall use this value to initialize the `cmi.scaled_passing_score` (See the SCORM RTE Book[2]).

SCORM 2004 RTE Version 1.3.1

Section 4.2.19: Scaled Passing Score

From:

LMS Behavior Requirements:
- The data model element is mandatory and shall be implemented by the LMS as read-only.
- The LMS is responsible for initializing this data model element based using the IMS Simple Sequencing namespace element <imsss:minNormalizedMeasure> element associated with an <imsss:primaryObjective> element for the <imscp:item> element that references a SCO resource. If the value is not provided in the manifest, then the LMS shall not make any assumptions of a scaled passing score.

To:

LMS Behavior Requirements:
- The data model element is mandatory and shall be implemented by the LMS as read-only.
- The LMS is responsible for initializing this data model element.
- If the IMS Simple Sequencing namespace attribute imsss:satisfiedByMeasure associated with the <imsss:primaryObjective> element for the <imscp:item> element that references the SCO is equal to true, then the value provided by the <imsss:minNormalizedMeasure> element associated with the <imsss:primaryObjective> element for the <imscp:item> element that references the SCO resource shall be used to initialize this data model element.
- If the IMS Simple Sequencing namespace attribute imsss:satisfiedByMeasure associated with the <imsss:primaryObjective> element for the <imscp:item> element that references the SCO is equal to true and no value is provided for the <imsss:minNormalizedMeasure> element associated with the <imsss:primaryObjective> element for the <imscp:item> element that references the SCO resource, then the LMS shall initialize scaled_passing_score to 1.0.
- If the IMS Simple Sequencing namespace attribute imsss:satisfiedByMeasure associated with the <imsss:primaryObjective> element for the <imscp:item> element that references the SCO is equal to false, then the LMS shall not make any assumptions of a scaled passing score.

Table 4.2.22.1a, last row:

From:

Scaled Passing Score	Scaled Score	Success Status	LMS Behavior
0.8	No value set by the SCO.	One of the defined vocabularies	No action, the `cmi.success_status` shall be set to the value that was set by the SCO.

To:

Scaled Passing Score	Scaled Score	Success Status	LMS Behavior
0.8	No value set by the SCO.	One of the defined vocabularies	The cmi.success_status set by the SCO shall be overwritten, and replaced with the default value "unknown".

SCORM 2004 SN Version 1.3.1

Table 3.10a:

From:

No	*Name*	Description	Value Space	Default Value
3	*Objective Minimum Satisfied Normalized Measure*	Indicates the minimum satisfaction measure for the objective. If the objective's measure equals or exceeds this threshold, the *Objective Satisfied Status* will become Satisfied, otherwise the *Objective Satisfied Status* will become Not Satisfied.	Real [-1..1] Precision of at least 4 significant decimal digits	1.0

To:

No	*Name*	Description	Value Space	Default Value
3	*Objective Minimum Satisfied Normalized Measure*	Indicates the minimum satisfaction measure for the objective. If the objective's measure equals or exceeds this threshold, the *Objective Satisfied Status* will become Satisfied, otherwise the *Objective Satisfied Status* will become Not Satisfied. The value is unreliable unless *Objective Satisfied by Measure* is True.	Real [-1..1] Precision of at least 4 significant decimal digits	1.0

Section 3.10: Objective Description

The ADL Note found on page SN-3-39 will be removed from the SCORM 2004 SN Version 1.3.1. The behavior described in this note was based on an assumption made in an early draft of SCORM 2004 that is no longer applies due to changes introduced by this addenda.

2.11.   Manifest Requirements Inconsistency

This addendum addresses an inconsistency between element requirements in the SCORM 2004 CAM Version 1.3.1, Table 3.5.3a and *Section 3.4.1.2: <metadata> Element*.

2.11.1.   Rationale For Change

Table 3.5.3a incorrectly states that the 1.4.1 <schema> and 1.4.2 <schemaversion> elements are optional, marked with an O in the table. This information contradicts the information found in *Section 3.4.1.2: <metadata> Element*, which correctly states these elements are mandatory for a Content Aggregation Package and Resource Package. The language in the SCORM 2004 Conformance Requirements Version 1.1 also states that these elements are mandatory.

2.11.2.   SCORM Update

The language in Table 3.5.3a will be updated to state that the 1.4.1 <schema> and 1.4.2 <schemaversion> elements tags are mandatory.

No.	Elements	Resource Package	Content Aggregation Package
1	<manifest>	M	M
1.1	identifier	M	M
1.2	version	O	O
1.3	xml:base	O	O
1.4	<metadata>	O	O
1.4.1	<schema>	M	M
1.4.2	<schemaversion>	M	M
1.4.3	{Meta-data}	O	O
1.5	<organizations>	M	M
1.5.1	default	NP	M
1.5.2	<organization>	NP	M
1.5.2.1	identifier	NP	M
1.5.2.2	structure	NP	O
1.5.2.3	adlseq:objectivesGlobalToSystem	NP	O
1.5.2.4	<title>	NP	M
1.5.2.5	<item>	NP	M
1.5.2.5.1	identifier	NP	M
1.5.2.5.2	identifierref	NP	O
1.5.2.5.3	<title>	NP	M
1.5.2.5.4	isvisible	NP	O
1.5.2.5.5	parameters	NP	O
1.5.2.5.6	<item>	NP	O
1.5.2.5.7	<metadata>	NP	O
1.5.2.5.7.1	{Meta-data}	NP	O

1.5.2.5.8	\<adlcp:timeLimitAction\>	NP	O
1.5.2.5.9	\<adlcp:dataFromLMS\>	NP	O
1.5.2.5.10	\<adlcp:completionThreshold\>	NP	O
1.5.2.5.11	\<imsss:sequencing\>	NP	O
1.5.2.5.12	\<adlnav:presentation\>	NP	O
1.5.2.6	\<metadata\>	NP	O
1.5.2.6.1	{Meta-data}	NP	O
1.5.2.7	\<imsss:sequencing\>	NP	O
1.6	\<resources\>	M	M
1.6.1	xml:base	O	O
1.6.2	\<resource\>	O	O
1.6.2.1	identifier	M	M
1.6.2.2	type	M	M
1.6.2.3	href	O	O
1.6.2.4	adlcp:scormType	M	M
1.6.2.5	xml:base	O	O
1.6.2.6	\<metadata\>	O	O
1.6.2.6.1	{Meta-data}	O	O
1.6.2.7	\<file\>	O	O
1.6.2.7.1	href	M	M
1.6.2.7.2	\<metadata\>	O	O
1.6.2.7.2.1	{Meta-data}	O	O
1.6.2.8	\<dependency\>	O	O
1.6.2.8.1	identifierref	M	M
1.7	\<manifest\>	O	O
1.8	\<imsss:sequencingCollection\>	NP	O

2.12. Error in Pseudo-Code Limit Conditions Check Process

This addendum addresses an error in the SCORM 2004 SN Version 1.3.1 pseudo code, Line 2 of the Limit Conditions Check Process (UP.1).

2.12.1. Rationale For Change

The evaluation of limit conditions is to prevent unintended attempts on activities from beginning. Limit conditions do not have any affects on activities while they are active or suspended – as described in the comment to Line 2 of UP.1. The current condition evaluated on Line 2 of UP.1 does not fully enforce the criteria for evaluating limit conditions.

Line 2 of the Limit Conditions Check Process (UP.1) should read:

"If the Activity is Active for the activity is True Or the Activity is Suspended for the activity is True Then"

2.12.2. SCORM Update

The correct language will be added to Line 2 of the Limit Conditions Check Process (UP.1).

The pseudo code will be updated to:

Limit Conditions Check Process [UP.1] (for an activity; returns *True* if any of the activity's limit conditions have been violated):		
Reference: Activity Attempt Count TM.1.2.1; Activity Progress Status TM.1.2.1; Activity Absolute Duration TM.1.2.1; Activity Experienced Duration TM.1.2.1; Attempt Progress Status TM.1.2.2; Attempt Absolute Duration TM.1.2.2; Attempt Experienced Duration TM.1.2.2; Limit Condition Activity Absolute Duration Control SM.3; Limit Condition Activity Absolute Duration Limit SM.3; Limit Condition Activity Experienced Duration Control SM.3; Limit Condition Activity Experienced Duration Limit SM.3; Limit Condition Attempt Absolute Duration Control SM.3; Limit Condition Attempt Absolute Duration Limit SM.3; Limit Condition Attempt Experienced Duration Control SM.3; Limit Condition Attempt Experienced Duration Limit SM.3; Limit Condition Attempt Control SM.3; Limit Condition Attempt Limit SM.3; Limit Condition Begin Time Limit SM.3; Limit Condition Begin Time Limit Control SM.3; Limit Condition End Time Limit SM.3; Limit Condition End Time Limit Control SM.3; Tracked SM.11		
1.	**If** *Tracked* for the activity is *False* **Then**	If the activity is not tracked, its limit conditions cannot be violated
1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *False*)	Activity is not tracked, no limit conditions can be violated
	End If	

2.	**If** the *Activity is Active* for the activity is *True* **Or** the *Activity is Suspended* for the activity is *True* **Then**	Only need to check activities that will begin a new attempt
2.1	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *False*)	
	End If	
3.	**If** the *Limit Condition Attempt Control* for the activity is *True* **Then**	
3.1.	**If** the *Activity Progress Status* for the activity is *True* **And** the *Activity Attempt Count* for the activity is greater than or equal *(>=)* to the *Limit Condition Attempt Limit* for the activity **Then**	
3.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	
4.	**If** the *Limit Condition Activity Absolute Duration Control* for the activity is *True* **Then**	
4.1.	**If** the *Activity Progress Status* for the activity is *True* **And** the *Activity Absolute Duration* for the activity is greater than or equal *(>=)* to *Limit Condition Activity Absolute Duration Limit* for the activity **Then**	
4.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	
5.	**If** the *Limit Condition Activity Experienced Duration Control* for the activity is *True* **Then**	
5.1.	**If** the *Activity Progress Status* for the activity is *True* **And** the *Activity Experienced Duration* for the activity is greater than or equal *(>=)* to the *Limit Condition Activity Experienced Duration Limit* for the activity **Then**	
5.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	
6.	**If** the *Limit Condition Attempt Absolute Duration Control* for the activity is *True* **Then**	
6.1.	**If** the *Activity Progress Status* for the activity is *True* **And** the *Attempt Progress Status* for the activity is *True* **And** the *Attempt Absolute Duration* for the activity is greater than or equal *(>=)* to the *Limit Condition Attempt Absolute Duration Limit* for the activity **Then**	
6.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	
7.	**If** the *Limit Condition Attempt Experienced Duration Control* for the activity is *True* **Then**	
7.1.	**If** the *Activity Progress Status* for the activity is *True* **And** the *Attempt Progress Status* for the activity is *True* **And** the *Attempt Experienced Duration* for the activity is greater than or equal *(>=)* to the *Limit Condition Attempt Experienced Duration Limit* for the activity **Then**	
7.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	

8.	**If** the *Limit Condition Begin Time Limit Control* for the activity is *True* **Then**	
8.1.	**If** the current time point is before the *Limit Condition Begin Time Limit* for the activity **Then**	
8.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	
9.	**If** the *Limit Condition End Time Limit Control* for the activity is *True* **Then**	
9.1.	**If** the current time point is after the *Limit Condition End Time Limit* for the activity **Then**	
9.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated
	End If	
	End If	
10..	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *False*)	No limit conditions have been violated

pseudo code for limit conditions check process

2.13. Error in Pseudo-Code Choice Flow Tree Traversal Sub-process

This addendum addresses an error in the SCORM 2004 SN Version 1.3.1 pseudo code, Line 2.1.1 of the Choice Flow Tree Traversal Sub-process (SB.2.9.2).

2.13.1. Rationale For Change

The Choice Flow Tree Traversal Sub-process (SB.2.9.2), Line 2.1.1 is missing some text. (Next Activity) should be changed to (Next Activity: *Nil*).

2.13.2. SCORM Update

(Next Activity) will be changed to (Next Activity: *Nil*) as follows:

Choice Flow Tree Traversal Subprocess [SB.2.9.2] (for an activity, a traversal direction; returns the 'next' activity in directed traversal of the activity tree):		
Reference: Available Children AM.1.1		
1.	**If** the traversal direction is *Forward* **Then**	
1.1.	**If** the activity is the last activity in a forward preorder tree traversal of the activity tree **Then**	Cannot walk off the activity tree
1.1.1.	**Exit** *Choice Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	
	End If	
1.2.	**If** the activity is the last activity in the activity's parent's list of *Available Children* **Then**	
1.2.1.	Apply the *Choice Flow Tree Traversal Subprocess* to the activity's parent in the *Forward* direction	Recursion - Move to the activity's parent's next forward sibling
1.2.2.	**Exit** *Choice Flow Tree Traversal Subprocess* (**Next Activity**: the results of the recursive *Choice Flow Tree Traversal Subprocess*)	Return the result of the recursion
1.3.	**Else**	
1.3.1.	Traverse the tree, forward preorder, one activity to the next activity, in the activity's parent's list of *Available Children*	
1.3.2.	**Exit** *Choice Flow Tree Traversal Subprocess* (**Next Activity**: *the activity identified by the traversal*)	
	End If	
	End If	
2.	**If** the traversal direction is *Backward* **Then**	
2.1.	**If** the activity is the root activity of the tree **Then**	Cannot walk off the root of the activity tree
2.1.1.	**Exit** *Choice Flow Tree Traversal Subprocess* (**Next Activity: *Nil***)	
	End If	
2.2.	**If** the activity is the first activity in the activity's parent's list of *Available Children* **Then**	
2.2.1.	Apply the *Choice Flow Tree Traversal Subprocess* to the	Recursion – Move

	activity's parent in the *Backward* direction	to the activity's parent's next backward sibling
2.2.1.1.	**Exit** *Choice Flow Tree Traversal Subprocess* (**Next Activity**: the results of the recursive *Choice Flow Tree Traversal Subprocess*)	Return the result of the recursion
2.3.	**Else**	
2.3.1.	Traverse the tree, reverse preorder, one activity to the previous activity, from the activity's parent's list of *Available Children*	
2.3.2.	**Exit** *Choice Flow Tree Traversal Subprocess* (**Next Activity**: *the activity identified by the traversal*)	
	End If	
	End If	

pseudo code for choice flow tree traversal subprocess

2.14.   Error in Pseudo-Code Previous Sequencing Request Process

This addendum addresses an error in the SCORM 2004 SN Version 1.3.1 pseudo code, Line 2.1 of the Previous Sequencing Request Process (SB.2.8).

2.14.1.   Rationale For Change

The Previous Sequencing Request Process (SB.2.8), Line 2.1 is missing the conditional value of "False".

2.14.2.   SCORM Update

The Previous Sequencing Request Process will be updated as follows:

Previous Sequencing Request Process [SB.2.8] (may return a delivery request; may return an exception code):		
Reference: Current Activity AM.1.2; Flow Subprocess SB.2.3		
1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun
1.1.	**Exit** *Previous Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.8-1*)	Nothing to deliver
	End If	
2.	**If** the activity is not the root activity of the activity tree **Then**	
2.1.	**If** *Sequencing Control Flow* for the parent of the activity is *False* **Then**	Confirm a 'flow' traversal is allowed from the activity
2.1.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*; **Exception**: *SB.2.8-2*)	
	End If	
	End If	
3.	Apply the *Flow Subprocess* to the *Current Activity* in the *Backward* direction with consider children equal to *False*	Flow in a backward direction to the next allowed activity
4.	**If** the *Flow Subprocess* returns *False* **Then**	
4.1.	**Exit** *Previous Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Flow Subprocess*)	Nothing to deliver
5.	**Else**	
5.1.	**Exit** *Previous Sequencing Request Process* (**Delivery Request**: the activity identified by the *Flow Subprocess*; **Exception**: *n/a*)	
	End If	

pseudo code for previous sequencing request process

2.15. Application of Measure Satisfaction if Active Behavior

This addendum addresses inconsistency in the application of the ADL extension attribute `measureSatisfactionIfActive`, as defined in the SCORM 2004 SN Version 1.3.1.

2.15.1. Rationale For Change

Two behavioral extensions (fixes) to IMS SS were put into the SCORM SN with regard to measure rollup. First, a cluster will always have a measure if **any** of its children have a defined measure. Second, since the rolled-up measure may not include sufficient information (some child activities may have not been attempted), a measure threshold evaluation (Satisfied by Measure) may prematurely affect the satisfaction status of the activity. To prevent a measure threshold evaluation from prematurely affecting an activity's satisfaction status, an ADL namespace element, `measureSatisfactionIfActive` was added. This attribute indicates when the measure threshold evaluation should affect an activity's satisfaction status – either during the current attempt on the activity or not until the current attempt on the activity ends (the activity becomes "inactive"). Enforcement of these behaviors was put directly into the normative Sequencing Behavior Pseudo Code (Appendix C).

Language was added to the SCORM SN (*Section 4.2.1.7: Tracking Behavior*, page SN-4-18) to ensure consistent evaluation of a shared global objective status. More specifically, clause #5 states that threshold evaluations will be performed any time the local status of activity is unknown and a **read** *Objective Map* is defined. As written, this behavior will circumvent the pseudo code's application of the ADL namespace element, `measureSatisfactionIfActive`.

2.15.2. SCORM Update

Section 3.9.1: Measure Satisfaction If Active (SN-3-35) will be updated as follows:

From:

> The *Measure Satisfaction If Active* element indicates when an activity's rolled-up objective measure will be applied to the rolled-up objective satisfaction. This element is applied during the Objective Rollup Using Measure Process (refer to *Appendix C*). This element contains a boolean (True/False) value. The default value for *Measure Satisfaction If Active*, if not defined explicitly for the activity, is True.

> If the *Measure Satisfaction If Active* element is defined as False, the LMS will only apply the *Objective Minimum Satisfied Normalized Measure* to the activity's rolled-up objective when the attempt on the activity ends.

To:

> Evaluation of a defined measure threshold (*Objective Satisfied by Measure*) is conditional. The *Measure Satisfaction If Active* element indicates when an activity's rolled-up objective measure will be applied to the activity's rolled-up objective satisfaction. This element is applied whenever the activity's satisfaction status is required and that status is determined through a measure threshold evaluation (*Objective Satisfied by Measure*). This element contains a boolean (True/False) value. The default value for *Measure Satisfaction If Active*, if not defined explicitly for the activity, is True.
>
> If the *Measure Satisfaction If Active* element is defined as False, the LMS will only apply the *Objective Minimum Satisfied Normalized Measure* to the activity's rolled-up objective when the attempt on the activity ends and the activity becomes inactive. Until that time the satisfaction status of the activity's rolled-up objective shall be considered "unknown".

And on page SN-3-36:

From:

> In contrast, if the *Measure Satisfaction If Active* element is False, the satisfaction of Module AA's objective will not be evaluated until Module AA is explicitly exited – it will remain "unknown". Module AA can only be explicitly exited by a *Sequencing Exit* action rule applied to Module A evaluating to true.

To:

> In contrast, if the *Measure Satisfaction If Active* element is False, the satisfaction of Module AA's rolled-up (primary) objective will not be evaluated until Module AA becomes inactive – it will remain "unknown". One way for Module AA to become inactive is for it to be explicitly exited by a *Sequencing Exit* action rule applied to Module AA evaluating to true.

In *Section 4.2.1.7: Tracking Behavior*, clause #5 (SN-4-18) will be updated as follows:

From:

> **ADL Note**: In this case, an LMS may store the evaluated Objective Satisfied Status in the activity's local Objective Progress Information. Doing so should not alter the local Objective Measure.

SCORM 2004 2nd Edition Addendum Version 1.2

To:

> **ADL Note**: Evaluation of a defined measure threshold (Objective Satisfied by Measure) is conditional on the state of the activity and the value of its Measure Satisfaction If Active element. If the activity is active and has Measure Satisfaction If Active is equal to false, no measure threshold evaluation shall be performed – the satisfaction status of the activity shall be considered "unknown".
>
> When this evaluation is performed, an LMS may store the evaluated Objective Satisfied Status in the activity's local Objective Progress Information. Doing so should not alter the local Objective Measure.

2.16. Initialization of a SCO's Objectives from Sequencing Information

This addendum addresses the incomplete definition of required LMS behavior concerning initialization and updates to a SCO's objectives collection (`cmi.objectives`) using sequencing information applied to the SCO's associated activity, as defined in the SCORM 2004 RTE Version 1.3.1.

2.16.1. Rationale For Change

Each SCO's Run-Time Data includes a collection of objectives – `cmi.objectives`. Each activity may have zero or more objectives defined in its sequencing information. The SCORM 2004 RTE Version 1.3.1 intends for SCOs to have some access to the state of its associated activity's objectives based on sequencing Tracking Information available at the time the SCO is launched. Although the SCORM 2004 RTE Version 1.3.1 describes this behavior, the information is spread across several sections of the document and that information does not clearly describe what should happen when a Learner Attempt on a SCO suspends and later resumes.

2.16.2. SCORM Update

The description of initializing Run-Time Data Model objectives, *Section 4.2.17.2: Utilizing Objective Status for Sequencing* (RTE-4-96), will be updated as follows:

From:

> **Utilizing Objective Status for Sequencing**
>
> Instructional designers may wish to use objectives to make conditional sequencing decisions; this desire is explicitly represented in the sequencing information associated with a learning activity (refer to the SCORM CAM book). As a SCO references objectives through their identifiers, so does the sequencing information associated with a learning activity. SCORM defines how identifiers are used to relate the objectives defined on the learning activity for sequencing purposes with the objectives (`cmi.objectives`) available to a SCO during the learning experience.
>
> For objectives associated with a learning activity, those that may be affected by the learning experience are those with identifiers that match identically with an identifier defined in the Objectives RTE Data Model element for the SCO associated with that learning activity. When a SCO is launched for a new learner attempt, the LMS will initialize a set of run-time objectives (`cmi.objectives.n.xxx`) for the SCO with the objective status information

managed for sequencing the SCO's associated learning activity. During the learner experience, the SCO may modify the status of these sets of objective status information; the updated status information will be used by the LMS during sequencing evaluations.

To:

Initialization of Run-Time Objectives from Sequencing Information

Instructional designers may wish to use objectives to make conditional sequencing decisions; this desire is explicitly represented in the sequencing information associated with a learning activity (refer to the SCORM CAM book). As a SCO references objectives through their identifiers, so does the sequencing information associated with a learning activity. For objectives associated with a learning activity, those that may be affected by the learning experience are those with identifiers that match identically with an identifier defined in the `cmi.objectives` collection for the SCO associated with that learning activity.

SCORM defines how identifiers are used to relate the objectives defined on the learning activity for sequencing purposes to the objectives collection (`cmi.objectives`) available to the activity's associated SCO during the learning experience. Whenever a SCO is launched for a new learner attempt, the LMS shall initialize a set of run-time objectives (`cmi.objectives`) for the SCO with the objective status information managed for sequencing the SCO's associated learning activity. The LMS shall create one entry in the SCO's objectives collection for each objective defined in the sequencing information (children of the `<imsss:objectives>` element) applied to the SCO's associated activity. The sequencing information shall be used to initialize the run-time objective as follows:

1. cmi.objectives.n.id shall be initialized with the objective's ID (the objective ID attribute associated to the objective in the activity's sequencing information.)

2. cmi.objectives.n.success_status shall be initialized with the Tracking Information associated with the objective (possibly through a **read** Objective Map).

3. cmi.objectives.n.score.scaled shall be initialized with the Tracking Information associated with the objective (possibly through a **read** Objective Map).

When the new learner attempt on the SCO begins and after LMS initialization of the SCO' objective collection, the number of objectives contained in its objective collection (`cmi.objectives._count`) shall equal the number of objectives defined in the sequencing information (children of the `<imsss:objectives>` element) applied to the SCO's associated activity that have defined Objective IDs. During the learner experience, the SCO may modify the status of these objectives;

the updated status information will be used by the LMS during sequencing evaluations.

ADL Note: If a SCO alters an objective ID (`cmi.objectives.id`), the association between the run-time objective and the associated activity's objective will be broken. The LMS will not know how to map the objective's run-time status to the sequencing Tracking Information.

ADL Note: If a learner attempt on a SCO is suspended and then later resumed, the LMS is not required to update the SCO's objectives collection to reflect changes in sequencing Tracking Information occurring with any of the SCO's associated activity's objectives while the SCO is suspended.

The LMS Behavior Requirements defined for `cmi.objectives.n.id`, (RTE-4-98), will be updated as follows:

From:

> If the <imsss:objectives> are defined for an <imscp:item> element in the Content Package Manifest, then the LMS is responsible for initializing objective run-time data (*cmi.objectives.n.xxx*) for the SCO based on the *Objective Progress Information* referenced and managed by the learning activity. Run-time data related to objectives (cmi.objectives.n.xxx) should not be initialized for an activity's associated SCO unless an objective ID attribute is defined in the sequencing information (<imsss:primaryObjective> or <imsss:objective>). The objective ID attribute shall be used to initialize the *cmi.objectives.n.id* value. The number of objectives defined in the manifest dictates the number of objective status information that need to be initialized. The LMS is also responsible for initializing status and score for the objective information data if that information is available to the LMS (refer to the SCORM SN book – Global objectives).

To:

> If the <imsss:objectives> are defined for an <imscp:item> element in the Content Package Manifest, then the LMS is responsible for initializing objective run-time data (*cmi.objectives.n.xxx*) for the SCO based on the *Objective Progress Information* referenced and managed for the learning activity. Run-time data related to objectives (cmi.objectives.n.xxx) should not be initialized for an activity's associated SCO unless an objective ID attribute is defined in the sequencing information (<imsss:primaryObjective> or <imsss:objective>). The objective ID attribute shall be used to initialize the *cmi.objectives.n.id* value.

2.17. Variable Error in Measure Rollup Process (RB.1.1)

This addendum addresses an error in the variables in the Measure Rollup Process (RB.1.1), as defined in the SCORM 2004 SN Version 1.3.1.

2.17.1. Rationale For Change

Lines 5.2 and 5.3 of the Measure Rollup Process (RB.1.1) refer to the wrong variable.

The intention of the Measure Rollup Process is to compute the average weighted measure of all the measures applied to an activity's children. This process should only provide a rolled-up measure if there is AT LEAST ONE child that contributes a known measure to its parent; otherwise there is insufficient information available to compute any value and the rolled-up (parent's) measure should be considered "unknown." This intended behavior is described in the comment on line 5.2 from the IMS SS Specification.

The wrong variable ("counted measures" instead of "total weighted measure") is used on lines 5.2 and 5.3. These lines were correct in the IMS SS Version 1.0 of the p-code because the IMS measure rollup process from that version would only result in a known rolled-up measure if ALL children contributed a known measure to its parent. If any child did not have a known measure, "counted measures" would b set to 0.0 – in this case it was sufficient to look at just the "counted measures" to evaluate clauses 5.2 and 5.3.

However, SCORM Sequencing altered the Measure Rollup Process to ensure that if ANY child had a known measure, then its parent would have a known measure. This extension resulted in a change to the IMS SS pseudo code and makes the clauses on lines 5.2 and 5.3 insufficient. The result of using the "counted measures" in these two lines is that the "number of children" (counted measures) is used as the criteria for determining if a rolled-up measure should be evaluated instead of the existence of a known measure on one of the children. In other words, based on the current SCORM Sequencing pseudo code, if a cluster has ANY child that is tracked and has a non-zero measure weight, then the "countered measures" will be non-zero (ignoring clause 5.2) and the cluster will have known measure of 0.0 (from line 5.3.2).

To illustrate the problem with an example: assume that Activity A has 3 children, A1, A2, and A3 that each have the default measure weight (1.0) and no defined measure (measure is "unknown") for their primary (rolled-up) objective. If the Measure Rollup Process is applied to A, the process will calculate 3 for "counted measure" (on line 5.1.1.3.1) but will not increment "total weighted measure". Line 5.3 would cause the activity to have a rolled-up measure of 0.0 (0.0 / 3.0).

2.17.2. SCORM Update

RB.1.1 will be updated to add a new line 2 (shift all other lines down 1):

> Set valid data to *False*

RB.1.1 will be updated to add a new line 5.1.1.3.2.2 (line number prior to renumbering):

> Set valid data to *True*

RB.1.1 line 5.2 will be updated to read:

> **If** valid data is *False* **Then**

In RB.1.1, the current line 5.2.2 will be deleted and the line **End If** that immediately follows 5.2.2 will be removed. The current line 5.3 will be updated to read:

> **Else**

RB.1.1 will be updated by swapping lines 5.3.3 and the **End If** line that immediately follows it.

From:

Measure Rollup Process [RB.1.1] (for an activity; may change the *Objective Information* for the activity):		
Reference: Objective Contributes to Rollup SM.6; Objective Description SM.6; Objective Measure Status TM.1.1; Objective Normalized Measure TM.1.1; Rollup Objective Measure Weight SM.8; Tracked SM.11		
1.	Set the total weighted measure to *Zero (0.0)*	
2.	Set the counted measures to *Zero (0.0)*	
3.	Set the target objective to *Undefined*	
4.	**For** each objective associated with the activity	
4.1.	**If** *Objective Contributes to Rollup* for the objective is *True* **Then**	Find the target objective for the rolled-up measure
4.1.1.	Set the target objective to the objective	
4.1.2.	**Break For**	
	End If	
	End For	
5.	**If** target objective is *Defined* **Then**	
5.1.	**For** each child of the activity	
5.1.1.	**If** *Tracked* for the child is *True* **Then**	Only include tracked children
5.1.1.1.	Set rolled-up objective to *Undefined*	
5.1.1.2.	**For** each objective associated with the child	
5.1.1.2.1.	**If** *Objective Contributes to Rollup* for the objective is *True* **Then**	
5.1.1.2.1.1.	Set rolled-up objective to the objective	
5.1.1.2.1.2.	**Break For**	
	End If	
	End For	
5.1.1.3.	**If** rolled-up objective is *Defined* **Then**	
5.1.1.3.1.	Increment counted measures by the *Rollup Objective Measure Weight* for the child	
5.1.1.3.2.	**If** the *Objective Measure Status* for the rolled-up	

	objective is *True* **Then**	
5.1.1.3.2.1.	Add the product of *Objective Normalized Measure* for the rolled-up objective multiplied by the *Rollup Objective Measure Weight* for the child to the total weighted measure	
	End If	
5.1.1.4.	**Else**	
5.1.1.4.1.	**Exit** *Measure Rollup Process*	One of the children does not include a rolled-up objective
	End If	
	End If	
	End For	
5.2.	**If** counted measures is *Zero (0.0)* **Then**	
5.2.1.	Set the *Objective Measure Status* for the target objective to *False*	No tracking state rolled-up, cannot determine the rolled-up measure
5.2.2.	**Exit** *Measure Rollup Process*	
	End If	
5.3.	**If** counted measures is greater than *(>) Zero (0.0)* **Then**	Set the rolled-up measure for the target objective
5.3.1.	Set the *Objective Measure Status* for the target objective to *True*	
5.3.2.	Set the *Objective Normalized Measure* for the target objective to the total weighted measure divided by counted measures	
5.3.3.	**Exit** *Measure Rollup Process*	
	End If	
	End If	
6.	**Exit** *Measure Rollup Process*	No objective contributes to rollup, so we cannot set anything

pseudo code for measure rollup process

To:

Measure Rollup Process [RB.1.1] (for an activity; may change the *Objective Information* for the activity):		
Reference: Objective Contributes to Rollup SM.6; Objective Description SM.6; Objective Measure Status TM.1.1; Objective Normalized Measure TM.1.1; Rollup Objective Measure Weight SM.8; Tracked SM.11		
1.	Set the total weighted measure to *Zero (0.0)*	
2.	Set valid data to *False*	
3.	Set the counted measures to *Zero (0.0)*	
4.	Set the target objective to *Undefined*	
5.	**For** each objective associated with the activity	
5.1.	**If** *Objective Contributes to Rollup* for the objective is *True* **Then**	Find the target objective for the rolled-up measure
5.1.1.	Set the target objective to the objective	
5.1.2.	**Break For**	
	End If	
	End For	
6.	**If** target objective is *Defined* **Then**	

6.1.	**For** each child of the activity	
6.1.1.	**If** *Tracked* for the child is *True* **Then**	Only include tracked children
6.1.1.1.	Set rolled-up objective to *Undefined*	
6.1.1.2.	**For** each objective associated with the child	
6.1.1.2.1.	**If** *Objective Contributes to Rollup* for the objective is *True* **Then**	
6.1.1.2.1.1.	Set rolled-up objective to the objective	
6.1.1.2.1.2.	**Break For**	
	End If	
	End For	
6.1.1.3.	**If** rolled-up objective is *Defined* **Then**	
6.1.1.3.1.	Increment counted measures by the *Rollup Objective Measure Weight* for the child	
6.1.1.3.2.	**If** the *Objective Measure Status* for the rolled-up objective is *True* **Then**	
6.1.1.3.2.1.	Add the product of *Objective Normalized Measure* for the rolled-up objective multiplied by the *Rollup Objective Measure Weight* for the child to the total weighted measure	
6.1.1.3.2.2	Set valid data to *True*	
	End If	
6.1.1.4.	**Else**	
6.1.1.4.1.	**Exit** *Measure Rollup Process*	One of the children does not include a rolled-up objective
	End If	
	End If	
	End For	
6.2.	**If** valid data is *False* **Then**	
6.2.1.	Set the *Objective Measure Status* for the target objective to *False*	No tracking state rolled-up, cannot determine the rolled-up measure
6.3.	**If** counted measures is greater than *(>) Zero (0.0)* **Then**	Set the rolled-up measure for the target objective
6.3.1.	Set the *Objective Measure Status* for the target objective to *True*	
6.3.2.	Set the *Objective Normalized Measure* for the target objective to the total weighted measure divided by counted measures	
	End If	
6.3.3.	**Exit** *Measure Rollup Process*	
	End If	
7.	**Exit** *Measure Rollup Process*	No objective contributes to rollup, so we cannot set anything

pseudo code for measure rollup process

2.18.   Incorrect Definition of Extended Rollup Set

This addendum addresses an incorrect definition of the "rollup set" in the Extended Rollup Process, as defined in the SCORM 2004 SN Version 1.3.1.

2.18.1.   Rationale For Change

The Extended Rollup Process (described in the SCORM 2004 SN Version 1.3.1 on page SN-4-34, Rule A) incorrectly defines the "rollup set" – too many activities are included. Adhering strictly with the language of the SCORM 2004 SN Version 1.3.1, LMS sequencing implementers would fail at least one of the test cases defined in the SCORM 2004 Conformance Requirements Version 1.2.

The purpose of the Extended Rollup Process is to propagate status information, made available through shared global objectives, throughout the entire Activity Tree. The Overall Rollup Process (RB.1.5) does not need to be, and should not be, applied directly to an activity that shares global state- (through a "read" map) with some other activity (through a "write" map). In these cases, the current and most valid state has already communicated directly through the shared global objective to the activity that shares state. Applying the Overall Rollup Process to this activity would, in most cases, overwrite that shared state and defeat intent of the Extended Rollup Process.

2.18.2.   SCORM Update

In the SCORM 2004 SN Version 1.3.1 (page SN-4-34) Rule A will be updated as follows:

From:

> Determine all activities that are affected by the change of status. This includes the *Current Activity* and any activity (**read** *Objective Map)* that shares a global objective with the *Current Activity* (**write** *Objective Map*) – this is the "rollup set".

To:

> Determine all activities that are affected by the change of status. This includes the *Current Activity* and the parent of any activity (**read** *Objective Map)* that shares a global objective with the *Current Activity* (**write** *Objective Map*) – this is the "rollup set".

2.19.   Portability (Interoperability) of URLs between Windows and UNIX Systems

This addendum addresses an issue that affects portability (interoperability) of Universal Resource Locators (URLs) between Windows and UNIX systems, specifically related to the use of the backward slash ("\") character.

2.19.1.   Rationale For Change

RFC 2396 allows backward slashes in URLs, but specifies that it is unwise to use them. The RFC does not specify the semantics of the character; however, in URLs for content that may be put on any Web server regardless of implementation technology, and also particular for `xml:base`, the character has specific semantics that affect interoperability, and only the interoperable form should be allowed. Currently, SCORM does not define any requirements and guidance in using the backward slash ("\") or slash ("/") in URLs.

2.19.2.   SCORM Update

The SCORM 2004 CAM Version 1.3.1 will be updated to describe how URLs should be represented. A new section, Section 3.4.4.4, will be added to the CAM to specify that if a backward slash ("\") is needed in a URL (`href` or `xml:base` attributes) then the backward slash shall be properly escaped. The section will also define that for those URLs that are hierarchical then the slash ("/") character shall be used to separate the hierarchical components (e.g., used to construct a file name hierarchy).

For example: `<resource href="lesson/module/topic1/sco1.htm">`

2.20. The \<adlnav:presentation\> Element Should be Permitted on an Asset Resource

This addendum addresses the need to permit the `<adlnav:presentation>` element on an Asset Resource.

2.20.1. Rationale For Change

SCORM 2004 currently prohibits the `<adlnav:presentation>` element from being a direct-descendant of an `<item>` element that references an Asset Resource. The `<adlnav:presentation>` element is only allowed on an `<item>` element that references a SCO Resource. In some cases, content developers may want to force a learner to choose an activity from a menu or table of contents while the learner is experiencing an Asset Resource. If this learning strategy is required, then the content designer may want to inform the LMS to hide any user interface controls that might indicate to the learner to flow (i.e., "continue" or "previous" buttons). This strategy will force the learner to choose an item from a menu or table of contents.

If the content developer wants to indicate to the LMS to hide the user interface controls provided by the LMS, then this information must be defined on the `<item>` element. To do this, the content developer must place the `<adlnav:presentation>` (and the appropriate descendants of this element) as a direct-descendant of the `<item>` in question.

The change is being made to permit the learning strategy described above. This change will permit the use of the `<adlnav:presentation>` element as a direct-descendant of any `<item>` element that references a resource.

2.20.2. SCORM Update

Section 5.2.1.1: \<presentation\> Element will be updated as follows:

From:

> The `<presentation>` element is a container element that encapsulates presentation information for a given learning activity. This element is an ADL-defined extension to the IMS Content Packaging Specification. The element shall only appear, if needed, as a child of a leaf `<item>` element that references a SCO. Only those `<item>` elements that reference a SCO resource can contain the `<presentation>` element.

To:

The `<presentation>` element is a container element that encapsulates presentation information for a given learning activity. This element is an ADL-defined extension to the IMS Content Packaging Specification. The element shall only appear, if needed, as a child of a leaf `<item>` element that references a resource.

2.21.   Default Value for timeinterval(second,10,2) Should be any Value that Evaluates to Zero

This addendum addresses an issue with the default value for the SCORM 2004 RTE Data Model element `cmi.total_time`.

2.21.1.   Rationale For Change

The SCORM 2004 RTE Version 1.3.1 defines the default value for the cmi.total_time data model element as PT0H0M0S. This default value evaluates to a total time of zero. This default value is too restrictive and will be updated to permit any value that evaluates to a total time of zero. This will permit an LMS to initialize this value to zero and not require the value to be represented in particular format. For example PT0M (i.e., a period of time of zero minutes) is semantically equivalent to PT0H0M0S (i.e., a period of time of zero hours, zero minutes and zero seconds).

How this value is syntactically represented is not important. The important fact is that the value evaluates to an interval of time of zero.

2.21.2.   SCORM Update

Table 4.2.25a: Dot-notation Binding for the Total Time Data Model Element will be updated:

From:

LMS Behavior Requirements:
- The data model element is mandatory and shall be implemented by the LMS as read-only.
- Since this data model element is implemented by the LMS as read-only, it is the responsibility of the LMS to manage this data. Since this value is the accumulated session times (*cmi.session_time*), the LMS cannot determine this value until the SCO sets session times. If the SCO requests the value before any session times have been set, then the LMS shall behave according to the API Implementation Requirements behaviors defined below.
- The value of the *cmi.total_time* shall not be updated by the LMS while a learner session is in progress.
- The default value for the *cmi.total_time* shall be PT0H0M0S.

To:

LMS Behavior Requirements:
- The data model element is mandatory and shall be implemented by the LMS as read-only.
- Since this data model element is implemented by the LMS as read-only, it is the responsibility of the LMS to manage this data. Since this value is the accumulated session times (*cmi.session_time*), the LMS cannot determine this value until the SCO sets session times. If the SCO requests the value before any session times have been set, then the LMS shall behave according to the API Implementation Requirements behaviors defined below.
- The value of the *cmi.total_time* shall not be updated by the LMS while a learner session is in progress.
- The default value for the *cmi.total_time* shall be any characterstring that evaluates to a duration of 0. The characterstring format shall adhere to the requirements defined for the timeinterval (second,10,2) data type defined in Section 4.1.1.7 *Data Types* (e.g., PT0S is equivalent to PT0H because they both evaluate to a duration of 0).

2.22. Exit Action Rule on the Activity Tree Root does not End the Sequencing Session

In the SCORM 2004 SN Version 1.3.1, if a Sequencing Exit Action Rule successfully evaluates on the root of the Activity Tree and either a Sequencing Post Condition Rule or a (user initiated) sequencing request of Continue or Previous is pending, that request would be incorrectly processed and result in an uncaught exception in the sequencing pseudo code.

2.22.1. Rationale For Change

In the case where root of the Activity Tree "exits", the pending (Continue or Previous) Sequencing Request Process would attempt to look at the "parent of the current activity". Since the root does not have a parent, this results in the uncaught exception, as defined in the pseudo code..

If the root of the Activity Tree exits due to an Exit Action Rule, only a "retry" Post Condition rule makes sense – a "retry" means that a new attempt on the root (and hence the entire Activity Tree) should begin. Otherwise, exiting the root should mean that the sequencing session has ended and control be given back to the LMS.

2.22.2. SCORM Update

The SCORM 2004 SN Version 1.3.1 will be updated to ensure that the current sequencing session ends if the root of the activity tree is exited without a corresponding "retry" Post Condition Rule. This will be accomplished by updating the Termination Request Process (TB.2.3) found in Appendix C: *Sequencing Behavior Pseudo Code* of the SCORM 2004 SN Version 1.3.1.

The Termination Request Process (TB.2.3) will be updated:

From:

Termination Request Process [TB.2.3] (for a termination request, ends the current attempt on the *Current Activity*; returns the validity of the termination request; may return a sequencing request; may return an exception code):		
Reference: Activity is Active AM.1.1; Activity is Suspended AM.1.1; Current Activity AM.1.2; End Attempt Process UP.4; Sequencing Exit Action Rules Subprocess TB.2.1; Sequencing Post Condition Rules Subprocess TB.2.2; Terminate Descendent Attempts Process UP.3		
1.	**If** the *Current Activity* is **Not** *Defined* **Then**	If the sequencing session has not begun, there is nothing to terminate
1.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;*	

	Sequencing Request: *n/a;* **Exception***: TB.2.3-1)*	
	End If	
2.	**If** (the termination request is *Exit* **Or** *Abandon*) **And** *Activity is Active* for the *Current Activity* is *False* **Then**	If the current activity has already been terminated, there is nothing to terminate
2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-2)*	
	End If	
3.	**Case:** termination request is *Exit*	
3.1.	Apply the *End Attempt Process* to the *Current Activity*	Ensure the state of the current activity is up to date
3.2.	Apply the *Sequencing Exit Action Rules Subprocess* to the *Current Activity*	Check if any of the current activity's ancestors need to terminate
3.3.	**Repeat**	
3.3.1.	Set the processed exit to *False*	
3.3.2.	Apply the *Sequencing Post Condition Rules Subprocess* to the *Current Activity*	
3.3.3.	**If** the *Sequencing Post Condition Rule Subprocess* returned a termination request of *Exit All* **Then**	
3.3.3.1.	Change the termination request to *Exit All*	
3.3.3.2.	**Break** to the next **Case**	Process an Exit All Termination Request
	End If	
3.3.4.	**If** the *Sequencing Post Condition Rule Subprocess* returned a termination request of *Exit Parent* **Then**	If we exit the parent of the current activity, move the current activity to the parent of the current activity.
3.3.4.1.	**If** the *Current Activity* is **Not** the root of the activity tree **Then**	root of the ...ity tree does ...ave a parent to
3.3.4.1.1.	Set the *Current Activity* to the parent of the *Current Activity*	
3.3.4.1.2.	Apply the *End Attempt Process* to the *Current Activity*	
3.3.4.1.3.	Set processed exit to *True*	Need to evaluate post conditions on the new current activity
3.3.4.2.	**Else**	
3.3.4.2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-4*)	
	End If	
	End If	
3.4.	**Until** processed exit is *False*	

3.5.	**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: is the sequencing request returned by the *Sequencing Post Condition Rule Subprocess*, if one exists, otherwise *n/a;* **Exception***: n/a)*	
	End Case	
4.	**Case:** termination request is *Exit All*	
4.1.	**If** *Activity is Active* for the *Current Activity* is *True* **Then**	Has the completion subprocess and rollup been applied to the current activity yet?
4.1.1.	Apply the *End Attempt Process* to the *Current Activity*	
	End If	
4.2.	Apply the *Terminate Descendent Attempts Process* to the root of the activity tree	
4.3.	Apply the *End Attempt Process* to the root of the activity tree	
4.4.	Set the *Current Activity* to the root of the activity tree	Move the current activity to the root of the activity tree
4.5.	**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: is the sequencing request returned by the *Sequencing Post Condition Rule Subprocess*, if one exists, otherwise an *Exit* sequencing request*;* **Exception***: n/a)*	Inform the sequencer that the sequencing session has ended
	End Case	
5.	**Case:** termination request is *Suspend All*	
5.1.	**If** (the *Activity is Active* for the *Current Activity* is *True*) **Or** (the *Activity is Suspended* for the *Current Activity* is *True*) **Then**	If the current activity is active or already suspended, suspend it and all of its descendents
5.1.1.	Set the *Suspended Activity* to the *Current Activity*	
5.2.	**Else**	
5.2.1.	**If** the *Current Activity* is not the root of the activity tree **Then**	Make sure the current activity is not the root of the activity tree
5.2.1.1.	Set the *Suspended Activity* to the parent of the *Current Activity*	
5.2.2.	**Else**	
5.2.2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-3)*	Nothing to suspend
	End If	
	End If	
5.3.	Form the activity path as the ordered series of all activities from the *Suspended Activity* to the root of the activity tree, inclusive	
5.4.	**If** the activity path is *Empty* **Then**	
5.4.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-5)*	Nothing to suspend
	End If	
5.5.	**For** each activity in the activity path	
5.5.1.	Set *Activity is Active* for the activity to *False*	
5.5.2.	Set *Activity is Suspended* for the activity to *True*	
	End For	
5.6.	Set the *Current Activity* to the root of the activity tree	Move the current activity to the root

		of the activity tree
5.7.	**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: *Exit;* **Exception***: n/a*)	Inform the sequencer that the sequencing session has ended
	End Case	
6.	**Case:** termination request is *Abandon*	
6.1.	Set *Activity is Active* for the *Current Activity* to *False*	
6.2.	**Exit** *Termination Request Process* (**Termination Request**: *Valid;* **Sequencing Request**: *n/a;* **Exception***: n/a*)	
	End Case	
7.	**Case:** termination request is *Abandon All*	
7.1.	Form the activity path as the ordered series of all activities from the *Current Activity* to the root of the activity tree, inclusive	
7.2.	**If** the activity path is *Empty* **Then**	
7.2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-6*)	Nothing to abandon
	End If	
7.3.	**For** each activity in the activity path	
7.3.1.	Set *Activity is Active* for the activity to *False*	
	End For	
7.4.	Set the *Current Activity* to the root of the activity tree	Move the current activity to the root of the activity tree
7.5.	**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: *Exit; Exception: n/a*)	Inform the sequencer that the sequencing session has ended
	End Case	
8.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a; Exception: TB.2.3-7*)	Undefined termination request

pseudo code for termination request process

To:

Termination Request Process [TB.2.3] (for a termination request, ends the current attempt on the *Current Activity*; returns the validity of the termination request; may return a sequencing request; may return an exception code):		
Reference: Activity is Active AM.1.1; Activity is Suspended AM.1.1; Current Activity AM.1.2; End Attempt Process UP.4; Sequencing Exit Action Rules Subprocess TB.2.1; Sequencing Post Condition Rules Subprocess TB.2.2; Terminate Descendent Attempts Process UP.3		
1.	**If** the *Current Activity* is **Not** *Defined* **Then**	If the sequencing session has not begun, there is nothing to terminate
1.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-1*)	
	End If	
2.	**If** (the termination request is *Exit* **Or** *Abandon*) **And** *Activity is Active* for the *Current Activity* is *False* **Then**	If the current activity has already been terminated, there is nothing to terminate
2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-2*)	
	End If	
3.	**Case:** termination request is *Exit*	
3.1.	Apply the *End Attempt Process* to the *Current Activity*	Ensure the state of the current activity is up to date
3.2.	Apply the *Sequencing Exit Action Rules Subprocess* to the *Current Activity*	Check if any of the current activity's ancestors need to terminate
3.3.	**Repeat**	
3.3.1.	Set the processed exit to *False*	
3.3.2.	Apply the *Sequencing Post Condition Rules Subprocess* to the *Current Activity*	
3.3.3.	**If** the *Sequencing Post Condition Rule Subprocess* returned a termination request of *Exit All* **Then**	
3.3.3.1.	Change the termination request to *Exit All*	
3.3.3.2.	**Break** to the next **Case**	Process an Exit All Termination Request
	End If	
3.3.4.	**If** the *Sequencing Post Condition Rule Subprocess* returned a termination request of *Exit Parent* **Then**	If we exit the parent of the current activity, move the current activity to the parent of the current activity.
3.3.4.1.	**If** the *Current Activity* is **Not** the root of the activity tree **Then**	root of the ~~ity~~ tree does ~~have~~ a parent to

3.3.4.1.1.		Set the *Current Activity* to the parent of the *Current Activity*	
3.3.4.1.2.		Apply the *End Attempt Process* to the *Current Activity*	
3.3.4.1.3.		Set processed exit to *True*	Need to evaluate post conditions on the new current activity
3.3.4.2.	**Else**		
3.3.4.2.1.		**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception***: TB.2.3-4*)	
	End If		
3.3.5.	**Else**		
3.3.5.1.		**If** the *Current Activity* is the Root of the Activity Tree **And** the sequencing request returned by the *Sequencing Post Condition Rule Subprocess* is **Not** *Retry* **Then**	If the attempt on the root of the Activity Tree is ending without a Retry, the Sequencing Session also ends
3.3.5.1.1.		**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: *Exit*; **Exception**: *n/a*)	
	End If		
3.4.	**Until** processed exit is *False*		
3.5.	**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: is the sequencing request returned by the *Sequencing Post Condition Rule Subprocess*, if one exists, otherwise *n/a;* **Exception***: n/a*)		
	End Case		
4.	**Case:** termination request is *Exit All*		
4.1.	**If** *Activity is Active* for the *Current Activity* is *True* **Then**		Has the completion subprocess and rollup been applied to the current activity yet?
4.1.1.		Apply the *End Attempt Process* to the *Current Activity*	
	End If		
4.2.	Apply the *Terminate Descendent Attempts Process* to the root of the activity tree		
4.3.	Apply the *End Attempt Process* to the root of the activity tree		
4.4.	Set the *Current Activity* to the root of the activity tree		Move the current activity to the root of the activity tree
4.5.	**Exit** *Termination Request Process* (**Termination Request**: *Valid*; **Sequencing Request**: is the sequencing request returned by the *Sequencing Post Condition Rule Subprocess*, if one exists, otherwise an *Exit* sequencing request*;* **Exception**: *n/a*)		Inform the sequencer that the sequencing session has ended
	End Case		
5.	**Case:** termination request is *Suspend All*		
5.1.	**If** (the *Activity is Active* for the *Current Activity* is *True*) **Or** (the *Activity is Suspended* for the *Current Activity* is *True*) **Then**		If the current activity is active or already suspended, suspend it and all

		of its descendents
5.1.1.	Set the *Suspended Activity* to the *Current Activity*	
5.2.	**Else**	
5.2.1.	**If** the *Current Activity* is not the root of the activity tree **Then**	Make sure the current activity is not the root of the activity tree
5.2.1.1.	Set the *Suspended Activity* to the parent of the *Current Activity*	
5.2.2.	**Else**	
5.2.2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception**: *TB.2.3-3*)	Nothing to suspend
	End If	
	End If	
5.3.	Form the activity path as the ordered series of all activities from the *Suspended Activity* to the root of the activity tree, inclusive	
5.4.	**If** the activity path is *Empty* **Then**	
5.4.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception**: *TB.2.3-5*)	Nothing to suspend
	End If	
5.5.	**For** each activity in the activity path	
5.5.1.	Set *Activity is Active* for the activity to *False*	
5.5.2.	Set *Activity is Suspended* for the activity to *True*	
	End For	
5.6.	Set the *Current Activity* to the root of the activity tree	Move the current activity to the root of the activity tree
5.7.	**Exit** *Termination Request Process* (**Termination Request**: *Valid;* **Sequencing Request**: *Exit;* **Exception**: *n/a*)	Inform the sequencer that the sequencing session has ended
	End Case	
6.	**Case:** termination request is *Abandon*	
6.1.	Set *Activity is Active* for the *Current Activity* to *False*	
6.2.	**Exit** *Termination Request Process* (**Termination Request**: *Valid;* **Sequencing Request**: *n/a;* **Exception**: *n/a*)	
	End Case	
7.	**Case:** termination request is *Abandon All*	
7.1.	Form the activity path as the ordered series of all activities from the *Current Activity* to the root of the activity tree, inclusive	
7.2.	**If** the activity path is *Empty* **Then**	
7.2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a;* **Exception**: *TB.2.3-6*)	Nothing to abandon
	End If	
7.3.	**For** each activity in the activity path	
7.3.1.	Set *Activity is Active* for the activity to *False*	
	End For	
7.4.	Set the *Current Activity* to the root of the activity tree	Move the current activity to the root of the activity tree
7.5.	**Exit** *Termination Request Process* (**Termination Request**: *Valid;* **Sequencing Request**: *Exit;* **Exception**: *n/a*)	Inform the sequencer that the sequencing session has ended
	End Case	

8.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a; **Exception**: TB.2.3-7*)	Undefined termination request

pseudo code for termination request process

2.23. The Completion Status Value of "not attempted" is Not Mapped to Sequencing Tracking Information

The SCORM 2004 RTE Data Model element `cmi.completion_status`, described in the SCORM 2004 RTE Version 1.3.1, allows a value of "not attempted" to be set, but does not describe how an LMS should map this value to the SCO's associated activity's Sequencing Tracking Information.

As described in the SCORM 2004 RTE Version 1.3.1, setting a SCO's completion status to "not attempted" asserts that the learner has not attempted the SCO in any significant way, but this result is clearly distinct from "unknown".

2.23.1. Rationale For Change

Since the mapping of "not attempted" to a Sequencing Tracking Information data element does not exist and this value may impact sequencing rule evaluations, the SCORM 2004 RTE Version 1.3.1 will be updated to define the mapping of the "not attempted" value to Sequencing Tracking Information.

2.23.2. SCORM Update

If a SCO sets cmi.completion_status to "not attempted", the SCO's associated Sequencing Tracking Information should be considered "incomplete". *Section 4.2.4: Completion Status* will be updated to define this mapping. Specifically, Table 4.2.4a: Dot-notation Binding for the Completion Status Data Model Element will be updated as follows:

From:

Sequencing Impacts:
- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "unknown", the Attempt Progress Status for the learning activity associated with the SCO shall be false.
- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "completed", the Attempt Progress Status for the learning activity associated with the SCO shall be true, and the Attempt Completion Status for the learning activity associated with the SCO shall be true.
- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "incomplete", the Attempt Progress Status for the learning activity associated with the SCO shall be true, and the Attempt Completion Status for the learning activity associated with the SCO shall be false.

To:

LMS Behavior Requirements:
- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "unknown", the Attempt Progress Status for the learning activity associated with the SCO shall be false.
- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "completed", the Attempt Progress Status for the learning activity associated with the SCO shall be true, and the Attempt Completion Status for the learning activity associated with the SCO shall be true.

- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "incomplete", the Attempt Progress Status for the learning activity associated with the SCO shall be true, and the Attempt Completion Status for the learning activity associated with the SCO shall be false.
- If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status* of the SCO to "not attempted", the Attempt Progress Status for the learning activity associated with the SCO shall be true and the Attempt Completion Status for the learning activity associated with the SCO shall be false.

2.24. Undefined Behavior for LMS Handling of Unique Identifier Collisions

The SCORM 2004 RTE Version 1.3.1 does not clearly define how LMSs are to behave when a unique identifier collision is encountered. The Objective identifier data model element (`cmi.objectives.n.id`) is supposed to be unique within the scope of the SCO. If a SCO tries to set the objective identifiers as follows:

- SetValue("cmi.objectives.0.id","objective-id-1")
- SetValue("cmi.objectives.1.id","objective-id-2")
- SetValue("cmi.objectives.2.id","objective-id-1")

the last SetValue call violates the unique identifier constraint defined in SCORM 2004. The SCO tries to set new objective information (in array position 2) into the collection of objectives, but the identifier (objective-id-1) has already been used earlier (in array position 0).

SCORM 2004 defines a specific error condition and behavior for this scenario (see *Section 3.1.7.6.6: Unique Identifier Constraint Violated*); however, the SCORM 2004 RTE Version 1.3.1 does not specifically call this error condition out. Table 4.2.17a: Dot-notation Binding for the Objectives Data Model Element does not define this error condition (located in the API Implementation Requirements section).

2.24.1. Rationale For Change

Due to the ambiguity around how LMS' should behave under the scenarios described above, the SCORM 2004 RTE Version 1.3.1, specifically Table 4.2.1.17a, will be updated to clearly define the LMS behavior for the handling of identifier collisions.

2.24.2. SCORM Update

If a SCO tries to set data as described above, the LMS shall set the appropriate error code. Table 4.2.17a will be updated as follows:

From:

SetValue(): The LMS shall set the *cmi.objectives.n.id* to the supplied value in the SetValue() request, set the error code to "0" – No error and return "`true`".

- If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request.
- Collection data model elements are required to be set in sequential order. If a SCO does not set objectives in a sequential order, then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*.

To:

> **SetValue():** The LMS shall set the *cmi.objectives.n.id* to the supplied value in the SetValue() request, set the error code to "0" – No error and return "`true`".
> - If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request.
> - Collection data model elements are required to be set in sequential order. If a SCO does not set objectives in a sequential order, then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*.
> - If the supplied value of the SetValue() is a value that has already been used (not unique within the set of objective information) in an earlier array position within a learner attempt, then the LMS shall set the error code to 351 – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions.* The LMS shall not alter the state of the data model element based on the request.

2.25. Clarification and Changes Needed for Non-Tracked Activities

The SCORM SN Version 1.3.1 does not adequately define LMS behaviors regarding activities that are "not tracked" (Delivery Controls element tracked = false). An interoperability problem may occur when an LMS incorrectly applies read maps (Read Objective Satisfied Status or Read Objective Normalized Measure) from a shared global objective to a "not tracked" activity.

2.25.1. Rationale For Change

In all cases, when tracking status information for the "not tracked" activity (regardless if the activity is a leaf or a cluster) is required for a sequencing evaluation, the LMS' sequencing implementation shall apply the default – "unknown" – status to that evaluation. The SCORM 2004 SN Version 1.3.1 will be updated to clearly describe this requirement clarification.

2.25.2. SCORM Update

Section 3.13.1: Tracked will be updated as follows:

From:

The *Tracked* element indicates whether any tracking status information (refer to *Section 4.2: Tracking Model*) is being managed for the activity. This element contains a boolean (True/False) value. The default value for *Tracked*, if not defined explicitly for the activity, is True.

If the *Tracked* element on an activity is defined as False, the LMS will behave is if it does not initialize, manage or access any tracking status information for the activity. All evaluations requiring tracking status information will receive the default ("unknown") value. Activities that have the *Tracked* element defined as False are not included in any rollup evaluations for their parent.

To:

The *Tracked* element indicates whether any tracking status information (refer to *Section 4.2: Tracking Model*) is being managed for the activity. This element contains a boolean (True/False) value. The default value for *Tracked*, if not defined explicitly for the activity, is True.

If the *Tracked* element on an activity is defined as False, the LMS will behave is if it does not initialize, manage or access any tracking status information for the activity. All

evaluations requiring tracking status information will receive the default ("unknown") value. Activities that have the *Tracked* element defined as False are not included in any rollup evaluations for their parent.

ADL Note: Content developers should be aware that declaring an activity "not tracked" (*Tracked* equals False) will prevent any "read" Objective Maps from being honored since the LMS is not managing any state for the "not tracked" activity. The LMS will return "unknown" for all status evaluations on the "not tracked" activity.

2.26. Error in Pseudo Code for the Choice Sequencing Request Process in Dealing with Constrained Chioces

This addendum addresses a discrepancy with the Choice Sequencing Request Process (SB.2.9), defined in the SCORM 2004 Sequencing and Navigation Version 1.3.1. The pseudo code contains an error that prohibits proper evaluation of a choice navigation request when a constrained activity is encountered.

2.26.1. Rationale for Change

Because of this error, LMS sequencing implementations cannot properly evaluate a choice navigation request when a constrained activity is encountered. Specifically, line 12.5.5 is too strict in reducing the set of "constrained choices". In this line, if the target activity is either a "constrained activity" or the "activity to consider", then the process fails with an error. However, both of these activities should be included in the constrained set of choices.

2.26.2. SCORM Update

The pseudo code will be updated to permit these two activities to be included in the set of choices. Line 12.5.5 of the Choice Sequencing Request Process (SB.2.9) will be updated to correct this descrepancy.

The updated Choice Sequencing Request Process is reproduced in whole for easy reference.

Choice Sequencing Request Process [SB.2.9] (for a target activity; may return a delivery request; may change the *Current Activity*; may return an exception code):	
Reference: Activity is Active AM.1.1; Activity is Suspended AM.1.1; Available Children AM.1.1; Check Activity Process UP.5; Choice Activity Traversal Subprocess SB.2.4; Current Activity AM.1.2; End Attempt Process UP.4; Flow Subprocess SB.2.3; Sequencing Control Mode Choice SM.1; Sequencing Control Choice Exit SM.1; Sequencing Rules Check Process UP.2; Terminate Descendant Attempts Process UP.3*; adlseq:constrainedChoice SCORM SN; adlseq:preventActivation SCORM SN*	
1. **If** there is no target activity **Then**	There must be a target activity for choice
1.1. **Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-1*)	Nothing to deliver
End If	
2. **If** the target activity is not the root of the activity tree **Then**	
2.1. **If** the *Available Children* for the parent of the target activity does not contain the target activity **Then**	The activity is currently not available
2.1.1. **Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-2*)	Nothing to deliver
End If	

	End If	
3.	Form the activity path as the ordered series of activities from the root of the activity tree to the target activity, inclusive	
4.	**For** each activity in the activity path	
4.1.	Apply the *Sequencing Rules Check Process* to the activity and the *Hide from Choice sequencing rules*	Cannot choose something that is hidden
4.2.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	
4.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-3*)	Nothing to deliver
	End If	
	End For	
5.	**If** the target activity is not the root of the activity tree **Then**	
5.1.	**If** the *Sequencing Control Mode Choice* for the parent of the target activity is *False* **Then**	Confirm that control mode allow 'choice' of the target
5.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-4*)	Nothing to deliver
	End If	
	End If	
6.	**If** the *Current Activity* is *Defined* **Then**	Has the sequencing session already begun?
6.1.	Find the common ancestor of the *Current Activity* and the target activity	
7.	**Else**	
7.1.	Set common ancestor is the root of the activity tree	No, choosing the target will start the sequencing session
	End If	
8.	**Case:** *Current Activity* and target activity are identical	Case #1 - select the current activity
8.1.	**Break All** Cases	Nothing to do in this case
	End Case	
9.	**Case:** *Current Activity* and the target activity are siblings	Case #2 - same cluster; move toward the target activity
9.1.	Form the activity list as the ordered sequence of activities from the *Current Activity* to the target activity, exclusive of the target activity	We are attempted to walk toward the target activity. Once we reach the target activity, we don't need to test it.
9.2.	**If** the activity list is *Empty* **Then**	Nothing to choose
9.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)	Nothing to deliver
	End If	
9.3.	**If** the target activity occurs after the *Current Activity* in preorder traversal of the activity tree **Then**	

9.3.1.	traverse is *Forward*	
9.4.	**Else**	
9.4.1.	traverse is *Backward*	
	End If	
9.5.	**For** each activity on the activity list	
9.5.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the traverse direction	
9.5.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**	
9.5.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Choice Activity Traversal Subprocess*)	Nothing to deliver
	End If	
	End For	
9.6.	**Break All** Cases	
	End Case	
10.	**Case:** *Current Activity* and common ancestor are the same **Or** *Current Activity* is **Not** D*efined*	Case #3 - path to the target is forward in the activity tree
10.1.	Form the activity path as the ordered series of activities from the common ancestor to the target activity, exclusive of the target activity	
10.2.	**If** the activity path is *Empty* **Then**	
10.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)	Nothing to deliver
	End If	
10.3.	**For** each activity on the activity path	
10.3.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the *Forward* direction	
10.3.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**	
10.3.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Choice Activity Traversal Subprocess*)	Nothing to deliver
	End If	
10.3.3.	**If** *Activity is Active* for the activity is *False* **And** (the activity is **Not** the common ancestor **And** *adlseq:preventActivation* for the activity is *True*) **Then**	If the activity being considered is not already active, make sure we are allowed to activate it
10.3.3.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-6*)	Nothing to deliver
	End If	
	End For	
10.4.	**Break All** Cases	
	End Case	
11.	**Case:** Target activity is the common ancestor of the *Current Activity*	Case #4 - path to the target is backward in the activity tree
11.1.	Form the activity path as the ordered series of activities from the *Current Activity* to the target activity, inclusive	
11.2.	**If** the activity path is *Empty* **Then**	
11.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5)*	Nothing to deliver

SCORM 2004 2nd Edition Addendum Version 1.2

	End If	
11.3.	**For** each activity on the activity path	
11.3.1.	**If** the activity is not the last activity in the activity path **Then**	
11.3.1.1.	**If** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	Make sure an activity that should not exit will exit if the target is delivered.
11.3.1.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-7*)	Nothing to deliver
	End If	
	End If	
	End For	
11.4.	**Break All** Cases	
	End Case	
12.	**Case:** Target activity is forward from the common ancestor activity	Case #5 - target is a descendant activity of the common ancestor
12.1.	Form the activity path as the ordered series of activities from the *Current Activity* to the common ancestor, inclusive	
12.2.	**If** the activity path is *Empty* **Then**	
12.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)	Nothing to deliver
	End If	
12.3.	Set constrained activity to *Undefined*	
12.4.	**For** each activity on the activity path	Walk up the tree to the common ancestor
12.4.1.	**If** the activity is not the last activity in the activity path **Then**	
12.4.1.1.	**If** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	Make sure an activity that should not exit will exit if the target is delivered
12.4.1.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-7*)	Nothing to deliver
	End If	
	End If	
12.4.2.	**If** constrained activity is *Undefined* **Then**	Find the closest constrained activity to the current activity
12.4.2.1.	**If** *adlseq:constrainedChoice* for the activity is *True* **Then**	
12.4.2.1.1.	Set constrained activity to activity	
	End If	
	End If	
	End For	
12.5.	**If** constrained activity is *Defined* **Then**	
12.5.1.	**If** the target activity is *Forward* in the activity tree relative to the constrained activity **Then**	
12.5.1.1.	traverse is *Forward*	'Flow' in a forward direction to see what

			activity comes next
12.5.2.	**Else**		
12.5.2.1.	traverse is *Backward*		'Flow' in a backward direction to see what activity comes next
	End If		
12.5.3.	Apply the *Choice Flow Subprocess* to the constrained activity in the traverse direction		
12.5.4.	Set activity to consider to the activity identified by the *Choice Flow Subprocess*		
12.5.5.	**If** the target activity is **Not** an available descendent of the activity to consider **And** the target activity is **Not** the activity to considered **And** the target activity is **Not** the constrained activity **Then**		Make sure the target activity is within the set of 'flow' constrained choices
12.5.5.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-8*)		
	End If		
	End If		
12.6.	Form the activity path as the ordered series of activities from the common ancestor to the target activity, exclusive of the target activity		
12.7.	**If** the activity path is *Empty* **Then**		
12.7.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-5*)		Nothing to deliver
	End If		
12.8.	**If** the target activity is forward in the activity tree relative to the *Current Activity* **Then**		Walk toward the target activity
12.8.1.	**For** each activity on the activity path		
12.8.1.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the *Forward* direction		
12.8.1.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**		
12.8.1.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: the exception identified by the *Choice Activity Traversal Subprocess*)		Nothing to deliver
	End If		
12.8.1.3.	**If** *Activity is Active* for the activity is *False* **And** (the activity is **Not** the common ancestor **And** *adlseq:preventActivation* for the activity is *True*) **Then**		If the activity being considered is not already active, make sure we are allowed to activate it
12.8.1.3.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-6*)		Nothing to deliver
	End If		
	End For		
12.9.	**Else**		
12.9.1.	**For** each activity on the activity path		
12.9.1.1.	**If** *Activity is Active* for the activity is *False* **And** (the activity is **Not** the common ancestor **And** *adlseq:preventActivation* for the activity is *True*) **Then**		If the activity being considered is not already

		active, make sure we are allowed to activate it
12.9.1.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-6*)	Nothing to deliver
	End If	
	End For	
	End If	
12.10.	**Break All Cases**	
	End Case	
13.	**If** the target activity is a leaf activity **Then**	
13.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: the target activity; **Exception**: *n/a)*	
	End If	
14.	Apply the *Flow Subprocess* to the target activity in the *Forward* direction with consider children equal to *True*	The identified activity is a cluster. Enter the cluster and attempt to find a descendant leaf to deliver
15.	**If** the *Flow Subprocess* returns *False* **Then**	Nothing to deliver, but we succeeded in reaching the target activity - move the current activity
15.1.	Apply the *Terminate Descendent Attempts Process* to the common ancestor	
15.2.	Apply the *End Attempt Process* to the common ancestor	
15.3.	Set the *Current Activity* to the target activity	
15.4.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*; **Exception**: *SB.2.9-9*)	Nothing to deliver
16.	**Else**	
16.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: for the activity identified by the *Flow Subprocess*; **Exception**: *n/a*)	
	End If	

This page intentionally left blank.

SECTION 3
SCORM 2004
Clarification/Enhancement Addenda

This page intentionally left blank.

3.1. Ambiguous Information Defined in the language_type Data Type

This addendum addresses a clarification to the language found in *Section 4.1.1.7: Data Types* of the SCORM 2004 RTE Version 1.3.1. The last paragraph found in the language_type section is ambiguous and is not needed in this section. This section is used to describe the specifics of the data types used by the SCORM 2004 RTE Version 1.3.1.

3.1.1. SCORM Update

The `cmi.learner_preference.language` data model element, found in *Section 4.2.13: Learner Preference*, will be updated to add the additional information found in *Section 4.1.1.7: Data Types* to further describe the special case of the `language_type` being an empty characterstring. The Data Model Element Implementation Requirements section will be updated as follows:

From:

Data Model Element Implementation Requirements:
- **Data Type:** language_type (SPM 250)
- **Value Space:** iso-646 [4]
- **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the language_type data type. The default language shall be "" (empty characterstring).

To:

Data Model Element Implementation Requirements:
- **Data Type:** language_type (SPM 250) or empty characterstring
- **Value Space:** iso-646 [4]
- **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the language_type data type. The default language shall be "" (empty characterstring).

3.2. Clarification of Learner Session Initialization Requirements

This addendum addresses a clarification in the behavior of the `cmi.session_time` and `cmi.exit` RTE Data Model elements found in *Section 4.2.21: Session Time* and *Section 4.2.8: Exit* respectively, of the SCORM 2004 RTE Version 1.3.1. The clarification is centered on the how to handle the `cmi.session_time` and `cmi.exit` value between learner sessions.

3.2.1. SCORM Update

The LMS Behavior Requirements section of the `cmi.session_time` and `cmi.exit` data model element found in *Section 4.2.21: Session Time* and *Section 4.2.8: Exit* respectively, will be updated to clarify behavior on handling the `cmi.session_time` value between learner sessions as follows:

Section 4.2.8: Exit

From:

LMS Behavior Requirements:
- The data model element is mandatory and shall be implemented by an LMS as write-only.
- The value is completely controlled by the SCO. The SCO is responsible for setting this value. If the SCO does not set the cmi.exit data model element, then the default value (empty characterstring – "") shall be used. If the LMS receives a request to get the *cmi.exit* value, then the LMS shall adhere to the requirements listed below for API Implementation Requirements.

To:

LMS Behavior Requirements:
- The data model element is mandatory and shall be implemented by an LMS as write-only.
- The value is completely controlled by the SCO. The SCO is responsible for setting this value. If the SCO does not set the cmi.exit data model element, then the default value (empty characterstring – "") shall be used. If the LMS receives a request to get the *cmi.exit* value, then the LMS shall adhere to the requirements listed below for API Implementation Requirements.
- If there are additional learner sessions within a learner attempt, the *cmi.exit* becomes uninitialized (i.e., reinitialized to its default value of ("") - empty characterstring) at the beginning of each additional learner session within the learner attempt.

Section 4.2.21: Session Time

From:

LMS Behavior Requirements:

- The data model element is mandatory and shall be implemented by an LMS as write-only.

- Since this data model element is implemented by the LMS as write-only, the LMS is not responsible for initializing this data model element. It is the responsibility of the SCO to manage this value. The LMS is only responsible for accepting a SetValue() call to this data model element and perform the accumulation with *cmi.total_time*.

- Since a SCO is not required to set a value for this data model element (not required to keep track of the session time), an LMS shall keep track of session time from the time the LMS launches the SCO. If the SCO reports a different session time, then the LMS shall use the session time as reported by the SCO instead of the session time as measured by the LMS.

To:

LMS Behavior Requirements:

- The data model element is mandatory and shall be implemented by an LMS as write-only.

- Since this data model element is implemented by the LMS as write-only, the LMS is not responsible for initializing this data model element. It is the responsibility of the SCO to manage this value. The LMS is only responsible for accepting a SetValue() call to this data model element and perform the accumulation with *cmi.total_time*.

- Since a SCO is not required to set a value for this data model element (not required to keep track of the session time), an LMS shall keep track of session time from the time the LMS launches the SCO. If the SCO reports a different session time, then the LMS shall use the session time as reported by the SCO instead of the session time as measured by the LMS.

- If there are additional learner sessions within a learner attempt, the *cmi.session_time* becomes uninitialized at the beginning of each additional learner session within the learner attempt.

3.3.   Setting the Current Activity to None

This addendum addresses a clarification in the SCORM 2004 SN Version 1.3.1 to explicitly define when an LMS should set the Global State Information (SCORM 2004 SN Version 1.3.1, *Section 4.2.1.6: Global State Information*) Current Activity element to *None* (or *undefined*).

3.3.1.   SCORM Update

The first paragraph of *Section 2.3: Starting and Stopping a Sequencing Session* will be updated as follows:

From:

A Sequencing Session is the time from when an attempt on the root activity of an Activity Tree begins until that attempt ends. The SCORM Sequencing Behaviors only specify which navigation requests can begin a sequencing session, but they do not specify when or how those navigation requests are triggered. Generally, the LMS will issue a *Start* navigation request in recognition of some system event, e.g., a login, begin course, etc. It is recommended, if the previous sequencing session ended due to a *Suspend All* navigation request, the LMS should issue a *Resume All* navigation request instead of a *Start*.

To:

A Sequencing Session is the time from when an attempt on the root activity of an Activity Tree begins until that attempt ends; outside of the context of a Sequencing Session the Current Activity is considered to be *undefined*. The SCORM Sequencing Behaviors only specify which navigation requests can begin a Sequencing Session, but they do not specify when or how those navigation requests are triggered. Generally, the LMS will issue a *Start* navigation request in recognition of some system event, e.g., a login, begin course, etc. It is recommended, if the previous sequencing session ended due to a *Suspend All* navigation request, the LMS should issue a *Resume All* navigation request instead of a *Start*.

The first set of steps defined in *Section 4.3.1: Sequencing Loop* will be updated as follows:

From:

Begin Sequencing Session

(1)   The learner initiates access to the LMS (e.g., accesses the system, logs in, etc.) and establishes a context within a particular unit of instruction (e.g., selects a course, a content organization, etc.).

> (2)   The LMS initiates a sequencing process by issuing a *Start*, *Resume All*, or *Choice* navigation request.
>
> (3)   The Navigation Behavior translates the *Start*, *Resume All*, or *Choice* navigation request into the appropriate sequencing request and processes it. The sequencing session "officially" begins when an activity is identified for delivery – one successful pass through the following Sequencing Loop.

To:

> Begin Sequence Session **:
>
> (1)   The learner initiates access to the LMS (e.g., accesses the system, logs in, etc.) and establishes a context within a particular unit of instruction (e.g., selects a course, a content organization, etc.).
>
> ** Prior to the beginning of a sequencing session, the Current Activity shall be considered to be *None* (or *undefined*).
>
> (2)   The LMS initiates a sequencing process by issuing a *Start*, *Resume All*, or *Choice* navigation request.
>
> (3)   The Navigation Behavior translates the *Start*, *Resume All*, or *Choice* navigation request into the appropriate sequencing request and processes it. The sequencing session "officially" begins when an activity is identified for delivery – one successful pass through the following Sequencing Loop.

3.4. Incorrect Rollup Condition Definition

This addendum addresses inconsistency between some of the tables provided in the IMS Simple Sequencing (SS) Specification Version 1.0 and the SCORM 2004 SN Version 1.3.1.

3.4.1. Rationale For Change

Some of the tables identified in the Sequencing Definition Model section of the IMS SS Version 1.0 are inconsistent with the tables in the SCORM 2004 SN Version 1.3.1.

For example, there is a discrepancy between the definition of Objective Measure Known in Table 3.7.2a and Table 3.4.2a of the SCORM 2004 SN Version 1.3.1. The correct language for Table 3.7.2a should read: "Objective Measure Known – The condition evaluates to True if the Objective Measure Status for the rolled-up objective associated with the child activity is True." The correct language for Table 3.2.2a should read: "Objective Measure Known – The condition evaluates to True if the Objective Measure Status for the objective associated with the activity (indicated by the Referenced Objective) is True."

3.4.2. SCORM Update

The SCORM 2004 SN Version 1.3.1 will be updated to reflect the language in the IMS SS specification. Other tables that may be affected will also be updated.

3.5. Ambiguous Language for the timeinterval(second, 10,2) Data Requirements

This addendum addresses ambiguity in the timeinterval(second,10,2) value requirements, as defined in the SCORM 2004 RTE Version 1.3.1.

3.5.1. Rationale For Change

Section 4.1.1.7: Data Types of the SCORM 2004 RTE Version 1.3.1 states:

"The character literals designators "P","Y","M","D","T","H","M","S" shall appear if the corresponding non-zero value is present."

However, the "P" is always required. "T" is required if hours, minutes or seconds are used. The current language implies that they can be left off if the corresponding value for the literal is zero.

3.5.2. SCORM Update

The language in question for the timeinterval(second, 10,2) value will be updated to state:

"If the data model element, that is of type timeinterval(second,10,2) contains a value, then the designator P shall be present. If the value of years, months, days, hours, minutes or seconds is zero, the value and corresponding designation (e.g., Y if there is no year) may be omitted, but at least one designator and value shall be present in addition to the designator P. The designator T shall be omitted if all of the time components (hours, minutes, and seconds) are zero."

3.6. Incorrect Completion Status Determination/Success Status Determination Operators

This addendum addresses incorrect Completion Status Determination and Success Status Determination operators, as found in the SCORM 2004 RTE Version 1.3.1.

3.6.1. Rationale For Change

Table 4.2.4.1a Completion Status Determination incorrectly use the > (greater than) operator. Some of the information in the LMS Behavior column incorrectly describes the condition of a value being greater than (>)another value. The correct condition should be that the value should be greater than or equal (>=) another value.

3.6.2. SCORM Update

Table 4.2.4.1a will be updated accordingly to include the correct operator. The table will be updated to change the operator to >=.

3.7. Conflicting Definitions of <adlnav:presentation> Element

This addendum addresses conflicting definitions of the <adlnav:presentation> element in the SCORM 2004 CAM Version 1.3.1.

3.7.1. Rationale For Change

Section 5.2.1.1: <presentation> Element states that: "The element shall only appear, if needed, as a child of a leaf <item> element that references a SCO." *Section 5.2.1.1.1.1: <hideLMSUI> Element* (which is a descendant of the <presentation> element) states that its values applies "when a child of this activity cluster is the current activity," which means this activity is cluster and has children. These definitions contradict one another. The <hideLMSUI> element is intended to affect only the leaf activity it is applied to.

3.7.2. SCORM Update

Section 5.2.1.1.1.1: <hideLMSUI> Element will be updated to reflect the intended application of the <presentation> element (*Section 5.2.1.1: <presentation> Element*). The definition of the <hideLMSUI> elements affects will be updated as follows:

- **previous** : If specified, the LMS shall not display a "Previous" navigation device when this activity is the current activity.
- **continue** : If specified, the LMS shall not display a "Continue" navigation device when this activity is the current activity.
- **exit** : If specified, the LMS shall not display an "Exit" navigation device when this activity is the current activity.
- **abandon** : If specified, the LMS shall not display a "Abandon" navigation device when this activity is the current activity.

3.8. Root of the Activity Tree Cannot be Targeted for Choice

This addendum addresses missing pseudo code in the Navigation Request Process (NB.2.1) that prevents the root of the Activity Tree from being targeted for choice after the sequencing session has already begun.

3.8.1. Rationale For Change

The Navigation Request Process (NB.2.1) is missing pseudo code that would allow the root of an Activity Tree from being targeted with a Choice Navigation Request after the sequencing session has already begun. Furthermore, if the root of the Activity Tree is targeted by a Choice Navigation Request, an exception will be thrown on line 7.1.1.2.4.1 – "No Activities to Consider." This is incorrect behavior. The intent of the Choice Navigation Request is that any activity can be targeted for choice, including the root.

3.8.2. SCORM Update

The pseudo code that allows the root of the Activity Tree to be targeted by a Choice Navigation Request after the sequencing session has begun will be added. The fix for this is to remove the phrase:, "excluding the common ancestor" from line 7.1.1.2.2. This fix causes targeting the root of the Activity tree to result in an activity path of one activity, the root, and clause 7.1.1.2.3.* will process the request successfully.

From:

Navigation Request Process [NB.2.1] (for a navigation request and possibly a specified activity, returns the validity of the navigation request; may return a termination request, a sequencing request, and/or a target activity; may return an exception code):		
Reference: Current Activity AM.1.2; Sequencing Control Choice SM.1; Sequencing Control Choice Exit SM.1; Sequencing Control Flow SM.1; Sequencing Control Forward Only SM.1; Suspended Activity AM.1.2		
1.	**Case:** navigation request is *Start*	
1.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has not already begun.
1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Start*; **Target Activity**: *n/a;* **Exception***: n/a*)	
1.2.	**Else**	
1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-1*)	
	End If	
	End Case	

SCORM 2004 2nd Edition Addendum Version 1.2

2.	**Case:** navigation request is *Resume All*	
2.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has not already begun
2.1.1.	**If** the *Suspended Activity* is *Defined* **Then**	Make sure the previous sequencing session ended with a suspend all request
2.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Resume All*; **Target Activity**: *n/a;* **Exception***: n/a*)	
2.1.2.	**Else**	
2.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-3*)	
	End If	
2.2.	**Else**	
2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-1*)	
	End If	
	End Case	
3.	**Case:** navigation request is *Continue*	
3.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun
3.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-2*)	
	End If	
3.2.	**If** the *Current Activity* is not the root of the activity tree **And** the *Sequencing Control Flow* for the parent of the *Current Activity* is *True* **Then**	Validate that a 'flow' sequencing request can be processed from the current activity
3.2.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the current activity has not been terminated, terminate the current the activity
3.2.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Continue*; **Target Activity**: *n/a;* **Exception***: n/a*)	
3.2.2.	**Else**	
3.2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Continue*; **Target Activity**: *n/a;* **Exception***: n/a*)	
	End If	
3.3.	**Else**	
3.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-4*)	Flow is not enabled or the current activity is the root

		of the activity tree
	End If	
	End Case	
4.	**Case:** navigation request is *Previous*	
4.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun
4.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-2*)	
	End If	
4.2.	**If** the *Current Activity* is not the root of the activity tree **Then**	There is no activity logically 'previous' to the root of the activity tree
4.2.1.	**If** the *Sequencing Control Flow* for the parent of the *Current Activity* is *True* **And** the *Sequencing Control Forward Only* for the parent of the *Current Activity* is *False* **Then**	Validate that a 'flow' sequencing request can be processed from the current activity
4.2.1.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the current activity has not been terminated, terminate the current the activity
4.2.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Previous*; **Target Activity**: *n/a;* **Exception***: n/a*)	
4.2.1.2.	**Else**	
4.2.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Previous*; **Target Activity**: *n/a;* **Exception***: n/a*)	
	End If	
4.2.2.	**Else**	
4.2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-5*)	Violates control mode
	End If	
4.3.	**Else**	
4.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-6*)	Cannot move backward from the root of the activity tree
	End If	
	End Case	
5.	**Case:** navigation request is *Forward*	Behavior not defined
5.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-7*)	

	End Case	
6.	**Case:** navigation request is *Backward*	Behavior not defined
6.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-7*)	
	End Case	
7.	**Case:** navigation request is *Choice*	
7.1.	**If** the activity specified by the *Choice* navigation request exists within the activity tree **Then**	Make sure the target activity exists in the activity tree
7.1.1.	**If** the activity specified by the *Choice* navigation request is the root of the activity tree **Or** the *Sequencing Control Choice* for the parent of the activity specified by the *Choice* navigation request is *True* **Then**	Validate that a 'choice' sequencing request can be processed on the target activity
7.1.1.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Attempt to start the sequencing session through choice
7.1.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request*;* **Exception***: n/a*)	
	End If	
7.1.1.2.	**If** the activity specified by the *Choice* navigation request is **Not** a sibling of the *Current Activity* **Then**	We are always allowed to choose a sibling of the current activity
7.1.1.2.1.	Find the common ancestor of the *Current Activity* and the activity specified by the *Choice* navigation request	
7.1.1.2.2.	Form the activity path as the ordered series of activities from the *Current Activity* to the common ancestor, excluding the common ancestor	The common ancestor will not terminate as a result of processing the choice sequencing request, unless the common ancestor is the Current Activity – the current activity should always be included in the activity path
7.1.1.2.3.	**If** the activity path is **Not** *Empty* **Then**	
7.1.1.2.3.1.	**For** each activity in the activity path	Make sure that 'choosing' the target will not force an active activity to terminate, if that activity does not allow choice to terminate it

7.1.1.2.3.1.1.	**If** *Activity is Active* for the activity is *True* **And** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	
7.1.1.2.3.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-8*)	Violates control mode
	End If	
	End For	
7.1.1.2.4.	**Else**	
7.1.1.2.4.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-9*)	
	End If	
	End If	
7.1.1.3.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the current activity has not been terminated, terminate the current the activity
7.1.1.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request*;* **Exception***: n/a*)	
7.1.1.4.	**Else**	
7.1.1.4.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request*;* **Exception***: n/a*)	
	End If	
7.1.2.	**Else**	
7.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-10*)	Violates control mode
	End If	
7.2.	**Else**	
7.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-11*)	Target activity does not exist
	End If	
	End Case	
8.	**Case:** navigation request is *Exit*	
8.1.	**If** the *Current Activity* is *Defined* **Then**	Make sure the sequencing session has already begun
8.1.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	Make sure the current activity has not already been terminated
8.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**:	

	Valid; **Termination Request**: *Exit*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a*) *;* **Exception**: *n/a*)	
8.1.2.	**Else**	
8.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*; **Exception**: *NB.2.1-12*)	Activity has already terminated
	End If	
8.2.	**Else**	
8.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-2*)	
	End If	
	End Case	
9.	**Case:** navigation request is *Exit All*	
9.1.	**If** the *Current Activity* is *Defined* **Then**	If the sequencing session has already begun, unconditionally terminate all active activities
9.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit All*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a;* **Exception**: *n/a*)	
9.2.	**Else**	
9.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-2*)	
	End If	
	End Case	
10.	**Case:** navigation request is *Abandon*	
10.1.	**If** the *Current Activity* is *Defined* **Then**	Make sure the sequencing session has already begun
10.1.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	Make sure the current activity has not already been terminated
10.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Abandon*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a;* **Exception**: *n/a*)	
10.1.2.	**Else**	
10.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-12*)	
	End If	
10.2.	**Else**	
10.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-2*)	
	End If	
	End Case	
11.	**Case:** navigation request is *Abandon All*	
11.1.	**If** the *Current Activity* is *Defined* **Then**	If the sequencing session has already

		begun, unconditionally abandon all active activities
11.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Abandon All*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a;* **Exception**: *n/a*)	
11.2.	**Else**	
11.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-2*)	
	End If	
	End Case	
12.	**Case:** navigation request is *Suspend All*	
12.1.	**If** the *Current Activity* is *Defined* **Then**	If the sequencing session has already begun
12.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Suspend All*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a;* **Exception**: *n/a*)	
12.2.	**Else**	
12.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-2*)	
	End If	
	End Case	
13.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-13*)	Undefined navigation request

pseudo code for navigation request process

To:

Navigation Request Process [NB.2.1] (for a navigation request and possibly a specified activity, returns the validity of the navigation request; may return a termination request, a sequencing request, and/or a target activity; may return an exception code):		
Reference: Current Activity AM.1.2; Sequencing Control Choice SM.1; Sequencing Control Choice Exit SM.1; Sequencing Control Flow SM.1; Sequencing Control Forward Only SM.1; Suspended Activity AM.1.2		
1.	**Case:** navigation request is *Start*	
1.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has not already begun.
1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Start*; **Target Activity**: *n/a;* **Exception**: *n/a*)	
1.2.	**Else**	
1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception**: *NB.2.1-1*)	
	End If	
	End Case	
2.	**Case:** navigation request is *Resume All*	

2.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has not already begun
2.1.1.	**If** the *Suspended Activity* is *Defined* **Then**	Make sure the previous sequencing session ended with a suspend all request
2.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Resume All*; **Target Activity**: *n/a*; **Exception**: *n/a*)	
2.1.2.	**Else**	
2.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception**: *NB.2.1-3*)	
	End If	
2.2.	**Else**	
2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception**: *NB.2.1-1*)	
	End If	
	End Case	
3.	**Case:** navigation request is *Continue*	
3.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun
3.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception**: *NB.2.1-2*)	
	End If	
3.2.	**If** the *Current Activity* is not the root of the activity tree **And** the *Sequencing Control Flow* for the parent of the *Current Activity* is *True* **Then**	Validate that a 'flow' sequencing request can be processed from the current activity
3.2.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the current activity has not been terminated, terminate the current the activity
3.2.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Continue*; **Target Activity**: *n/a*; **Exception**: *n/a*)	
3.2.2.	**Else**	
3.2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Continue*; **Target Activity**: *n/a*; **Exception**: *n/a*)	
	End If	
3.3.	**Else**	
3.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception**: *NB.2.1-4*)	Flow is not enabled or the current activity is the root of the activity tree

	End If	
	End Case	
4.	**Case:** navigation request is *Previous*	
4.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun
4.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-2*)	
	End If	
4.2.	**If** the *Current Activity* is not the root of the activity tree **Then**	There is no activity logically 'previous' to the root of the activity tree
4.2.1.	**If** the *Sequencing Control Flow* for the parent of the *Current Activity* is *True* **And** the *Sequencing Control Forward Only* for the parent of the *Current Activity* is *False* **Then**	Validate that a 'flow' sequencing request can be processed from the current activity
4.2.1.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the current activity has not been terminated, terminate the current the activity
4.2.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Previous*; **Target Activity**: *n/a;* **Exception***: n/a*)	
4.2.1.2.	**Else**	
4.2.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Previous*; **Target Activity**: *n/a;* **Exception***: n/a*)	
	End If	
4.2.2.	**Else**	
4.2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-5*)	Violates control mode
	End If	
4.3.	**Else**	
4.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-6*)	Cannot move backward from the root of the activity tree
	End If	
	End Case	
5.	**Case:** navigation request is *Forward*	Behavior not defined
5.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-7*)	
	End Case	
6.	**Case:** navigation request is *Backward*	Behavior not defined

SCORM 2004 2nd Edition Addendum Version 1.2

6.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-7*)	
	End Case	
7.	**Case:** navigation request is *Choice*	
7.1.	**If** the activity specified by the *Choice* navigation request exists within the activity tree **Then**	Make sure the target activity exists in the activity tree
7.1.1.	**If** the activity specified by the *Choice* navigation request is the root of the activity tree **Or** the *Sequencing Control Choice* for the parent of the activity specified by the *Choice* navigation request is *True* **Then**	Validate that a 'choice' sequencing request can be processed on the target activity
7.1.1.1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Attempt to start the sequencing session through choice
7.1.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request*;* **Exception***: n/a*)	
	End If	
7.1.1.2.	**If** the activity specified by the *Choice* navigation request is **Not** a sibling of the *Current Activity* **Then**	We are always allowed to choose a sibling of the current activity
7.1.1.2.1.	Find the common ancestor of the *Current Activity* and the activity specified by the *Choice* navigation request	
7.1.1.2.2.	Form the activity path as the ordered series of activities from the *Current Activity* to the common ancestor, ~~excluding the common ancestor~~	The common ancestor will not terminate as a result of processing the choice sequencing request, unless the common ancestor is the Current Activity – the current activity should always be included in the activity path
7.1.1.2.3.	**If** the activity path is **Not** *Empty* **Then**	
7.1.1.2.3.1.	**For** each activity in the activity path	Make sure that 'choosing' the target will not force an active activity to terminate, if that activity does not allow choice to terminate it
7.1.1.2.3.1.1.	**If** *Activity is Active* for the activity is *True* **And** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	

7.1.1.2.3.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception***: NB.2.1-8*)	Violates control mode
	End If	
	End For	
7.1.1.2.4.	**Else**	
7.1.1.2.4.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception***: NB.2.1-9*)	
	End If	
	End If	
7.1.1.3.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the current activity has not been terminated, terminate the current the activity
7.1.1.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request*;* **Exception***: n/a*)	
7.1.1.4.	**Else**	
7.1.1.4.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request*;* **Exception***: n/a*)	
	End If	
7.1.2.	**Else**	
7.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Termination Request**: *n/a*; **Sequencing Request**: *n/a*; **Target Activity**: *n/a*; **Exception***: NB.2.1-10*)	Violates control mode
	End If	
7.2.	**Else**	
7.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*; **Exception***: NB.2.1-11*)	Target activity does not exist
	End If	
	End Case	
8.	**Case:** navigation request is *Exit*	
8.1.	**If** the *Current Activity* is *Defined* **Then**	Make sure the sequencing session has already begun
8.1.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	Make sure the current activity has not already been terminated
8.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a*) *;* **Exception***: n/a*)	
8.1.2.	**Else**	

SCORM 2004 2nd Edition Addendum Version 1.2

8.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a; **Exception***: NB.2.1-12*)	Activity has already terminated
	End If	
8.2.	**Else**	
8.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a; **Exception***: NB.2.1-2*)	
	End If	
	End Case	
9.	**Case:** navigation request is *Exit All*	
9.1.	**If** the *Current Activity* is *Defined* **Then**	If the sequencing session has already begun, unconditionally terminate all active activities
9.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit All*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a; **Exception***: n/a*)	
9.2.	**Else**	
9.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a; **Exception***: NB.2.1-2*)	
	End If	
	End Case	
10.	**Case:** navigation request is *Abandon*	
10.1.	**If** the *Current Activity* is *Defined* **Then**	Make sure the sequencing session has already begun
10.1.1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	Make sure the current activity has not already been terminated
10.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Abandon*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a; **Exception***: n/a*)	
10.1.2.	**Else**	
10.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a; **Exception***: NB.2.1-12*)	
	End If	
10.2.	**Else**	
10.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a; **Exception***: NB.2.1-2*)	
	End If	
	End Case	
11.	**Case:** navigation request is *Abandon All*	
11.1.	**If** the *Current Activity* is *Defined* **Then**	If the sequencing session has already begun, unconditionally abandon all active

		activities
11.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Abandon All*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a;* **Exception***: n/a*)	
11.2.	**Else**	
11.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-2*)	
	End If	
	End Case	
12.	**Case:** navigation request is *Suspend All*	
12.1.	**If** the *Current Activity* is *Defined* **Then**	If the sequencing session has already begun
12.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Suspend All*; **Sequencing Request**: *Exit*; **Target Activity**: *n/a;* **Exception***: n/a*)	
12.2.	**Else**	
12.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-2*)	
	End If	
	End Case	
13.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a;* **Exception***: NB.2.1-13*)	Undefined navigation request

pseudo code for navigation request process

3.9. Choice Exit not Enforced on the Activity Tree Root

This addendum addresses a missing clause to enforce Choice Exit on the Activity Tree Root in the Navigation Request Process (NB.2.1).

3.9.1. Rationale For Change

If the root of the Activity Tree has Choice Exit = false, this would indicate that the content designer intends that a new attempt on the root activity cannot be initiated through a Choice Navigation Request – a learner cannot "restart" the Activity Tree by selecting its root.

The Navigation Request Process (NB.2.1) is missing a clause to enforce Choice Exit = false on the root of the Activity Tree, although it enforces this constraint on all other activities in the tree (see line 7.1.1.2.3.1.1.). In this situation, the Navigation Request Process (incorrectly) allows the root of the activity tree to be the target of a Choice Navigation Request; processing this request would result in the current activity's content being unloaded and a subsequent failure of processing the Choice Sequencing Request.

3.9.2. SCORM Update

NB.2.1 will be updated so that the root of the activity tree can be targeted by a Choice Navigation Request when it has Choice Exit = false. The fix for this is to remove, "excluding the common ancestor" from line 7.1.1.2.2 – this is the same fix proposed in *Section 3.8: Root of the Activity Tree Cannot be Targeted for Choice*. See this section for more details on the exact change.

This page intentionally left blank.

APPENDIX A
Acronym Listing

This page intentionally left blank.

Acronym Listing

ADL	Advanced Distributed Learning
API	Application Programming Interface
CAM	Content Aggregation Model
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
LMS	Learning Management System
RFC	Request For Comment
RTE	Run-Time Environment
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
SN	Sequencing and Navigation
SPM	Smallest Permitted Maximum
SS	Simple Sequencing
URL	Universal Resource Locator
XML	Extensible Markup Language

This page intentionally left blank.

SCORM 2004 2nd Edition Addendum Version 1.2

APPENDIX B
Document Revision History

This page intentionally left blank.

Document Revision History

Version	Release Date	Description of Change
1.0	09/15/2004	Initial Version. Added the following Defect Addenda: • Addendum: Handling of Invalid SetValue() Requests for Data Model Element Collections • Addendum: Ambiguous Pseudo-Code in Case #4 of Choice Sequencing Request Process • Addendum: Misevaluation of Traversal Direction • Addendum: Measure Rollup Should not be Applied to Leaf Activities • Addendum: Invalid Default Value defined for the measureSatisfactionIfActive attribute • Addendum: Incorrect SPM for the <dataFromLMS> Element • Addendum: Handling of Reserved Delimiters • Addendum: Deprecating the adlcp:persistState Attribute Added the following Clarification/Enhancement Addenda: • Addendum: Ambiguous information defined in the language_type Data Type • Addendum: Clarification of Learner Session Initialization Requirements • Addendum: Setting the Current Activity to None
1.1	10/29/2004	Added the following Defect Addenda: • Addendum: Language Type Syntax • Addendum: Objective Satisfied By Measure Evaluation Behavior Discrepancy • Addendum: Manifest Requirements Inconsistency • Addendum: Error in Pseudo-Code Limit Conditions Check Process • Addendum: Error in Pseudo-Code Choice Flow Tree Traversal Sub-process • Addendum: Error in Pseudo-Code Previous Sequencing Request Process • Addendum: Application of Measure Satisfaction if Active Behavior • Addendum: Initialization of a SCO's Objectives from Sequencing Information • Addendum: Error in Pseudo-Code Navigation Request Process Added the following Clarification/Enhancement Addenda: • Addendum: Incorrect Rollup Condition Definition

		• Addendum: Ambiguous Language for the timeinterval(second,10,2) Data Requirements • Addendum: Incorrect Completion Status Determination/Success Status Determination Operators
1.2	4/15/2005	Added the following Defect Addenda: • Addendum 2.17: Variable Error in Measure Rollup Process (RB.1.1) • Addendum 2.18: Incorrect definition of Extended Rollup Set • Addendum 2.19: Portability (Interoperability) of URLs between Windows and UNIX Systems • Addendum 2.20: The <adlnav:presentation> Element Should be Permitted on an Asset Resource • Addendum 2.21: Default Value for timeinterval(second,10,2) Should be any Value that Evaluates to Zero • Addendum 2.22: Exit Action Rule on the Activity Tree Root does not End the Sequencing Session • Addendum 2.23: The Completion Status Value of "not attempted" is Not Mapped to Sequencing Tracking Information • Addendum 2.24: Undefined Behavior for LMS Handling of Unique Identifier Collisions • Addendum 2.25: Clarification and Changes Needed for Non-Tracked Activities • Addendum 2.26: Error in Pseudo-Code for the Choice Sequencing Request Process in Dealing with Constrained Choices Added the following Clarification/Enhancement Addenda: • Addendum 3.8: Root of the Activity Tree Cannot be Targeted for Choice • Addendum 3.9: Choice Exit not Enforced on the Activity Tree Root