IMS Simple Sequencing Information and Behavior Model

Version 1.0 Final Specification

Document Name: IMS Simple Sequencing Information and Behavior Model

Date: 03 March 2003

Table of Contents

I. Introduction

I.1 Overview

This document, the “IMS Simple Sequencing Specification,” defines a method for representing the intended behavior of an authored learning experience such that any learning technology system (LTS) can sequence discrete learning activities in a consistent way. A learning designer or content developer declares the relative order in which elements of content are to be presented to the learner and the conditions under which a piece of content is selected, delivered, or skipped during presentation.

This specification defines the required behaviors and functionality that conforming systems must implement. It incorporates rules that describe the branching or flow of learning activities through content according to the outcomes of a learner’s interactions with content. This representation of intended instructional flow may be created manually or with authoring systems that produce output that conforms to this specification. While learning content developers need to know how to create and describe content sequences, authoring systems may hide the details of the models presented in this specification. The representation of sequencing may be interchanged between systems designed to deliver instructional activities to learners. The components of an LTS used to execute the specified rules and behaviors, when content is delivered to a learner, are referred to in this specification as a ‘sequencing engine’.

Simple Sequencing is labeled as *simple* because it includes a limited number of widely used sequencing behaviors, not because the specification itself is simple. Simple Sequencing is not all-inclusive. In particular, Simple Sequencing does not address, but does not necessarily preclude, artificial intelligence-based sequencing, schedule-based sequencing, sequencing requiring data from closed external systems and services (e.g., sequencing of embedded simulations), collaborative learning, customized learning, or synchronization between multiple parallel learning activities.

Simple Sequencing recognizes only the role of the learner and does not define sequencing capabilities that utilize or are dependent on other actors, such as instructors, mentors, or peers. This specification does not prohibit usage in contexts involving other actors; however, it does not define the roles of other actors or sequencing behaviors that result from participation of other actors.

Simple Sequencing enables digital content to be rendered and presented to a learner in a Web-based environment; however, it is not restricted to a Web-based environment. The nature of the control and communication interfaces, and the mechanisms for mediating interactions between a learner and a LTS, are not part of this specification. In addition, issues such as look and feel, presentation style, and placement of navigation controls are not defined by this specification.

Simple Sequencing provides an external representation, via extensions to the “IMS Content Packaging XML Binding” [3][1], to exchange sequencing descriptions between different runtime components or LTSs. Although the Simple Sequencing Specification is based on the same content organization and tree structure as the Content Packaging Specification, Simple Sequencing does not require Content Packaging. However, Content Packaging is currently the only defined mechanism for exchanging definition model instances.

I.2 IMS Simple Sequencing Components

Simple Sequencing is described in the following set of interrelated documents:

- **Simple Sequencing Information and Behavior Model** (this document) – This document is the normative reference that defines the behavior and an Information Model consisting of a Sequencing Definition Model, a Tracking Model, and an Activity State Model. It is written for implementers of sequencing services and authoring tools.
- **XML Binding [4] –** This document describes how the Sequencing Definition Model is represented as extensions to the organization component of the manifest of the XML binding of the IMS Content Packaging Specification.

1. The numbers in brackets refer to the bibliography: subsection I.5.

- **Best Practice and Implementation Guide [5] –** This document describes implementation considerations for those implementing the behaviors described in this specification.

I.3 Simple Sequencing Specification Sections

This specification defines a small subset of all possible sequencing operations and the informational elements required to describe the desired operations and their parameters. To enable interoperability, systems that deliver learning content must be able to interpret the sequencing information and exhibit the behaviors defined in this specification. This presentation is divided into several sections, each section being standalone and describing one part of the Simple Sequencing Specification.

- *Sequencing Definition Model* – The information model in Simple Sequencing used to describe the desired sequencing behavior (see part **SM**).
- *Tracking Model* – The information model in Simple Sequencing used to record information about the results of a learner's interactions with activities, and the learner's record for objectives (e.g., completion, measure) to control the selection and sequencing of other activities (see part **TM**).
- *Activity State Model* – The information model in Simple Sequencing used to record information about the state or status of a learner's interactions with an activity and a set of global attributes for activities (see part **AM**).
- *Overall Sequencing Process* – The overall Simple Sequencing process that relates the selection and randomization, navigation, sequencing, exit, rollup and delivery processes (see part **OP**).
- *Navigation Behavior Model* – The Simple Sequencing process that evaluates a navigation request and determines the sequencing and termination requests that should be processed to identify and deliver content to the learner (see part **NB**).
- *Termination Behavior Model* – The Simple Sequencing process that evaluates a termination request to end an attempt on an activity (see part **TB**).
- *Rollup Behavior Model* – The Simple Sequencing process that computes the results data for an activity from the results data from the children of the activity (see part **RB**).
- *Selection and Randomization Behavior Model* – The Simple Sequencing process that selects and reorders a subset of activities (see part **SR**).
- *Sequencing Behavior Model* – The Simple Sequencing process that evaluates a sequencing request in terms of the content model described by the activity tree and determines what actual content object should be delivered to the learner (see part **SB**).
- *Delivery Behavior Model* – The Simple Sequencing process that validates that the learning resources for the identified activity may be delivered, i.e., all of the conditions that apply to the delivery of the content for the activity and attempt still hold (see part **DB**).
- *Utility Processes* – Sequencing processes that are utilized by multiple behavioral models (see part **UP**).
- *Extensibility* – The extensibility mechanisms in Simple Sequencing (see part **E**).
- *Conformance* – Requirements on an implementation to conform to the Simple Sequencing Specification (see part **C**).
- *Glossary* – A collection of terms used throughout the Simple Sequencing document set to describe parts of the information models of sequencing behaviors specific to this description of Simple Sequencing (see part **G**).
- *Use Cases* – A collection of sample sequencing cases used throughout the Simple Sequencing development process to guide the selection of features and functions (see part **UC**).

Section	**Mnemonic**
Introduction	I
Sequencing Definition Model	SM
Tracking Model	TM
Activity State Model	AM
Overall Sequencing Process	OP
Navigation Behavior Model	NB
Termination Behavior Model	TB
Rollup Behavior Model	RB
Selection and Randomization Behavior Model	SR
Sequencing Behavior Model	SB
Delivery Behavior Model	DB
Utility Processes	UP
Extensibility	E
Conformance	C
Glossary	G
Use Cases	UC

The sections are intentionally independent so that an implementer or reader has all the information related to the topic and other key information needed to interpret and implement the corresponding part of a Simple Sequencing system. Each section may include:

- *Information Model* – The detailed description of the corresponding information model (the Sequencing Definition Model, the Tracking Model, and the Activity State Model). Each information model is divided into logically related sets of data attributes. Each set describes the data items in a binding-independent fashion.
- *Behavior* – The detailed description of the behaviors and procedures that describe the parts of Simple Sequencing (for each of the behavior models). Each behavior includes one or more process descriptions in pseudo code.

I.4 Assumptions

I.4.1 Activity Tree

The Simple Sequencing Specification relies on the concept of *learning activities*. A learning activity may be loosely described as an instructional event or events embedded in a content resource, or as an aggregation of activities that eventually resolve to discrete content resources with their contained instructional events.

Content in Simple Sequencing is organized into a hierarchical structure. Each activity may include one or more child activities. Each activity has an associated set of sequencing behaviors, defined by the sequencing definition model (see Sequencing Definition Model **SM**). The sequencing behaviors describe how the activity or how the children of the activity are used to create the desired learning experience.

The sequencing behavior process (see Sequencing Behavior Model **SB**) traverses the activity tree, applying the sequencing rules, to determine the activities to deliver to the learner. Content resources from the identified activities are delivered to the learner to create the desired learning experience.

Simple Sequencing makes no requirements on the structure, organization, or instantiation of the activity tree. The tree and the associated sequencing definitions may be static or they may be dynamically created. How to create, represent, and maintain the activity tree and associated sequencing definition is not specified.

I.4.2 Sequencing Loop

The following sequencing loop describes how the various behavior processes interact during sequencing and delivery. The description assumes the activity tree exists and has been initialized.

Begin Sequencing Session

1) The learner initiates access to the content delivery environment or LTS (e.g., accesses the system, logs in) and establishes a context within a particular unit of instruction (e.g., selects a course or content aggregation).
2) The LTS initiates a sequencing session 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 corresponding sequencing request and processes it. The sequencing session ‘officially’ begins when an activity is identified for delivery (successful completion of steps 4 and 5).

Start of Sequencing Loop

4) Using the information in the tracking model and the sequencing request, the sequencing behavior traverses the activity tree attempting to identify the appropriate activity (node in the tree) to deliver to the learner.
5) The delivery behavior determines if the identified activity can be delivered, and if so, prepares to deliver the activity’s content resource(s) to the learner. If the selected activity cannot be delivered, then the overall sequencing process stops and waits for another navigation request (step 8).
6) The learner interacts with the content resource. The sequencing processes are idle and waiting for requests while the learner interacts with the content resource.
7) The activity may report values that update the various tracking model elements during the learner’s interactions with the content resource.
8) The learner, delivery system, or activity invokes a navigation event, such as “Continue”, “Previous”, “Choose activity X”, “Abandon”, “Exit”, etc…
9) The LTS informs the sequencing process of the navigation event by issuing a navigation request.
10) The navigation behavior translates the navigation request into a termination request and a sequencing request. If the navigation request indicates that the learner wants to end the sequencing session, the sequencing session ends (the behavior of ending the sequencing session and the persistence of the activity state model is unspecified and left to the implementation).
11) If the activity triggered the navigation request by terminating or exiting, it may report additional values that update the tracking model. The activity then exits. The rollup process is invoked to determine the effects of any state changes that occurred as a result of delivering the content resource for the activity. The process updates the tracking model for the activity and for any of its parent activities within the activity tree.
12) The process repeats beginning at step 4, until the sequencing session ends.

Several variations exist on the order described above, including:

- Selection and randomization of activities may occur prior to the processing of a sequencing request.
- An activity may report state and status information and exit without triggering a specific navigation request.
- Only certain classes of resources and activities may report information to the tracking model during delivery of content and learner interaction with the content.

The sequencing process may not be able to identify an activity to deliver, or the activity may not be delivered due to failure to meet the defined delivery conditions.

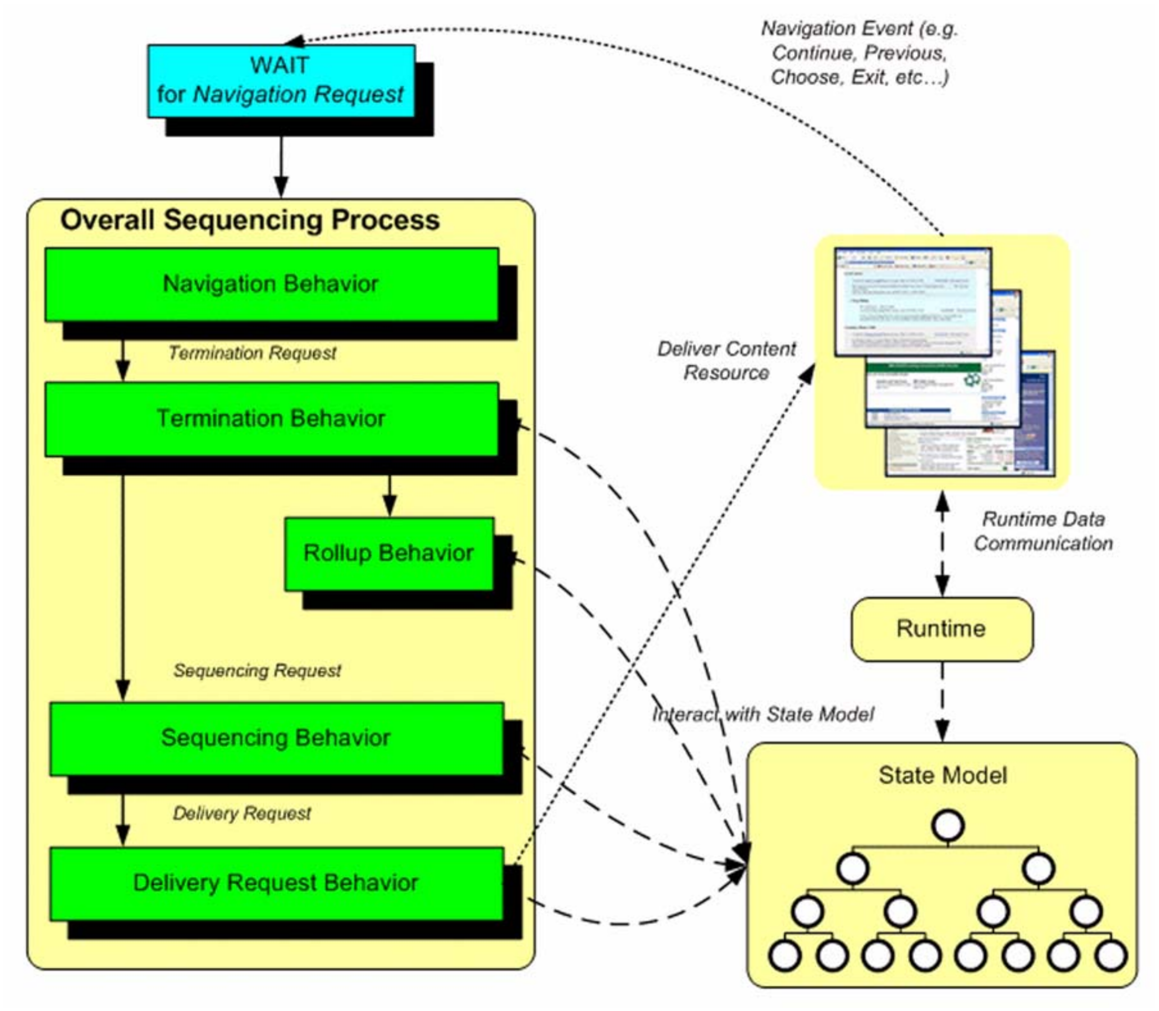

Figure I.1 - The various steps in the sequencing process.

Figure I.1 above shows the various steps in the sequencing process. The control process is shown on the left. In normal operation, the overall sequencing process flows from navigation behavior to termination behavior to sequencing behavior to delivery behavior, followed by a *wait* while the learner interacts with the content resource, as described in steps 4-12 above.

The right side of Figure I.1 shows the learner's view of the learning experience. A content resource is delivered to the learner. The learner interacts with the content resource and results may be returned to the tracking model. The learner triggers an event that maps to a navigation request. The navigation request triggers the various steps in the sequencing process.

Throughout all sequencing processes, a collection of state and tracking model data is maintained. Content resources may directly set values in the tracking model through a runtime communications interface to tracking model; this interface is not part of the specification and is not required. All of the other sequencing processes access and update elements of the tracking models.

Changes to the state of an activity occur because of learner interaction with a content resource delivered for the activity or one of its descendents. If an external event affects the tracking, such as an instructor changing a grade, the model assumes processes are invoked as required to update the activity state model.

Changes to the activity state model and tracking model that occur outside of the scope of delivering the learning experience via the sequencing process are outside of the scope of the specification. Systems that implement such side effects and additional capabilities must deal with these conditions accordingly.

I.4.3 Rule-Based Model

The behaviors described in the Simple Sequencing Specification are defined in terms of sequencing information (see Sequencing Definition Model **SM**) associated with activities in the activity tree. Sequencing information includes 'rules' that provide a means to describe conditional sequencing behavior at content development time. These 'rules' and tracking status information affect the various sequencing processes during content delivery.

I.4.4 Default and Authored Behavior

In the absence of explicitly defined sequencing information, activity trees exhibit default sequencing behaviors when content resources are delivered. These default behaviors and associated values are specified in the sequencing information (see Sequencing Definition Model **SM**). Content developers may change default behaviors by explicitly defining sequencing information and associating it with activities.

I.4.5 Objectives

Learning objectives are separate from activities in Simple Sequencing. Learning objectives represent a set of locally and globally scoped data items, each with a satisfaction status and a satisfaction measure. Simple Sequencing makes no assumption as to how to interpret the objective (e.g., is it a competency, is it a mastery, or is it simply a shared value?). Activities may have more than one associated local objective and may reference multiple globally shared objectives. Multiple activities may reference the same global objective, thus sharing the data values.

The resolution of local and global objective IDs is not specified. An objective may be shared within a single activity tree or may be shared globally across multiple instantiations of tracking models. The lifetime of shared global objectives and the scope of sharing is not specified; it is determined by implementations. Activities may reference multiple objectives, thus providing a mechanism for activities to have sub-objectives. However, this specification makes no assumptions about the semantics or meanings of multiple objectives associated with an activity.

I.4.6 Auxiliary Resources

An activity may have auxiliary resources associated with it that provide additional services or resources for the learner. This specification does not define any semantics or meanings for these auxiliary resources. This specification does not define which resources may be made available, or how the resources are to be used; it only provides a means for the auxiliary resources to be associated with an activity.

I.4.7 Presentation and Environment

Simple Sequencing makes no assumption as to how content and controls are rendered or presented to the learner (e.g., style, placement, GUI widgets). How events are triggered in the interface environment, how they are communicated to the LTS, and how the LTS delivers content to the learner and the learner's environment are not specified.

While an external event may trigger a navigation event and a resulting navigation request, the external event also may trigger other actions. These behaviors are not specified.

I.4.8 Suspending and Resuming Activities

An activity may be suspended and later resumed. Certain navigation requests result in an activity or collection of activities being suspended. Other activities may be delivered while the activity is suspended. A suspended activity may be resumed later without counting as a new activity attempt. Additionally, the suspended activity may be abandoned or exited.

The current sequencing session may be suspended and later resumed at the last activity experienced by the learner.

I.4.9 Starting and Stopping the Sequencing Process

Simple Sequencing does not specify how to start the overall sequencing process or how to stop the process. Generally, the LTS will recognize some event, e.g., a course login, to start sequencing. Some other event, e.g., a logout, is mapped to the appropriate navigation, exit and sequencing requests, after which the LTS may terminate the overall sequencing process.

I.4.10 Data Persistence

Simple Sequencing does not specify how data (e.g., tracking data) is to be persisted across multiple instantiations of the overall sequencing process for a particular learner and activity tree or learning experience, e.g., across multiple login sessions. Implementations shall persist control, tracking, and state data at least until the current attempt on the activity tree is terminated. Such an attempt may include one or more login sessions. LTS policies govern whether to persist data beyond that time; such policies are beyond the scope of this specification.

I.4.11 Content Types

The Simple Sequencing Specification is independent of the types of learning content and learning objects and can be used to sequence all types of content. For example, content may include simple static Seb pages, MIME resources of any type (e.g., DOC, PDF files), services and proxies for services, and dynamically created objects. In particular, content need not use a communications adapter, such as the AICC/IEEE/ADL SCORM API communications adapter used by sharable content objects [2]. Nor does Simple Sequencing require a runtime service or assume the existence of other data models, such as the CMI data model [1].

I.4.12 Active/Passive Content

Simple Sequencing relies on values within the tracking model to control sequencing. Simple Sequencing does not specify how the tracking model values for an activity or an objective are set or updated. Simple Sequencing differentiates between active and passive content and supports both active and passive content on an activity-by-activity basis. Active content is responsible for setting elements of the tracking model directly. For passive content, Simple Sequencing will automatically set certain values in the tracking model. Simple Sequencing makes no assumptions about how content actually behaves, e.g., passive content may set values in the tracking model, active content may fail to set values Simple Sequencing relies only on the declaration of how the content *should* behave.

I.4.13 Relationship to IMS Content Packaging Specification

The Simple Sequencing Specification relies on the concept of *learning activities*. A learning activity may be loosely described as an instructional event or events embedded in a content resource, or as an aggregation of activities that eventually resolve to discrete content resources with their contained instructional events.

The IMS Content Packaging Specification provides a ready structure for relating a learning activity to a content resource – the *item* element and its relationship to a *resource* element. Furthermore, *item* elements can be clustered into collections, with such collections contained in a parent *organization* element, just like learning activities may clustered together in a parent activity or activities. Therefore, Simple Sequencing maps the concept of a learning activity to an *item* element or a collection of *item* elements within an *organization* element, and to an *organization* element itself, as defined by the Content Packaging Specification. The Content Packaging XML Binding is extended by this specification to define how sequencing information is associated with packaged content.

The process of defining a specific sequence of learning activities begins with the creation of an aggregation of content. The Content Packaging Specification enables aggregations to be interchanged among systems. As shown in the figure below, the content packaging *organization* element and each *item* element within it can have defined sequencing behaviors through the association of sequencing information defined in the sequencing definition Model (see Sequencing Definition Model **SM**). Sets of sequencing definition elements can be created and defined for or referenced by items within the organization, as well as for the organization itself. A single set of sequencing definition elements and values can be referenced by more than one item and therefore are reusable.

All information defining sequencing behavior for an organization is present within that organization. Where multiple organizations are present within a manifest, each organization may have its own set of sequencing behaviors.

IMS Simple Sequencing only describes how sequencing information relates to IMS Content Packaging, but does not prohibit sequencing information and behavior from being used in other contexts.

This version of Simple Sequencing does not use or prohibit the use of sub manifests, defined in IMS Content Packaging.

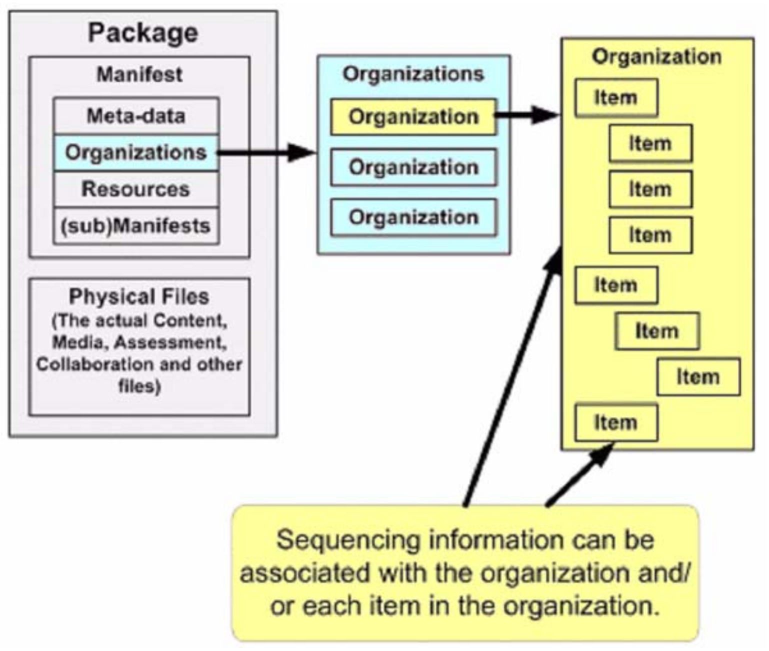
Package
Manifest
Meta-data
Organizations
Resources
(sub)Manifests
Physical Files
(The actual Content, Media, Assessment, Collaboration and other files)
Organizations
Organization
Organization
Organization
Organization
Item
Item
Item
Item
Item
Item
Item
Item
Sequencing information can be associated with the organization and/ or each item in the organization.

Figure I.2 - How sequencing information can be associated with an IMS Content Package.

I.5 Nomenclature

The following abbreviations and acronyms are used in this document:

ADL	Advanced Distributed Learning
AICC	Aviation Industry CBT Committee
ANSI	American National Standards Institute
API	Application Programming Interface
CBT	Computer Based Training
CMI	Computer Managed Instruction
CPI	Content Packaging Interchange
DTD	Document Type Definition
IEEE	Institute of Electronic and Electrical Engineering
ISO	International Standards Organization
JTC	Joint Technical Committee
LTS	Learning Technology System
LTSC	Learning Technology Standards Committee
SCORM	Shareable Content Object Reference Model
SS	Simple Sequencing
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language

I.6 References

[1] IEEE P1484.11.1 Draft Standard for Data Model to Content Object Communication

[2] IEEE P1484.11.2 Draft Standard for ECMAScript API for Content to Runtime Services Communication

[3] *IMS Content Packaging XML Binding* v1.1.2 Final Specification, IMS Global Learning Consortium, Inc., August 2001

[4] *IMS Persistent, Location-Independent Resource Identifier Handbook* v1.0, Final Specification, IMS Global Learning Consortium, Inc., April 2001

[5] *IMS Simple Sequencing XML Binding* v1.0 Final Specification, IMS Global Learning Consortium, Inc., March 2003

[6] *IMS Simple Sequencing Best Practice and Implementation Guide* v1.0 Specification, IMS Global Learning Consortium, Inc., March 2003

SM. Sequencing Definition Model

The Simple Sequencing process uses information about the desired sequencing behavior to control the sequencing, selection and delivery of activities to the learner. The intended sequence is described by a specific set of data attributes. These attributes are associated with learning activities in the activity tree to describe the sequencing behavior. The set of attributes used by Simple Sequencing is called the "sequencing definition model".

The sequencing definition model consists of:

- *Sequencing Control Modes* – controls for types of sequencing requests that may apply to a collection of activities.
- *Sequencing Rule Description* – rules applied to an activity used to specify sequencing behaviors for the activity.
- *Limit Conditions Description* – limits on how many times, how long and when an activity is allowed.
- *Rollup Rule Description* – rules that specify how tracking data from an activity or its associated objectives is produced from the results of the child activities.
- *Objective Description* – the learning objectives associated with an activity.
- *Objective Map* – the mapping of local objective information to and from shared global objectives.
- *Rollup Controls* – controls which data from an activity contributes to the rollup of the parent activity.
- *Selection Controls* – controls how an activity's children are selected.
- *Randomization Controls* – controls how an activity's children are ordered.
- *Delivery Controls* – controls when progress and objective information for an activity are recorded.

There are no overall behavior requirements on the use and instantiation of the sequencing definition model. Individual parts of the model describe how the sequencing definitions are associated with the activity tree to define intended learning experiences. The use of the sequencing definition model is detailed as part of the behavior descriptions of the navigation, sequencing, delivery, exit, selection, randomization, and rollup processes.

Sequencing definitions describe the intended learning experience for a learner. How the learning experience and definition for a learner relate to the definition for a cohort of learners (e.g., individualized experiences versus a common experience for all members of the cohort) is not specified. A sequencing definition model instance defines a learning experience independently of how it is instantiated for one or more learners.

SM.1 Sequencing Control Modes

Sequencing control mode information (the set of attributes shown below) includes descriptions of the types of sequencing behaviors specified for an activity. Sequencing control mode information for an activity includes the associated data listed below.

Simple Sequencing processes may reference the sequencing control modes for any activity in the activity tree. Sequencing control modes are optional. Default data is used if the data is not defined for a given activity.

The control modes are not exclusive; multiple control modes may be specified.

No.	Name	Description	Value Space	Default Value
1	*Sequencing Control Choice*	Indicates that a *Choice* sequencing request is permitted (True or False) to target the children of the activity.	Boolean	True
2	*Sequencing Control Choice Exit*	Indicates that an active child of this activity is permitted to terminate (True or False) if a *Choice* sequencing request is processed.	Boolean	True
3	*Sequencing Control Flow*	Indicates the *Flow Subprocess* (SB.2.3) may be applied (True or False) to the children of this activity.	Boolean	False
4	*Sequencing Control Forward Only*	Indicates that backward targets (in terms of activity tree traversal) are not permitted (True or False) for the children of this activity.	Boolean	False

No.	Name	Description	Value Space	Default Value
5	*Use Current Attempt Objective Information*	Indicates that the objective progress information (TM.1.1) for the children of the activity will only be used (True or False) in rule evaluations and rollup if that information was recorded during the current attempt on the activity.	Boolean	True
6	*Use Current Attempt Progress Information*	Indicates that the attempt progress information (TM.1.2.2) for the children of the activity will only be used (True or False) in rule evaluations and rollup if that information was recorded during the current attempt on the activity.	Boolean	True

SM.2 Sequencing Rule Description

Sequencing rule description (the set of attributes shown below) specifies the details of individual rule-based sequencing behaviors for an activity. Sequencing rule description information for an activity includes the associated data listed below.

Simple Sequencing processes may reference the sequencing rules for any activity in the activity tree. Sequencing rules are optional. The data attributes below specify one rule. Each activity may have an unlimited number of sequencing rules. Sequencing rules are defined only where needed. Default data is used if the data is not instantiated for a given rule, if needed.

- The data attributes below describe one rule. Each activity may have an unlimited number of sequencing rules.
- The general format of a rule can be expressed informally as: *If condition set Then action*.
- There may be multiple conditions.
- The conditions may be combined with a single *and* combination (all conditions must be True) or a single *or* combination (only one condition must be True).
- Individual condition values may be negated before being combined in the rule evaluation.
- No other operators on the conditions are defined.
- Each condition references an item in the tracking model for the activity or one of its associated objectives.
- There is one action that may result if the rule conditions evaluate to True.

Actions are divided into three groups:

- *Preconditions* – Actions that control sequencing decisions and delivery of a specific activity. Rules that include such actions are used to determine if the activity will be delivered.
- *Post Conditions* – Actions that control sequencing flow by issuing sequencing requests. Rules that include such actions are applied when an activity terminates.
- *Exit Actions* – Actions that terminate an activity. Rules that include such actions are applied when a descendent of an activity exits.

The *Rule Conditions*, *Condition Combination,* and *Rule Actions* are tokens in a vocabulary. The tokens have no semantics or meanings themselves. The description of the condition or action is the complete definition of the required behavior.

No.	Name	Description	Value Space	Default Value
1	*Condition Combination*	How rule conditions are combined in evaluating the rule. • *All* – The rule condition evaluates to True if and only if all of the individual rule conditions evaluate to True (logical **and**). • *Any* – The rule condition evaluations to True if and only if any of the individual rule conditions evaluates to True (logical **or**).	Vocabulary	All
2	*Rule Conditions*	An unordered collection of conditions for a sequencing rule. A rule may include multiple rule conditions.	Collection 0:N	

No.	Name	Description	Value Space	Default Value
2.1	*Rule Condition*	A condition element for the rule. • *Satisfied* – evaluates to True if the *Objective Progress Status* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective*) is True **and** the *Objective Satisfied Status* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective*) is True. • *Objective Status Known* – evaluates to True if the *Objective Progress Status* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective*) is True. • *Objective Measure Known* – evaluates to True if the *Objective Progress Status* for the objective (TM.1.1) associated with the activity (indicated by *Rule Condition Referenced Objective*) is True **and** the *Objective Measure Status* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective)* is True. • *Objective Measure Greater Than* – evaluates to True if the *Objective Measure Status* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective*) is True **and** the *Objective Normalized Measure* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective)* is greater than the *Rule Condition Measure Threshold*. • *Objective Measure Less Than* – evaluates to True if the *Objective Measure Status* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective*) is True **and** the *Objective Normalized Measure* (TM.1.1) for the objective associated with the activity (indicated by *Rule Condition Referenced Objective)* is less than the *Rule Condition Measure Threshold*. • *Completed* – evaluates to True if the *Attempt Progress Status* (TM.1.2.2) for the activity is True **and** the *Attempt Completion Status* (TM.1.2.2) for the activity is True. • *Activity Progress Known* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the activity is True **and** the *Attempt Progress Status* (TM.1.2.2) for the activity is True. • *Attempted* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the activity is True **and** the *Activity Attempt Count* (TM.1.2.1) for the activity is positive (i.e., the activity has been attempted). • *Attempt Limit Exceeded* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the activity is True **and** the *Limit Condition Attempt Limit Control* (SM.3) for the activity is True **and** the *Activity Attempt Count* (TM.1.2.1) for the activity is equal to or greater than the *Limit Condition Attempt Limit* (SM.3) for the activity. • *Time Limit Exceeded* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the activity is True **and** any of the durations for the activity (Activity Absolute Duration, Activity Experienced Duration, Attempt Absolute Duration, Attempt Experienced Duration) exceed the corresponding duration Limit Condition values for the activity (SM.3 – Limit Condition *Activity Absolute Duration, Limit Condition Activity Experienced Duration, Limit Condition Attempt Absolute Duration, Limit Condition Attempt Experienced Duration*) **and** the associated Limit Condition control value (SM.3 – Limit Condition *Activity Absolute Duration Control, Limit Condition Activity Experienced Duration Control, Limit Condition Attempt Absolute Duration Control, Limit Condition Attempt Experienced Duration Control*) for the activity is True. • *Outside Available Time Range* – evaluates to True if the current time is before or after the corresponding time Limit Condition values for the activity (SM.3 – Limit Condition *Begin Time, Limit Condition End Time*) **and** the associated Limit Condition control value (SM.3 – Limit Condition *Begin Time Control, Limit Condition End Time Control*) for the activity is True. • *Always* – always evaluates to True.	Vocabulary	Always
2.2	*Rule Condition Referenced Objective*	The identifier of an objective associated with the activity used during the evaluation of the condition. If a rule for an activity does not explicitly reference an objective by an identifier, the rule references the objective that contributes to rollup for an activity by default.	Unique Identifier[a]	None
2.3	*Rule Condition Measure Threshold*	The value used as a threshold during measure-based condition evaluations.	Real [-1..1] Precision of at least 4 significant decimal digits	0.0
2.4	*Rule Condition Operator*	The unary logical operator to be applied to the condition. • *Not* – The corresponding condition is negated in rule evaluation. • *NO-OP* – The corresponding condition is used as is in rule evaluation.	Vocabulary	NO-OP

No.	Name	Description	Value Space	Default Value
3	*Rule Action*	The desired sequencing behavior if the rule evaluates to True. Actions are divided into groups based on when the action is applied. Precondition actions apply when traversing the activity tree to identify an activity for delivery: • *Skip* – The activity is not considered a candidate for delivery during a *Flow* sequencing process (SB.2.3). • *Disabled* – The activity may not be the target of any sequencing or delivery request. • *Hidden from Choice* – The activity may not be the target of a *Choice* sequencing request. • *Stop Forward Traversal* – Stop walking the activity tree, in a forward direction, at the activity while processing a sequencing request. • *Ignore* – No action. Post Condition actions apply when an attempt on the activity terminates: • *Exit Parent* – Process an *Exit Parent* exit request. • *Exit All* – Process an *Exit All* termination request and return an *Exit* sequencing request. • *Retry* – Return a *Retry* sequencing request. • *Retry All* – Process an *Exit All* termination request and return a *Start* sequencing request. • *Continue* – Return a *Continue* sequencing request. • *Previous* – Return a *Previous* sequencing request. • *Ignore* – No action. Exit actions apply after an attempt on a descendent activity terminates: • *Exit* – Unconditionally terminate the activity. • *Ignore* – No action.	Vocabulary	Ignore

a. This identifier should be globally unique, or at least unique in any scope in which sequenced activities might be used. The IMS Persistent, Location-Independent Resource Identifier Handbook [4] describes the recommended way to construct such an identifier.

SM.3 Limit Conditions Description

Limit conditions description (the set of attributes shown below) define constraints on the access to an activity based on time of day, time spent on the activity and number of attempts. Limit conditions description for an activity includes the associated data listed below.

Simple Sequencing processes may reference the limit conditions description for any activity in the activity tree. Limit Conditions are optional. Limit condition data need not be defined for each activity in the activity tree. Default data is used if the data is not instantiated for the activity, if needed.

No.	Name	Description	Value Space	Default Value
1	*Limit Condition Attempt Control*	Indicates that a limit condition on the number of attempts for the activity has been established (True or False) for the activity. If the value is False, there is no constraint on how many times the activity may be attempted.	Boolean	False
2	*Limit Condition Attempt Limit*	The maximum number of attempts for the activity. A zero value indicates the activity may not be accessed. The value is unreliable unless *Limit Condition Attempt Control* is True.	Non Negative Integer	0
3	*Limit Condition Attempt Absolute Duration Control*	Indicates that a limit condition on the maximum time duration that a learner is permitted to spend on any single attempt on the activity has been established (True or False) for the activity. If the value is False, there is no constraint on how long the learner may spend on the activity.	Boolean	False

No.	Name	Description	Value Space	Default Value
4	*Limit Condition Attempt Absolute Duration Limit*	The maximum time duration that a learner is permitted to spend on any single attempt on the activity. This limit applies to the time the activity is *active* – from the time the activity begins until the time the activity ends, including any time the activity was *suspended*. A zero value indicates the activity may not be accessed. The value is unreliable unless *Limit Condition Attempt Absolute Duration Control* is True.	Duration Accuracy 0.1 second	0.0
5	*Limit Condition Attempt Experienced Duration Control*	Indicates that a limit condition on the maximum time duration that a learner is permitted to spend on any single attempt on the activity has been established (True or False) for the activity. If the value is False, there is no constraint on how long the learner may spend on the activity.	Boolean	False
6	*Limit Condition Attempt Experienced Duration Limit*	The maximum time duration that a learner is permitted to spend experiencing a single attempt on the activity. The limit applies to only the time the learner is actually interacting with the activity and does not apply when the activity is suspended (i.e., when the activity is not being experienced or is inactive). A zero value indicates the activity may not be accessed. The value is unreliable unless *Limit Condition Attempt Experienced Duration Control* is True.	Duration Accuracy 0.1 second	0.0
7	*Limit Condition Activity Absolute Duration Control*	Indicates that a limit condition on the maximum duration that a learner is permitted to spend on the activity (which includes all attempts) has been established (True or False) for the activity. If the value is False, there is no constraint on how long the learner may spend on the activity.	Boolean	False
8	*Limit Condition Activity Absolute Duration Limit*	The maximum time duration that a learner is permitted to spend on all attempts at the activity. A zero value indicates the activity may not be accessed. The value is unreliable unless *Limit Condition Activity Absolute Duration Control* is True.	Duration Accuracy 0.1 second	0.0
9	*Limit Condition Activity Experienced Duration Control*	Indicates that a limit condition on the maximum duration that a learner is permitted to spend on the activity (which includes all attempts) has been established (True or False) for the activity. If the value is False, there is no constraint on how long the learner may spend on the activity.	Boolean	False
10	*Limit Condition Activity Experienced Duration Limit*	The maximum time duration that a learner is permitted to spend on all attempts of the activity. The limit applies to only the time the learner is actually experiencing the activity and does not apply when the activity is suspended (i.e., when the activity is not being experienced or is inactive). A zero value indicates the activity may not be accessed. The value is unreliable unless *Limit Condition Activity Experienced Duration Control* is True.	Duration Accuracy 0.1 second	0.0
11	*Limit Condition Begin Time Limit Control*	Indicates that a limit condition on the availability of the activity has been established (True or False) for the activity. If the value is False, there is no constraint on when the learner may access the activity.	Boolean	False
12	*Limit Condition Begin Time Limit*	The date and time before which the activity is not available. The value is unreliable unless *Limit Condition Begin Time Limit Control* is True.	Time Point Accuracy 0.1 second	October, 15 1582 00:00:00.0

No.	Name	Description	Value Space	Default Value
13	*Limit Condition End Time Limit Control*	Indicates that a limit condition on the availability of the activity has been established (True or False) for the activity. If the value is False, there is no constraint on when the learner may access the activity.	Boolean	False
14	*Limit Condition End Time Limit*	The date and time after which the activity is not available. The value is unreliable unless *Limit Condition End Time Limit Control* is True.	Timepoint Accuracy 0.1 second	October, 15 1582 00:00:00.0

SM.4 Auxiliary Resource Description

Auxiliary resource description (the set of attributes shown below) defines an auxiliary resource associated with an activity. When an activity is delivered to the learner, the auxiliary resources are also made available to the learner. An auxiliary resource description for an activity includes the associated data listed below.

Simple Sequencing processes may reference the auxiliary resource description for any activity in the activity tree. Auxiliary resources are optional. Auxiliary resource data need not be defined for each activity in the activity tree. If an auxiliary resource is instantiated for an activity, it must provide the required data.

The data attributes below describe one auxiliary resource. Each activity may have an unlimited number of unordered auxiliary resources.

Purpose is an open, unspecified vocabulary. The vocabulary tokens have no semantics or meanings themselves. No behaviors are specified for any token value.

No.	Name	Description	Value Space	Default Value
1	*Resource ID*	The identifier of the auxiliary resource.	Unique Identifier[a]	None Value is Required
2	*Purpose*	Indicates the purpose of the auxiliary resource.	Open Vocabulary	None Value is Required

a. This identifier should be globally unique, or at least unique in any scope in which sequenced activities might be used. The IMS Persistent, Location-Independent Resource Identifier Handbook [4] describes the recommended way to construct such an identifier.

SM.5 Rollup Rule Description

Rollup rule description (the set of attributes shown below) specifies the details of individual rule-based rollup behaviors for an activity. Rollup rules describe how values of the child activities influence the Objective Progress Information (TM.1.1) and Activity/Attempt Progress Progress Information (TM.1.2) for an activity. Rollup rule description information for an activity includes the associated data listed below.

Simple Sequencing processes may reference the rollup rules for any activity in the activity tree. Rollup rules are optional. Rollup rules are defined only where needed. Default data is used if the data is not instantiated for a given rule, if needed.

The data attributes describe one rule. Each activity may have an unlimited number of unordered rollup rules.

The general format of a rule can be expressed informally as: *If child-activity set, condition set Then action*.

- There is one child activity set that determines the child activities that are required for the condition to be true.
- Multiple conditions are permitted.

- The conditions may be combined with a single *and* combination (all conditions must be True) or a single *or* combination (only one condition must be True).
- Individual condition values may be negated before being combined in the rule evaluation.
- No other operators on the conditions are defined.
- Each condition references an item in the tracking model for the activity; conditions that reference objectives refer to the objective that has the attribute value *Objective Contributes to Rollup* value of True.
- There is one action that may result if the rollup conditions evaluate to True.
- Actions may result in setting of the *Attempt Completion Status* (TM.1.2.2) or *Objective Satisfied Status* (TM.1.1) (for the objective that has the attribute value *Objective Contributes to Rollup* (SM.6) value of True) for the activity.

The *Rollup Child Activity Set, Rollup Conditions, Condition Combination,* and *Rollup Actions* are tokens in a vocabulary. The tokens have no semantics or meanings themselves. The description of the condition or action is the complete definition of the required behavior.

No.	Name	Description	Value Space	Default Value
1	*Rollup Child Activity Set*	The set of children of the activity whose data values are used to evaluate the rollup condition. • *All* – The rollup rule evaluates to True if and only if all of the children have a rollup condition (the result of the *Condition Combination*) value of True. • *Any* – The rollup rule condition evaluates to True if any of the children have a rollup condition (the result of the *Condition Combination*) value of True. • *None* – The rollup rule condition evaluates to True if none of the children have a rollup condition (the result of the *Condition Combination*) value of True. • *At Least Count* – The rollup rule condition evaluates to True if at least the number of children specified by the *Rollup Minimum Count* attribute have a rollup condition (the result of the *Condition Combination*) value of True. • *At Least Percent* – The rollup rule condition evaluates to True if at least the percentage of children specified in the *Rollup Minimum Percent* attribute have a rollup condition (the result of the *Condition Combination*) value of True.	Vocabulary	All
1.1	*Rollup Minimum Count*	The number of children activities associated with a *Rollup Child Activity Set* attribute value of *At Least Count.* This value is meaningless if the *Rollup Child Activity Set* value is not *At Least Count*.	Non Negative Integer	0
1.2	*Rollup Minimum Percent*	The percentage of children activities associated with a *Rollup Child Activity Set* attribute value of *At Least Percent.* This value is meaningless if the *Rollup Child Activity Set* value is not *At Least Percent*.	Real[a] [0..1] Precision of at least 4 digits	0.0
2	*Condition Combination*	How rollup conditions are combined in evaluating the rule. • *All* – The rule condition evaluates to True if and only if all of the individual rollup conditions evaluate to True (logical **and**). • *Any* – The rule condition evaluations to True if any of the individual rollup conditions evaluates to True (logical **or**).	Vocabulary	Any
3	*Rollup Conditions*	An unordered collection of conditions for a rollup rule. A rollup rule may include multiple conditions.	Collection 0:N	

No.	Name	Description	Value Space	Default Value
3.1	*Rollup Condition*	A condition element for the rule. • *Satisfied* – evaluates to True if the *Objective Progress Status* (TM.1.1) for the rolled up objective associated with the child activity is True **and** *Objective Satisfied Status* (TM.1.1) for the rolled up objective associated with the child activity is True. • *Objective Status Known* – evaluates to True if *Objective Progress Status* (TM.1.1) for the rolled up objective associated with the child activity is True. • *Objective Measure Known* – evaluates to True if the *Objective Measure Status* (TM.1.1) for the rolled up objective associated with the child activity is True. • *Completed* – evaluates to True if the *Attempt Progress Status* (TM.1.2.2) for the child activity is True **and** the *Attempt Completion Status* (TM.1.2.2) for the child activity is True. • *Activity Progress Known* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the child activity is True **and** the *Attempt Progress Status* (TM.1.2.2) for the child activity is True. • *Attempted* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the child activity is True **and** the *Activity Attempt Count* (TM.1.2.1) for the child activity is positive (i.e., the child activity has been attempted). • *Attempt Limit Exceeded* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the child activity is True **and** the *Limit Condition Attempt Limit Control* (SM.3) for the child activity is True **and** the *Activity Attempt Count* (TM.1.2.1) for the child activity is equal to or greater than the *Limit Condition Attempt Limit* (SM.3) for the child activity. • *Time Limit Exceeded* – evaluates to True if the *Activity Progress Status* (TM.1.2.1) for the child activity is True **and** any of the tracked durations for the child activity (Activity Absolute Duration, Activity Experienced Duration, Attempt Absolute Duration, Attempt Experienced Duration) exceed the corresponding duration Limit Condition values for the child activity (SM.3 – Limit Condition *Activity Absolute Duration, Limit Condition Activity Experienced Duration, Limit Condition Attempt Absolute Duration, Limit Condition Attempt Experienced Duration*) **and** the associated Limit Condition control value (SM.3 – Limit Condition *Activity Absolute Duration Control, Limit Condition Activity Experienced Duration Control, Limit Condition Attempt Absolute Duration Control, Limit Condition Attempt Experienced Duration Control*) for the child activity is True. • *Outside Available Time Range* – evaluates to True if the current time is before or after the corresponding time Limit Condition values for the child activity (SM.3 – Limit Condition *Begin Time, Limit ConditionEnd Time*) **and** the associated Limit Condition control value (SM.3 – Limit Condition *Begin Time Control, Limit Condition End Time Control*) for the child activity is True. • *Never* – always evaluates to False.	Vocabulary	Never
3.2	*Rollup Condition Operator*	The unary logical operator to be applied to the condition. • *Not* – The corresponding condition is negated in rule evaluation. • *NO-OP* – The corresponding condition is used as is in rule evaluation.	Vocabulary	NO-OP
4	*Rollup Action*	The desired rollup behavior if the rule evaluates to True. • *Satisfied* – The *Objective Progress Status* (TM.1.1) for the rolled up objective associated with the activity is set to True. The *Objective Satisfied Status*(TM.1.1) for the rolled up objective associated with the activity is set to True. • *Not Satisfied* – The *Objective Progress Status* (TM.1.1) for the rolled up objective associated with the activity is set to True. The *Objective Satisfied Status* (TM.1.1) for the rolled up objective associated with the activity is set to False. • *Completed* – The *Attempt Progress Status* (TM.1.2.2) for the activity is set to True. The *Attempt Completion Status* (TM.1.2.2) for the activity is set to True. • *Incomplete* – The *Attempt Progress Status* (TM.1.2.2) for the activity is set to True. The *Attempt Completion Status* (TM.1.2.2) for the activity is set to False.	Vocabulary	Satisfied

a. The value is normalized between 0 and 1.

SM.6 Objective Description

The objective description (the set of attributes shown below) defines the learning objective(s) associated with an activity.

Each activity may have an unlimited number of learning objectives. The tracking model (see Tracking Model **TM**) defines a set of data that records the satisfaction status (e.g., passed/failed) and measure (e.g., score) for each objective, for each attempt on the activity. The meaning of a learning objective is not defined in this model; it is defined only in terms of its ID and its association with an activity.

Simple Sequencing processes may reference the local objective information for any activity. The use of Objective Maps (described in SM.7) allows Simple Sequencing processes to also reference objective information for globally shared objectives. An activity will have local objective information for each objective associated with the activity. An activity must have at least one objective; if one is not defined in the sequencing description associated with the activity, one will be instantiated for the purposes of sequencing. Each activity must have one and only one objective that contributes to rollup; if one is not defined in the sequencing description associated with the activity, one will be instantiated for the purposes of sequencing.

The *Objective ID* is required if the activity has more than one objective or if the objective information for the activity is to be shared with another activity.

No.	Name	Description	Value Space	Default Value
1	*Objective ID*	The identifier of an objective associated with the activity. The ID is a link to the corresponding objective information.	Unique Identifier[a]	NoneRequired Value
2	*Objective Satisfied by Measure*	Indicates that the *Objective Minimum Satisfied Normalized Measure* is to be used (True or False) in place of any other method to determine if the objective associated with the activity has been satisfied.	Boolean	False
3	*Objective Minimum Satisfied Normalized Measure*	The minimum satisfaction measure for the objective, normalized between -1..1 (inclusive). If the *Objective Measure Status* (TM.1.1) for the objective is True **and** the *Objective Normalized Measure* (TM.1.1) for the objective exceeds this value, the *Objective Progress Status* (TM.1.1) is set to True and the *Objective Satisfied Status* (TM.1.1) is set to True. If the *Objective Measure Status* (TM.1.1) for the objective is True **and** the *Objective Normalized Measure* (TM.1.1) for the objective is less than this value, the *Objective Progress Status* (TM.1.1) is set to True and the *Objective Satisfied Status* (TM.1.1) is set to False. The value is unreliable unless *Objective Satisfied by Measure* is True.	Real [-1..1] Precision of at least 4 significant decimal digits	1.0
4	*Objective Contributes to Rollup*	Indicates that the *Objective Satisfied Status* (TM.1.1) and *Objective Normalized Measure* (TM.1.1) for the objective are used (True or False) during rollup.	Boolean	False

a. This identifier should be globally unique, or at least unique in any scope in which sequenced activities might be used. The IMS Persistent, Location-Independent Resource Identifier Handbook [4] describes the recommended way to construct such an identifier.

SM.7 Objective Map

The objective map description (the set of attributes shown below) defines a mapping of an activity's local objective information to and from a shared global objective.

Each activity may have an unlimited number of objective maps.

By default, no objective information is shared between activities. If objective mapping is desired, each activity must define a set of *Objective Maps* to describe how local objective information is mapped to shared global objectives. The *Objective Map* data is evaluated whenever local objective information is processed, as described in the tracking model (see Tracking Model **TM**) behaviors. For any given local objective, a 'read' map with at most one global objective may be defined. Also, for any global objective, for any activity, a 'write' map with at most one local objective can be defined.

Simple Sequencing processes may reference the objective map data for any activity. Objective mapping does not occur if objective map data is not defined. Default data is used if the data is not instantiated for the activity, if needed.

No.	Name	Description	Value Space	Default Value
1	*Activity Objective ID*	The identifier of the local objective associated with the activity.	Unique Identifier[a]	None Value is Required
2	*Target Objective ID*	The identifier of global shared objective targeted for the mapping.	Unique Identifier[b]	None Value is Required
3	*Read Objective Satisfied Status*	Indicates that the *Objective Satisfied Status* (TM.1.1) for the identified local objective (*Activity Objective ID*), should be retrieved (True or False) from the identified shared global objective (*Target Objective ID*), when the progress for the local objective is undefined – *Objective Progress Status* (TM.1.1) for the identified local objective (*Activity Objective ID*) is False. This operation does not change the Objective Information associated with the local objective.	Boolean	True
4	*Write Objective Satisfied Status*	Indicates that the *Objective Progress Status* (TM.1.1) and *Objective Satisfied Status* (TM.1.1) values, for the identified local objective (*Activity Objective ID*), should be transferred (True or False) to the identified global shared objective (*Target Objective ID*), upon termination of an attempt on the activity.	Boolean	False
5	*Read Objective Normalized Measure*	Indicates that the *Objective Normalized Measure* (TM.1.1) for the identified local objective (*Activity Objective ID*), should be retrieved (True or False) from the identified shared global objective (*Target Objective ID*), when the measure for the local objective is undefined – *Objective Measure Status* (TM.1.1) for the identified local objective (*Activity Objective ID*) is False. This operation does not change the Objective Information associated with the local objective.	Boolean	True
6	*Write Objective Normalized Measure*	Indicates that the *Objective Measure Status* (TM.1.1) and *Objective Normalized Measure* (TM.1.1) values, for the identified local objective (*Activity Objective ID*), should be transferred (True or False) to the identified global shared objective (*Target Objective ID*), upon termination of an attempt on the activity.	Boolean	False

a. This identifier should be globally unique, or at least unique in any scope in which sequenced activities might be used. The IMS Persistent, Location-Independent Resource Identifier Handbook [4] describes the recommended way to construct such an identifier.

b. This identifier should be globally unique, or at least unique in any scope in which sequenced activities might be used. The IMS Persistent, Location-Independent Resource Identifier Handbook [4] describes the recommended way to construct such an identifier.

SM.8 Rollup Controls

Rollup controls (the set of attributes shown below) include descriptions of the types of rollup behaviors specified for an activity. Rollup controls for an activity include the associated data listed below.

Simple Sequencing processes may reference the rollup control data for any activity in the activity tree. Rollup control data need not be defined for each activity in the activity tree. Default data is used if the data is not instantiated for the activity, if needed.

No.	Name	Description	Value Space	Default Value
1	*Rollup Objective Satisfied*	Indicates that *Objective Satisfied Status* (TM.1.1) for the objective (which has *Objective Contributes to Rollup* (SM.6) equal to *True*) associated with the activity is included (True or False) in the rollup for its parent activity.	Boolean	True

No.	Name	Description	Value Space	Default Value
2	*Rollup Objective Measure Weight*	A weighting factor applied to the *Objective Normalized Measure* (TM.1.1) for the objective (which has *Objective Contributes to Rollup* (SM.6) equal to *True*) associated with the activity used during rollup for its parent activity.	Real [0..1] Precision of at least 4 significant decimal digits	1.0
3	*Rollup Progress Completion*	Indicates that the *Attempt Completion Status* (TM.1.2.2) value for the activity is included (True or False) in the rollup for its parent activity.	Boolean	True

SM.9 Selection Controls

Selection controls (the set of attributes shown below) include descriptions of how the children of an activity should be selected during the sequencing process. Selection controls for an activity include the associated data listed below.

Simple Sequencing processes may reference the selection control data for any activity in the activity tree. Selection control data is optional. Selection control data need not be defined for each activity in the activity tree. Default data is used if the data is not instantiated for the activity, if needed.

No.	Name	Description	Value Space	Default Value
1	*Selection Timing*	Indicates when selection should occur. • *Never* – Selection is never applied; all of the children of the activity are selected by default. • *Once* – Selection is applied before the first attempt on an activity. • *On Each New Attempt* – Selection is applied before each new attempt on an activity. The *On Each New Attempt* option and its associated behavior is not specified in this version of the Simple Sequencing Specification[a].	Vocabulary	Never
2	*Selection Count Status*	Indicates the selection count data is (True or False) meaningful for the activity.	Boolean	False
3	*Selection Count*	Indicates the number of child activities that must be selected from the set of child activities associated with the activity. If *Selection Count* is larger than the number of child activities, all child activities are selected. This value is unreliable unless *Selection Count Status* is *True*. If *Selection Count Status* is *False* all child activities are selected.	Non Negative Integer	0

a. Although the selection of child activities 'on each new attempt' is a desired feature, when to perform the operation is not well defined.

SM.10 Randomization Controls

Randomization controls (the set of attributes shown below) include descriptions of how the children of an activity should be ordered during the sequencing process. Randomization controls for an activity include the associated data listed below.

Simple Sequencing processes may reference the randomization control data for any activity in the activity tree. Randomization control data is optional. Randomization control data need not be instantiated for each activity in the activity tree. Default data is used if the data is not instantiated for the activity.

No.	Name	Description	Value Space	Default Value
1	Randomization Timing	Indicates when the ordering of the children of the activity should occur. • *Never* – Randomization is never applied. • *Once* – Randomization is applied before the first attempt on the activity. • *On Each New Attempt* – Randomization is applied before each new attempt on the activity.	Vocabulary	Never

No.	Name	Description	Value Space	Default Value
2	Randomize Children	Indicates that the order of the child activities is randomized (True or False).	Boolean	False

SM.11 Delivery Controls

Delivery controls (the set of attributes shown below) describe actions and controls used when an activity is delivered, i.e., Objective, Activity, and Attempt Progress Data are recorded when the activity is delivered. Delivery controls for an activity include the associated data listed below.

Simple Sequencing processes may reference the delivery control data for any activity in the activity tree. Delivery controls are optional. Delivery data need not be defined for each activity in the activity tree. Default data is used if the data is not instantiated for the activity, if needed.

No.	Name	Description	Value Space	Default Value
1	*Tracked*	Indicates that Objective Progress Information and Activity/Attempt Progress Information (TM) for the attempt should be recorded (True or False) and the data will contribute to the rollup for its parent activity. How the data is tracked and recorded is not specified.	Boolean	True
2	*Completion Set by Content*	Indicates that the *Attempt Completion Status* (TM.1.2.2) for the activity will be set by the content object (True or False). A False value indicates that default rules will be used to set progress data. For a True value, how, if or when the completion data is set is not specified.	Boolean	False
3	*Objective Set by Content*	Indicates that the *Objective Satisfied Status* (TM.1.1) for the activity's associated objective that has the *Objective Contributes to Rollup* (SM.6) value of True will be set by the content object (True or False). A False value indicates that default rules will be used to set objective information. For a True value, how, if or when the objective information is set is not specified.	Boolean	False

SM.12 Sequencing Description

The complete set of sequencing information (all of the items described above) is associated with each activity in the activity tree. The table lists all of the elements in the overall sequencing description and their multiplicities.

The sequencing description model describes the data that specifies the intended sequencing behavior. This data is used by a system that delivers sequenced activities. How this information is encoded, stored, represented, or bound is outside the scope of this specification. The overall sequencing definition model only describes a set of related data items and internal constraints on the values of those items.

There is no requirement that the value for any specific sequencing definition model data item exist. There is a defined default or initial value for most attributes. When a data item does not exist and its attributes are required for some process, the default value is supplied upon reference if the data value does not exist.

The mechanisms used to create or maintain the sequencing definition model data are not specified as part of the information model.

An implementation must be capable of representing the range of values described. There are no additional requirements on implementing the sequencing definition model.

The sequencing behavior model, navigation behavior model, termination behavior model, rollup behavior model, selection and randomization behavior model and delivery behavior model (see Sequencing Behavior Model **SB**, Navigation Behavior Model **NB**, Termination Behavior Model **TB**, Rollup Behavior Model **RB**, Selection and Randomization Behavior Model **SR**, Delivery Behavior Model **DB**) describe how a sequencing system uses a sequencing definition instance for an activity tree to control the sequencing and delivery of activities.

No.	**Name**	**Description**	**Value Space**	**Default Value**
1	*Sequencing Control Modes*	Describes the types of sequencing behaviors specified for an activity.	Sequencing Control Mode	0..1
2	*Sequencing Rules*	Describes the details of individual rule-based sequencing behaviors for an activity.	Ordered set of *Sequencing Rule Description*	0..N
3	*Limit Conditions*	Describes the constraints on the access to an activity based on time of day, time spent on the activity and number of attempts.	Limit Conditions Description	0..1
4	*Auxiliary Resources*	Describes the auxiliary resources available while the activity is active.	Auxiliary Resource Description	0..N
5	*Rollup Rules*	Describes the details of individual rule-based rollup behaviors for an activity.	Rollup Rule Description	0..N
6	*Objectives*	Describes an unordered collection of objectives associated with an activity.	Objective Description	0..N
7	*Objective Maps*	Describes mappings of local objective information for an activity to a global shared objective.	Objective Map	0..N
8	*Rollup Controls*	Describes the types of rollup behaviors specified for an activity.	Rollup Controls	0..1
9	*Selection Controls*	Describes how the children of an activity should be selected during the sequencing process.	Selection Controls	0..1
10	*Randomization Controls*	Describes how the children of an activity should be ordered during the sequencing process.	Randomizatio n Controls	0..1
11	*Delivery Controls*	Describes actions and controls used when an activity is delivered.	Delivery Controls	0..1

TM. Tracking Model

Simple Sequencing processes uses information about the results of a learner's interactions with activities, and the learner's record for objectives (e.g., completion, measure) to control the selection and sequencing of other activities. The sequencing behaviors are defined in terms of a limited set of specific data attributes that describe the results of the learner's interactions. The defined set of attributes used by Simple Sequencing is called the "tracking model".

The tracking model consists of:

- An information model, defined in two parts:
  - *Objective Progress Information* – information about the results of the learner's interactions related to an *objective*.
  - *Activity / Attempt Progress Information* – information about a learner's *attempts* at an activity.
- Behaviors – requirements on instantiation and use of the tracking information model.

TM.1 Tracking Information Model

The tracking information model describes the data used by a system that delivers sequenced activities. How this information is encoded, stored, represented, or bound is outside the scope of this specification. The tracking information model only describes a set of related data items and internal constraints on the values of those items.

An implementation must be capable of representing the range of values described. There are no additional requirements on implementing the information model.

The tracking model behaviors description states requirements for the instantiation and use of the information model.

TM.1.1 Objective Progress Information

Objective progress information includes results of the learner's interactions related to an objective. Tracking information for an objective includes the associated data (i.e., the objective information) listed below. How the tracking information is associated with an objective is not specified.

Simple Sequencing descriptions may reference objective information for any local objective associated with any activity in the activity tree, or any shared global objective. Objective progress information should be instantiated for objectives referenced in the activity tree for each learner as appropriate.

No.	Name	Description	Value Space	Default Value
1	*Objective Progress Status*	Indicates the objective has a satisfaction value (True or False).	Boolean	False
2	*Objective Satisfied Status*	Indicates the objective is satisfied (True or False). The determination or meaning of satisfied or not satisfied is not defined in this model. The value is unreliable unless *Objective Progress Status* is True.	Boolean	False
3	*Objective Measure Status*	Indicates the objective has a measure value (True or False).	Boolean	False
4	*Objective Normalized Measure*	The measure (e.g., score) for the objective, normalized between -1..1 (inclusive). The mechanism to normalize a measure is not defined in this model. The value is unreliable unless *Objective Measure Status* is True.	Real [-1..1] Precision of at least 4 significant decimal digits	0.0

TM.1.2 Activity/Attempt Progress Information

TM. 1.2.1 Activity Progress Information

Activity progress information (the set of attributes shown below) describes a learner’s progress on an activity. This information describes the cumulative progress information for an individual activity. Tracking information for an activity includes the data listed below.

Simple Sequencing descriptions may reference activity progress information for any activity in the activity tree. Activity progress information should be instantiated for each activity in the activity tree for each learner. The mechanism for determining an activity’s duration is not defined in this model. How the activity progress information is associated with an activity is not specified.

No.	Name	Description	Value Space	Default Value
1	*Activity Progress Status*	Indicates the activity progress information is (True or False) meaningful for the activity.	Boolean	False
2	*Activity Absolute Duration*	The cumulative duration of all attempts on the activity, i.e., the time from the initial start of the activity to the end of the activity. The mechanism for determining the duration is not defined in this model. The value is unreliable unless *Activity Progress Status* is True.	Duration Accuracy 0.1 second	0.0
3	*Activity Experienced Duration*	The cumulative *experienced* duration of all attempts on the activity, i.e., the time from the initial start of the activity to the end of the activity, not including any time elapsed while the activity is suspended (i.e., when the activity is not being experienced or is inactive). The mechanism for determining the duration of the suspend time is not defined in this model. The value is unreliable unless *Activity Progress Status* is True and the activity is a leaf.	Duration Accuracy 0.1 second	0.0
4	*Activity Attempt Count*	The number of attempts on the activity. The count includes the current attempt, i.e., 0 means the activity was not attempted and 1 or greater means it either is in progress or completed. The value is unreliable unless *Activity Progress Status* is True.	Non Negative Integer	0

TM. 1.2.2 Attempt Progress Information

Attempt progress information (the set of attributes shown below) describes a learner’s progress for a unique attempt on an activity. This information describes the progress information for one attempt on an individual activity. Tracking information for an attempt on an activity includes the data listed below.

Simple Sequencing descriptions may reference attempt progress for any activity in the activity tree. Attempt progress information should be instantiated for each new attempt for each activity in the activity tree for each learner. The mechanism for determining an activity’s duration is not defined in this model. How the attempt progress information is associated with an activity is not specified.

No.	Name	Description	Value Space	Default Value
1	*Attempt Progress Status*	Indicates the attempt progress information (True or False) is meaningful for the activity attempt. The value is unreliable unless *Activity Attempt Count* (TM.1.2.1) is greater than (>) 0.	Boolean	False
2	*Attempt Completion Amount*[a]	The measure of the completion of the attempt on the activity, normalized between 0..1 (inclusive) where 1 means the activity attempt is complete and any lesser value means the activity attempt is not complete. The mechanism to define the completion amount is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Real [0..1] Precision of at least 4 significant decimal digits	0.0

No.	Name	Description	Value Space	Default Value
3	*Attempt Completion Status*	Indicates the activity attempt is completed (True or False). The determination or meaning of completed or incomplete is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Boolean	False
4	*Attempt Absolute Duration*	The duration of the attempt on the activity, i.e., time from the start of the attempt to the end of the attempt. The mechanism for determining the duration is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Duration Accuracy 0.1 second	0.0
5	*Attempt Experienced Duration*	The *experienced* duration of the attempt on the activity, i.e., the time from the start of the attempt to the end of the attempt, not including elapsed time while the activity attempt is suspended (i.e., when the activity attempt is not being experienced or is inactive). The mechanism for determining the duration or the suspend time is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Duration Accuracy 0.1 second	0.0

a. In SS v1.0, Attempt Completion Amount cannot be evaluated as a rule condition and is not managed by the sequencing engine.

TM.2 Tracking Model Behaviors

The objective progress information applies to an objective. It should be instantiated for each local objective for each learner, and should be initialized for each new attempt on each activity. It should be instantiated once for each shared global objective for each learner.

The activity progress information is for an activity. It should be instantiated for each activity for each learner.

The attempt progress information is for an attempt. It should be instantiated for each learner, and should be initialized for each new attempt on each activity

There is no requirement that the value for any specific tracking data item exist. There is a defined default or initial value for each data item. The default value shall be supplied upon reference if the data item does not exist.

Activity and Attempt durations (absolute and experienced) should be maintained by the LTS. The mechanisms or processes for managing durations is not defined in this document. The only requirements for duration values are:

- The *Activity Absolute Duration* (TM.1.2.1) shall equal the sum of all of the activity's *Attempt Absolute Durations* – any absolute duration will occur during some attempt.
- The *Activity Absolute Duration* (TM.1.2.1) of a parent activity shall equal the sum of all of its children's *Activity Absolute Durations*. See Figure TM.1.

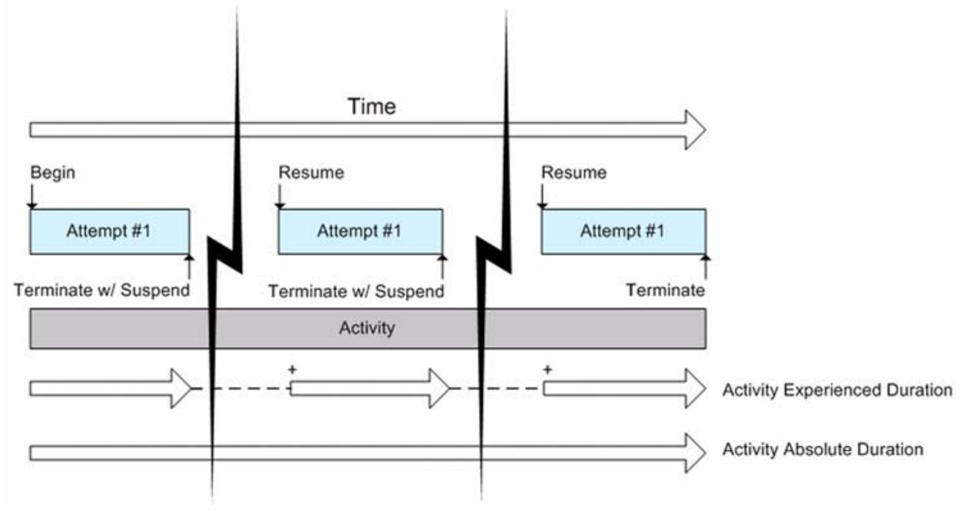

Figure TM.1 - Activity Duration Tracking.

Figure TM.1 illustrates how *Activity E*xperienced Duration and *Activity Absolute Duration* are tracked. It shows the tracking of one attempt on one activity. An attempt on an activity begins; is suspended; and is resumed twice, finally terminating without a suspend. Absolute duration is accumulated from the time the activity begins until it exits. Experienced duration is accumulated from when each attempt on the activity begins until when it is suspended. Accumulation resumes when the attempt is resumed, and continues until it is suspended. Accumulation again resumes when the attempt is again resumed, and continues until the activity terminates.

As only one attempt on the activity is shown, the attempt durations (experienced and absolute) are the same as the activity durations (experienced and absolute).

The timeline is independent of the state of the sequencing session. The attempts may occur across multiple sequencing sessions.

The objective ID and an associated activity identifier are required to reference a specific objective. If an activity only has one local objective and that objective is not shared with any other activity, only the activity identifier is required.

When a local objective's data is undefined (either *Objective Progress Status* (TM.1.1) or *Objective Measure Status* (TM.1.1) are False) and a 'read' *Objective Map* (SM.7) is associated with the objective, the corresponding shared global objective information is used. This is an access-only operation and does not affect the local objective information for the activity. The reference to a global objective only uses the objective ID to locate the actual objective information.

Simple Sequencing processes access local objective information and the attempt progress information only for the most recent attempt at an activity by a learner. If the learner is interacting with the activity, for the purposes of the tracking model, the most recent attempt is the current interaction or current attempt. Otherwise, the most recent attempt is the last fully completed interaction with the activity (if there was one), i.e., the last completed attempt.

Accessing tracking information for the most recent attempt at an activity is controlled by the *Use Current Attempt Objective Information* (SM.1) and *Use Current Attempt Progress* Information (SM.1) elements. If either of these elements for an activity is *True,* the tracking model will not maintain the most recent attempt tracking data (although an implementation is free to retain the historical tracking information) for any of the activity's children, unless those attempts occurred while the current active attempt on the activity is in progress, i.e., the attempt on the child activity occurred while the current attempt on its parent activity was active.

There is no requirement that the attempt progress information be maintained for any attempt other than the most recent attempt. Simple Sequencing makes no requirement to maintain prior history for objective information or overall activity information.

There are no requirements on how or when the values of the tracking data are set and updated for the learner. Simple Sequencing relies only on the “current” values in the information model. Generally, learner interactions with an activity will result in appropriate changes to the tracking data, but these mechanics are not specified.

The overall rollup process (see Rollup Behavior **RB**) defines how tracking data for one activity contributes to the tracking data for ancestors of that activity. How the rollup process uses the tracking data is not specified here but is detailed in the description of the behavior of the rollup process.

The termination, sequencing, and delivery processes (see **TB**, Sequencing Behavior **SB**, and Delivery Behavior **DB**) use the tracking data for elements of the activity tree to determine what happens when an attempt on an activity ends and how subsequent activities are sequenced and delivered to the learner. How the processes use the tracking data are not specified here but are detailed in the descriptions of the behavior of these processes.

The mechanisms used to record and set the tracking data are not specified as part of the information model.

AM. Activity State Model

Simple Sequencing processes uses information about the state or status of the learner's interactions with activities. These attributes are used to control the overall sequencing process, but are not referenced in any specific sequencing definition or sequencing rule. The defined set of state attributes used by Simple Sequencing is called the "activity state model".

The activity state model consists of:

- An information model, defined in two parts:
  - *Activity State Information* – information that describes the state of the most recent attempt at an activity.
  - *Global State Information* – information that describes the overall state of sequencing.
- Behaviors – requirements on instantiation and use of the activity state model.

AM.1 Activity State Information Model

The activity state information model describes the data used by a system that delivers sequenced activities. How this information is encoded, stored, represented, or bound is outside the scope of this specification. The state information model only describes a set of related data items and internal constraints on the values of those items.

An implementation must be capable of representing the range of values described. There are no additional requirements on implementing the activity state information model.

The activity state model behaviors description gives requirements for the instantiation and use of the activity state information model.

AM.1.1 Activity State Information

Activity state information (the set of attributes shown below) describes a learner's state or status for an activity.

Simple Sequencing descriptions and behaviors may reference activity state data for any activity in the activity tree. Activity state data should be instantiated for each activity in the activity tree for each learner. How the activity state information is associated with an activity is not specified.

The activity state values of *Activity is Active* and *Activity is Suspended* cannot both be True at the same time.

No.	Name	Description	Value Space	Default Value
1	*Activity is Active*	Indicates that an attempt is currently in progress for the activity, i.e. the activity has been delivered to the learner and has not been terminated, or the activity is an ancestor of the *Current Activity (AM.1.2) (True or False).*	Boolean	False
2	*Activity is Suspended*	Indicates the activity is currently suspended (True or False).	Boolean	False
3	*Available Children*	A list indicating the ordering of the available child activities for the activity.	Ordered List of Activities	All children

AM.1.2 Global State Information

Global state information (the set of attributes shown below) describes a learner's state or status within the overall sequenced learning experience.

Global state information should be instantiated once for an activity tree for each learner. How the global state information is associated with an activity tree is not specified.

No.	**Name**	**Description**	**Value Space**	**Default Value**
1	*Current Activity*	Indicates the current activity. If an activity is being experienced by the learner, the current activity is the activity delivered by the most recently completed *Content Delivery Environment Process* (see Delivery Behavior **DB**). If an activity is not being experienced by the learner, the current activity is the last activity identified to terminate by the most recently completed *Termination Request Process* (see Termination Behavior **TB**), or the undeliverable target activity of the last successful *Choice Sequencing Request Process* (see Sequencing Behavior SB)	Activity	None
2	*Suspended Activity*	Indicates the activity from which a *Suspend All* navigation request was triggered.	Activity	None

AM.2 Activity State Model Behaviors

The activity state data is for an activity. It should be instantiated for each activity for each learner.

The global state data is for the overall sequencing process. It should be instantiated for the entire sequencing process for each learner.

There is no requirement that the value for any activity state or global state data item exist. There is a defined default or initial value for each data item. The default value shall be supplied upon reference if the data item does not exist.

The mechanisms used to record and set the activity state data are not specified as part of the information model.

The navigation, termination, rollup, selection and randomization, sequencing, and delivery processes (see Navigation Behavior **NB**, Termination Behavior **TB**, Selection and Randomization Behavior **SR**, Sequencing Behavior **SB**, Rollup Behavior **RB**, Delivery Behavior **DB**) use the activity data for elements of the activity tree and the global state data to determine the activities that are sequenced and delivered to the learner. How these processes use the activity and global state data are not specified here but are detailed in the descriptions of the behavior of these processes.

Although the detailed behavior for changing the *Current Activity* is described in the termination, sequencing, and delivery processes (see Termination Behavior **TB**, Sequencing Behavior **SB**, and Delivery Behavior **DB**) that follow, the state transition diagram in Figure AM.1 for the *Current Activity* is intended to provide a summary of the behaviors.

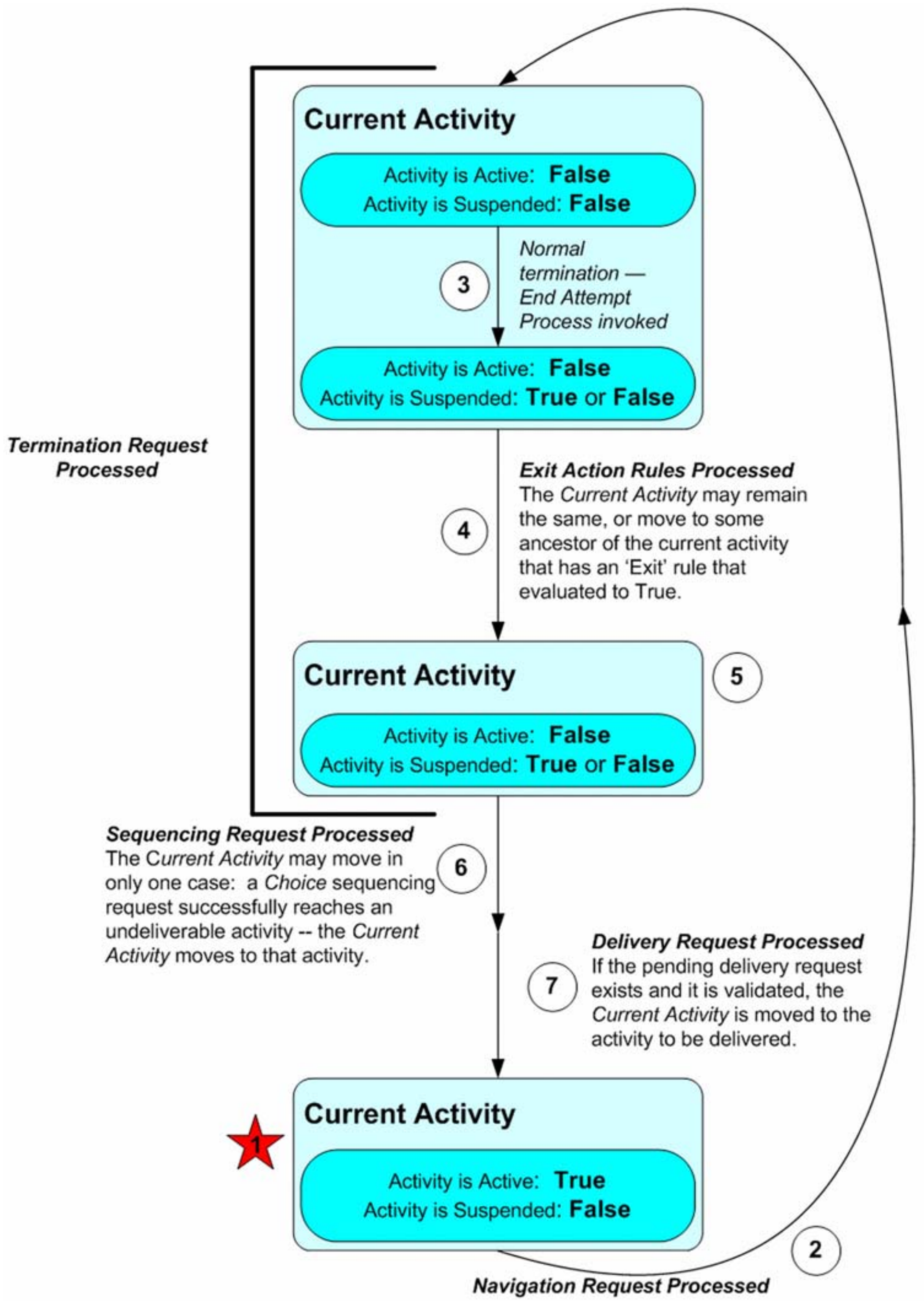
Current Activity
Activity is Active: False
Activity is Suspended: False
3
Normal termination — End Attempt Process invoked
Activity is Active: False
Activity is Suspended: True or False
Termination Request Processed
Exit Action Rules Processed
The Current Activity may remain the same, or move to some ancestor of the current activity that has an 'Exit' rule that evaluated to True.
4
Current Activity
5
Activity is Active: False
Activity is Suspended: True or False
Sequencing Request Processed
The Current Activity may move in only one case: a Choice sequencing request successfully reaches an undeliverable activity -- the Current Activity moves to that activity.
6
Delivery Request Processed
If the pending delivery request exists and it is validated, the Current Activity is moved to the activity to be delivered.
7
Current Activity
1
Activity is Active: True
Activity is Suspended: False
2
Navigation Request Processed

Figure AM.1 - The Current Activity State Transition Diagram.

Figure AM.1 depicts the state transitions of the *Current Activity*; it summarizes the effects of the various sequencing processes on the *Current Activity* and the activity's state.

Begin Loop:

Starting at the star, the sequencer assumes that a content resource has been delivered to the learner; the resource's associated activity is the *Current Activity* (AM.1.2) – it must be leaf of the activity tree and it is currently active. All ancestors (along the 'active path') of the *Current Activity* are also active, because attempts on an activity only occur within the context of attempts on the activity's ancestors.

The state of the activity tree remains like this until some navigation request triggers the Overall Sequencing Process (see Overall Sequencing Process **OP**). If the navigation request is valid, it will result in a termination request to end the attempt on the *Current Activity* – its Activity State attribute of *Activity is Active* becomes false. The *Current Activity* remains the leaf activity.

When an attempt on a leaf activity ends, the system may indicate the state of the termination was to suspend the activity – this is done in the End Attempt Process (UP.4). If the system indicates this, the *Activity is Suspended* (AM.1.1) for the leaf activity (still the *Current Activity*) becomes true.

During Termination Behavior, Exit Action Rules (see *Sequencing Exit Action Rules Subprocess* TB.2.1) are evaluated on all of the ancestors of the *Current Activity*. The result of this subprocess will be that either the leaf activity remains the *Current Activity*, or an ancestor of the leaf activity becomes the *Current Activity.*

During the Termination Behavior, post condition rules are evaluated (see *Sequencing Post Condition Rules Subprocess* TB.2.2) only on the *Current Activity*, which is either the leaf activity or the activity identified during the *Sequencing Exit Action Rule Subprocess*, if that *Current Activity* is not suspended.

The Sequencing Behavior processes any pending sequencing request, which may result in a delivery request. It is possible, during the evaluation of a *Choice* sequencing request (see *Choice Sequencing Request Process* SB.2.9), the the process succeeds in reaching the target activity, but the activity is not deliverable, e.g., the activity is not a leaf. If this happens, the *Current Activity* is moved to the target of the successful *Choice* sequencing request and any intermediate activities are terminated.

The Delivery Behavior validates any pending delivery request. If the delivery request is validated, the *Content Delivery Environment Process* (DB.2) is invoked. During this process, the activity identified by the delivery request becomes the *Current Activity* and an attempt is started (or resumed) on it – the *Current Activity* has *Activity is Active* equal to true and *Activity is Suspended* equal to false.

Repeat Loop

OP. Overall Sequencing Process

The overall sequencing process combines all of the other elements of Simple Sequencing to produce the complete sequencing, delivery and control of learning experience and relates all of the other Simple Sequencing processes and behaviors.

- *Navigation Behavior Model* – The Simple Sequencing process that evaluates a navigation request and determines the exit and sequencing requests that should be processed to identify and deliver content to the learner (see Navigation Behavior **NB**).
- *Termination Behavior Model* – The Simple Sequencing process that processes a termination request to terminate or suspend an activity or set of activities (see Termination Behavior **TB**).
- *Selection and Randomization Behavior Model* – The Simple Sequencing process that selects and reorders activities before being used in sequencing (see Selection and Randomization **SR**).
- *Sequencing Behavior Model* – The Simple Sequencing process that evaluates a sequencing request in terms of the content model described by the activity tree and determines the actual content object that should be delivered to the learner (see Sequencing Behavior **SB**).
- *Delivery Behavior Model* – The Simple Sequencing process that validates that the content resources for the identified activity may be delivered, i.e., all of the conditions that apply to the delivery of the content for the activity and attempt still hold (see Delivery Behavior **DB**).

The overall sequencing process describes only a logical workflow. It makes no assumption on how or when the various steps of the process are invoked, or in what order they are processed.

The behavior of the sequencing process is defined in terms of different processes:

- *Navigation Request Process* – processes the navigation request to identify the corresponding sequencing and termination requests.
- *Termination Request Process* – processes the termination request, sets the corresponding activity state, evaluates exit and post condition rules, and may identify a sequencing request.
- *Selection and Randomization Processes* – select and order activities as defined prior to processing a sequencing request.
- *Sequence Request Process* – processes the sequencing request that controls all sequencing behavior, combining:
    - *Start Sequencing Request Process* – processes a *Start* sequencing request.
    - *Resume All Sequencing Request Process* – processes a *Resume All* sequencing request.
    - *Continue Sequencing Request Process* – processes a *Continue* sequencing request.
    - *Previous Sequencing Request Process* – processes a *Previous* sequencing request.
    - *Choice Sequencing Request Process* – processes a *Choice* sequencing request.
    - *Retry Sequencing Request Process* – processes a *Retry* sequencing request.
    - *Exit Sequencing Request Process* – processes an *Exit* sequencing request.
- *Delivery Request Process* – processes a delivery request.
- *Content Delivery Environment Process* – manages the actual delivery of content resources to the learner.

OP.1 Overall Behavior

The overall behavior functions within the scope of some Learning Technology System (LTS), e.g., an LMS, that has identified learners and internal representations of content activity trees, sequencing descriptions, content resources, and information models used to track the learner and record state. How these actual information models are implemented or initialized is not specified.

An implementation must be capable of representing the processes described and have the implemented processes exhibit the behavior described. There are no additional requirements on implementing the overall sequencing behavior.

When the learner initiates a learning experience, the LTS will generate either a *Start* or *Resume All* navigation request. Other interactions with the LTS will result in other navigation requests. How or when a navigation request is generated is not specified.

The sequencing system operates in a loop, awaiting external navigation requests, processing these, possibly delivering content, and then waiting for additional requests.

The sequencing system continues operations until terminated by the controlling LTS.

The navigation requests are translated by the navigation process into termination and sequencing requests. If the learner is experiencing an activity, the navigation request results in the current activity being terminated by the termination process when it processes the termination request – only one activity may be active. Sequencing rules in the termination process may override the sequencing request. The rollup process is performed to propagate results from an activity to its ancestor activities. The sequencing request is processed by the sequencing process to determine the next activity in the sequence, i.e., the next activity to present to the learner. If the sequencing process identifies an activity, the designated activity is passed to the delivery process. The delivery process will verify that the activity should be delivered, i.e., that the limit conditions and sequencing rules applied to the activity permit it to be delivered. If the delivery process identifies an activity for delivery, the auxiliary resource descriptions are provided to the LTS, content resources associated with the identified activity are presented to the learner, and the learner's interactions with the content may be tracked by the LTS (how this tracking is done is not specified in Simple Sequencing). Once the content has been delivered to the learner, the *Overall Sequencing Process* waits for the next navigation request.

Figure OP.1 depicts the process flow of the *Overall Sequencing Process*.

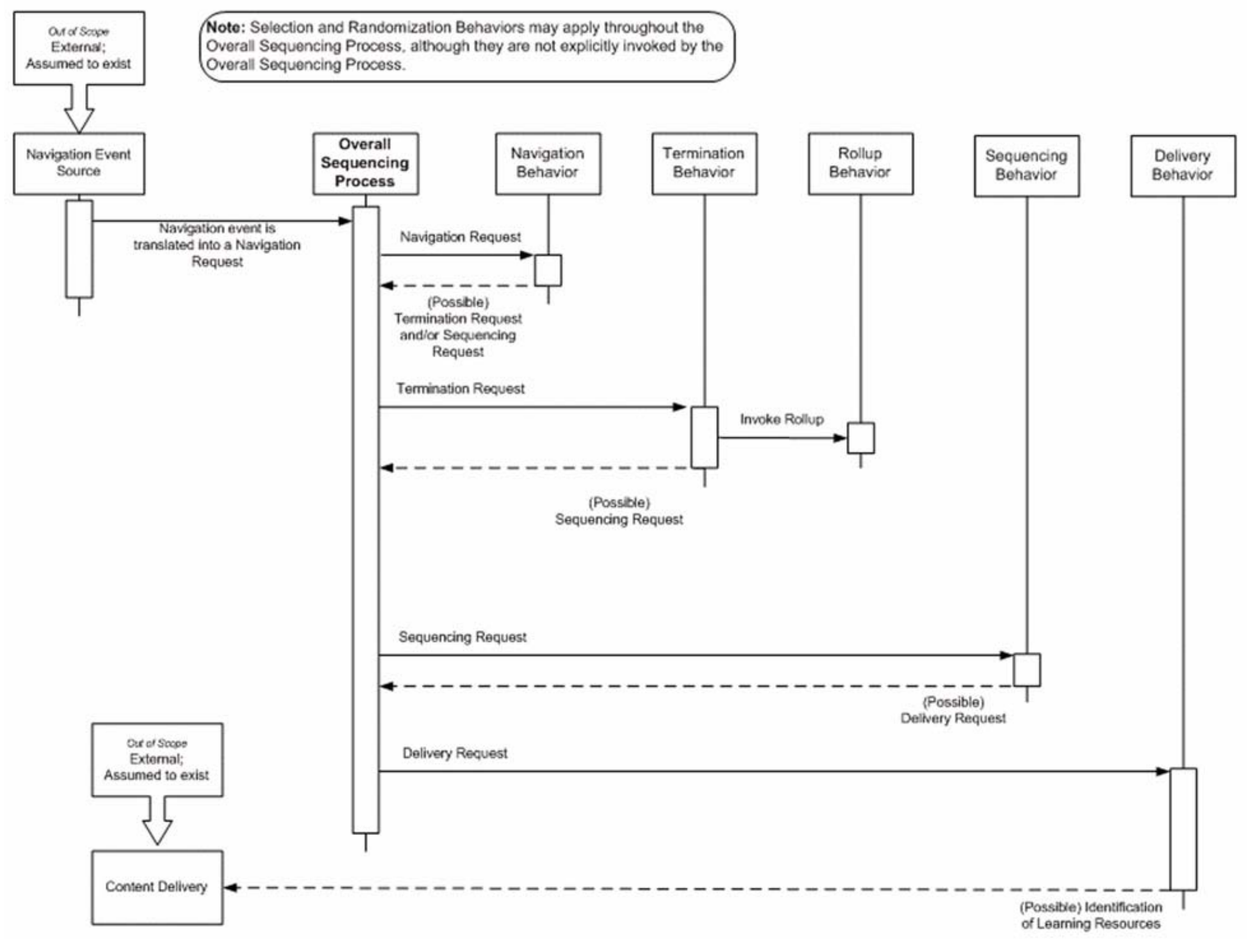
Out of Scope
External;
Assumed to exist
Note: Selection and Randomization Behaviors may apply throughout the Overall Sequencing Process, although they are not explicitly invoked by the Overall Sequencing Process.
Navigation Event Source
Overall Sequencing Process
Navigation Behavior
Termination Behavior
Rollup Behavior
Sequencing Behavior
Delivery Behavior
Navigation event is translated into a Navigation Request
Navigation Request
(Possible) Termination Request and/or Sequencing Request
Termination Request
Invoke Rollup
(Possible) Sequencing Request
Sequencing Request
(Possible) Delivery Request
Delivery Request
Out of Scope
External;
Assumed to exist
Content Delivery
(Possible) Identification of Learning Resources

Figure OP.1 - The process flow of the Overall Sequencing Process.

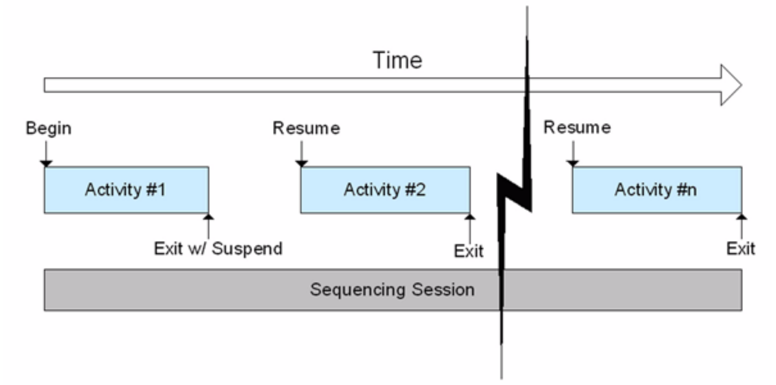

Figure OP.2 - Sequencing Session Timeline.

Figure OP.2 illustrates a portion of a sequencing session. A sequencing session may be triggered in one of three ways:

1) A *Start* sequencing request.
2) A *Resume All* sequencing request.
3) A *Choice* sequencing request prior to either a *Start* or *Resume All* sequencing request.

A sequencing session ends normally when an *Exit* sequencing request is processed from the root of the activity tree.

The overall behavior assumes one request is fully processed before another request is processed. There is no constraint on the timing of when requests are received. The description does not detail or otherwise make any assumptions as to how the requests are serialized or how any request may preempt or terminate any in-process request.

The *Overall Sequencing Process* is specified by the following pseudo code. The pseudo code describes only the processing. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Overall Sequencing Process* only describes the expected behavior that an implementation will exhibit. See Figure OP.2.

Overall Sequencing Process:		
	Reference: Content Delivery Environment Process Delivery Request Process Navigation Request Process Sequencing Request Process Termination Request Process	**Section:** DB.2 DB.1.3 NB.2.1 SB.2.12 TB.2.3
1.	**Loop** – Wait for a navigation request	
1.1.	Apply the *Navigation Request Process* to the navigation request	
1.2.	**If** the *Navigation Request Process* returned navigation request *Not Valid* **Then**	
1.2.1.	Handle the navigation request error	Behavior not specified
1.2.2.	**Continue Loop** – wait for the next navigation request	
	End If	
1.3.	**If** there is a termination request **Then**	If the current activity is active, end the attempt on the current activity.
1.3.1.	Apply the *Termination Request Process* to the termination request	
1.3.2.	**If** the *Termination Request Process* returned termination request *Not Valid* **Then**	
1.3.2.1.	Handle the termination request error	Behavior not specified

1.3.2.2.	**Continue Loop** – wait for the next navigation request	
	End If	
1.3.3.	**If** *Termination Request Process* returned a sequencing request **Then**	
1.3.3.1.	Replace any pending sequencing request by the sequencing request returned by the *Termination Request Process*	There can only be one pending sequencing request. Use the one returned by the termination request process, if it exists.
	End If	
	End If	
1.4.	**If** there is a sequencing request **Then**	
1.4.1.	Apply the *Sequencing Request Process* to the sequencing request	
1.4.2.	**If** the *Sequencing Request Process* (SB.2.12) returned sequencing request *Not Valid* **Then**	
1.4.2.1.	Handle the sequencing request error	Behavior not specified
1.4.2.2.	**Continue Loop** – wait for the next navigation request	
	End If	
1.4.3.	**If** the *Sequencing Request Process* returned a request to end the sequencing session **Then**	
1.4.3.1.	**Exit** *Overall Sequencing Process* – the sequencing session has terminated; return control to LTS	Exiting from the root of the activity tree ends the sequencing session; return control to the LTS
	End If	
1.4.4.	**If** the *Sequencing Request Process* did not identify an activity for delivery **Then**	
1.4.4.1.	**Continue Loop** – wait for the next navigation request	
	End If	
1.4.5.	delivery request is for the activity identified by the *Sequencing Request Process*	
	End If	
1.5.	**If** there is a delivery request **Then**	
1.5.1.	Apply the *Delivery Request Process* to the delivery request	
1.5.2.	**If** the *Delivery Request Process* returned delivery request *Not Valid* **Then**	
1.5.2.1.	Handle the delivery request error	Behavior not specified
1.5.2.2.	**Continue Loop** – wait for the next navigation request	
	End If	
1.5.3.	Apply the *Content Delivery Environment Process* to the delivery request	
	End If	
2.	**End Loop** – wait for the next navigation request	

NB. Navigation Behavior Model

When a learner is interacting with a learning experience through a user interface, each leaner request to move through the content or branch within the learning experience results in a "navigation event", e.g., a click to move to the next activity. These events map to a set of "navigation requests". In turn, each navigation request maps to an "termination request" and a "sequencing request" that will be used to both terminate the attempt on the current activity and determine the 'next' activity in the learning experience. There is a mapping or resolution process that converts a navigation request into the corresponding termination and sequencing requests. The process of evaluating the navigation request and identifying the appropriate termination and sequencing requests (or returning an error) is called the "navigation process".

The navigation process makes no assumptions as to how or when a navigation request is generated. The navigation process makes no assumptions about user interfaces and rendering of navigation structures and controls, either within content objects or external to content objects. Mapping of a user interaction to the generation of a navigation requests outside of the scope of this specification. The navigation behavior is defined only for the navigation request.

The navigation process only provides minimal insurance that navigation requests are valid. If the navigation process declares a navigation request valid, it does not guarantee that any ensuing sequencing request will result in identifying a 'next' activity. The navigation process makes no assumptions on how or when the identified termination and sequencing requests will be processed. In most cases, the termination and sequencing requests will be passed immediately, by the overall sequencing process (see Overall Sequencing Process **OP**), to the termination and sequencing processes.

The overall sequencing process relates the navigation process to the termination, rollup, selection and randomization, sequencing, and delivery processes.

The navigation process only uses *Sequencing Control Mode* data from the sequencing definition model.

The navigation process does not use data from the tracking model.

The navigation process uses data from the activity state model.

The behavior of the navigation process is defined in terms of a single process:

- *Navigation Request Process* – processes the navigation request to identify the corresponding sequencing and termination requests.

NB.1 Navigation Requests

The navigation process responds to one of a set of different navigation requests. A navigation request defines the actions to be performed when the request is processed. The request names are tokens in a vocabulary. The names have no semantics or meanings themselves. The definition of the action is the complete definition of the required behavior.

Navigation Request	Action
Start	If the *Current Activity (AM.1.2) is undefined, Issue a Start* sequencing request.
Resume All	If the *Current Activity (AM.1.2) is undefined and the Suspended Activity* (AM.1.2) is defined, Issue a *Resume All* sequencing request.
Continue	If *Activity is Active* (AM.1.1) for the *Current Activity* (AM.1.2) is *True*, Issue an *Exit* termination request. Issue a *Continue* sequencing request.
Previous	If *Activity is Active* (AM.1.1) for the *Current Activity* (AM.1.2) is *True*, Issue an *Exit* termination request. Issue a *Previous* sequencing request.
Forward	Issue a sequencing request to traverse the "history-based activity record" forward in time. The corresponding sequencing request and associated behavior is not specified in this version of the Simple Sequencing Specification.
Backward	Issue a sequencing request to traverse the "history-based activity record" backward in time. The corresponding sequencing request and associated behavior is not specified in this version of the Simple Sequencing Specification.

Navigation Request	Action
Choice	If *Activity is Active* (AM.1.1) for the *Current Activity* (AM.1.2) is *True*, Issue an *Exit* termination request. Issue a *Choice* sequencing request. The request is accompanied by the identification of the target activity.
Exit	Issue an *Exit* termination request. Issue an *Exit* sequencing request. The current attempt on the *Current Activity* (AM.1.2) is terminated normally; the attempt is over. The termination of the activity was not the result of any other external navigation event (e.g., *Continue, Previous, Choice*).
Exit All	Issue an *Exit All* termination request. Issue an *Exit* sequencing request.
Suspend All	Issue a *Suspend All* termination request. Issue an *Exit* sequencing request. The current attempt on the *Current Activity* (AM.1.2) and all of its ancestors are terminated normally; the attempts are not over and the activities are not complete. The activities may be resumed at some time in the future (resumption is not a new attempt). A Simple Sequencing processor must record sufficient state and tracking information so that the activities may be resumed in the future.
Abandon	Issue an *Abandon* termination request. Issue an *Exit* sequencing request. The current attempt on the *Current Activity* (AM.1.2) is terminated abnormally and the activity is not complete. The attempt may not be resumed. There is no rollback of any tracking data.
Abandon All	Issue an *Abandon All* termination request. Issue an *Exit* sequencing request. The current attempt on the *Current Activity* (AM.1.2) and all of its ancestors are terminated abnormally and the activities are not complete. The attempts may not be resumed. There is no rollback of any tracking data.

NB.2 Navigation Behavior

The navigation behavior describes how a navigation processor interprets a navigation request to validate the navigation request and to identify the corresponding termination and sequencing requests.

An implementation must be capable of representing the processes described and have the implemented process exhibit the behavior described. There are no additional requirements on implementing the navigation behavior model.

NB.2.1 Navigation Request Process

The *Navigation Request Process* examines the navigation request, validates it and determines the corresponding termination and sequencing requests. The process may return a sequencing request. The process may optionally return an termination request. If the process does not return an termination request, there is no termination request for further processing. The process implements the behaviors that describe the navigation requests.

The *Navigation Request Process* for a navigation request is specified by the following pseudo code. The pseudo code describes only the navigation request processing and conversion to termination and sequencing requests. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Navigation Request Process* only describes the expected behavior that an implementation will exhibit.

Navigation Request Process (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):

Reference:	**Section:**
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*; **Sequencing Request**: *Start*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
1.2.	**Else**	
1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	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*; **Sequencing Request**: *Resume All*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
2.1.2.	**Else**	
2.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	End If	
2.2.	**Else**	
2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	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*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	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*)	
3.2.2.	**Else**	
3.2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Sequencing Request**: *Continue*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	End If	
3.3.	**Else**	
3.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	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*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	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*)	
4.2.1.2.	**Else**	
4.2.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Sequencing Request**: *Previous*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	End If	
4.2.2.	**Else**	
4.2.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	Violates control mode.
	End If	

4.3.	**Else**	
4.3.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	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*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	End Case	
6.	**Case:** navigation request is *Backward*	Behavior not defined.
6.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	
	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.	Find the common ancestor of the *Current Activity* and the activity specified by the *Choice* navigation request	
7.1.1.2.	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.
7.1.1.3.	**If** the activity path is **Not** *Empty* **Then**	
7.1.1.3.1.	**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.
7.1.1.3.1.1.	**If** the *Activity is Active* for the activity is *True* **And** the *Sequencing Control Choice Exit* for the activity is *False* **Then**	
7.1.1.3.1.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	Violates control mode.
	End If	
	End For	
7.1.1.4.	**End If**	
7.1.1.4.1.	**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.

7.1.1.4.1.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Termination Request**: *Exit*; **Sequencing Request**: *Choice*; **Target Activity**: the activity specified by the *Choice* navigation request)	
7.1.1.4.2.	**Else**	
7.1.1.4.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Valid*; **Sequencing Request**: *Choice*; **Termination Request**: *n/a*; **Target Activity**: the activity specified by the *Choice* navigation request)	
	End If	
7.1.2.	**Else**	
7.1.2.1.	**Exit** *Navigation Request Process* (**Navigation Request**: *Not Valid*; **Sequencing Request**: *n/a*; **Termination Request**: *n/a*; **Target Activity**: *n/a*)	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*)	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*)	
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*)	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*)	
	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*)	
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*)	
	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*)	
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*)	
	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*)	
	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*)	
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*)	
	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*)	
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*)	
	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*)	Undefined navigation request.

TB. Termination Behavior Model

The Simple Sequencing process delivers activities and tracks the current activity and other activities that have been attempted but not completed. Activities finish or are terminated due to other events and requests. When the activity finishes, the triggering event may imply that one or more other activities must also terminate or exit. Upon termination, post condition sequencing rules may imply that a pending sequencing request is replaced by the sequencing request specified in the sequencing rule. The process of evaluating exit action and post condition sequencing rules, recording information about the state of the activity when it finishes, and terminating other activities based on the defined sequencing behaviors is called the “termination process”.

The termination process makes no assumptions as to how or when a termination request is generated.

The overall sequencing process (see Overall Sequencing Process **OP**) relates the termination process to the navigation, sequencing, rollup, selection and randomization, and delivery processes.

The termination process is controlled by parts of the sequencing definition model:

- *Sequencing Rule Definitions* – rules that when applied to an activity are used to specify sequencing behaviors for the activity.

The termination process uses parts of the tracking model:

- *Objective Information* – information about the results of the learner’s interactions related to an objective.
- *Progress Information* – information about a learner’s attempt at an activity.

The termination process uses data from the activity state model:

- *Activity State Information* – information about the results of the learner’s state for an activity.

The behavior of the termination process is defined in terms of three process and two associated subprocesses:

- *Termination Request Process* – processes the termination request on the Current Activity (AM.1.2).
- *End Attempt Process* (UP.4) – processes completion data when the sequencing engine, not content objects, controls completion, and ends the current attempt on the activity.
- *Overall Rollup Process* – propagates state information (see Tracking Model **TM**) from Current Activity (AM.1.2) to its ancestors.
- *Sequencing Exit Action Rules Subprocess* – evaluates Exit action sequencing rules when an activity terminates and may identify an exiting ancestor.
- *Sequencing Post Condition Rules Subprocess* – evaluates Post Condition sequencing rules when an non-suspended activity terminates and may identify a sequencing request.

TB.1 Termination Requests

The termination process responds to one of a set of different termination requests. A termination request defines the actions to be performed when the request is processed. The request names are tokens in a vocabulary. The names have no semantics or meanings themselves. The definition of the action is the complete definition of the required behavior.

Termination Request	**Action**
Exit	The current attempt on the *Current Activity (AM.1.2) is terminated normally; the attempt is over.*
Exit All	The current attempts on all active activities (from the root to the *Current Activity* (AM.1.2), inclusive) are terminated normally; the attempts are over.
Suspend All	The current attempts on all active activities (from the root to the *Current Activity* (AM.1.2), inclusive) are suspended. The attempt on the *Current Activity* (AM.1.2) may be resumed.
Abandon	The current attempt on the *Current Activity (AM.1.2) is terminated abnormally; the activity is not complete. The attempt may not be resumed. There is no rollback of any tracking data.*

Termination Request	Action
Abandon All	The current attempts on all active activities (from the root to the *Current Activity* (AM.1.2), inclusive) are terminated abnormally; the activities are not complete. Attempts on any abandoned activity may not be resumed. There is no rollback of any tracking data.

TB.2 Termination Behavior

The termination behavior describes how a termination request is processed. It marks attempts on activities as ended, updates the state of the activity tree, processes exit action and post condition sequencing rules, and may identify an alternative sequencing request.

An implementation must be capable of representing the processes described and have the implemented process exhibit the behavior described. There are no additional requirements on implementing the termination behavior model.

The termination behavior relies on the data descriptions from the sequencing definition model (see Sequencing Definition Model **SM**), the tracking model (see Tracking Model **TM**), and the activity state model (see Activity State Model **AM**). These information models also specify default data values in the activity tree that govern the access to activity, tracking, or state data. Simple Sequencing does not specify how content can set the data values in the tracking model.

TB.2.1 Sequencing Exit Action Rules Subprocess

Exit action sequencing rules on the current activity's ancestors are evaluated when the attempt on the current activity ends. This rule evaluation may result in the identification of one and only one of the current activity's ancestors to also terminate. The current attempt on all descendents of the identified activity end.

Sequencing rule conditions are specified by the Sequencing Rule Definitions. Sequencing rules are evaluated using the *Sequencing Rules Check Process* (UP.2).

- This subprocess assumes that the *Overall Rollup Process* (RB.1.4) has completed. It evaluates only sequencing rules whose actions are defined as Exit Action rules (see SM 1.2 *Sequencing Rule Description*), for all ancestors of the current activity. Rules on each activity, beginning at the root of the sequencing tree, are evaluated in order. The subprocess stops at the first rule that evaluates to True. This subprocess may cause one and only one of the current activity's ancestors to terminate. If one of the current activity's ancestors terminates, all of its active descendents also terminate and that activity becomes the current activity. See Figures TB.1 and TB.2.

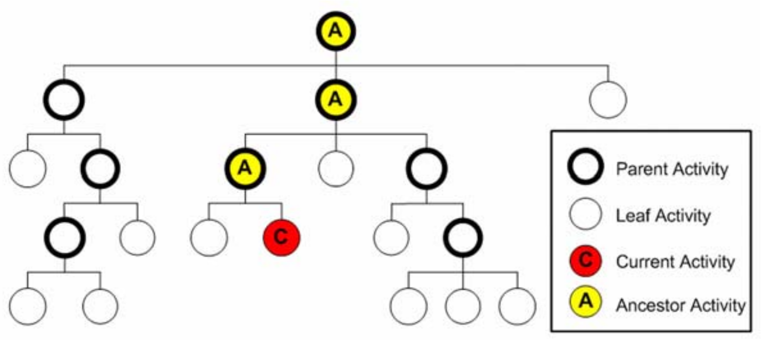

Figure TB.1 - Sequencing Exit Action Rules Subprocess – Start State.

Figure TB.1 illustrates an activity tree for a sequencing session that is "in progress". The current activity and all of its ancestors are active – an attempt on them is in progress and durations are accumulating.

The current activity has been identified in a termination request and the *Exit* request process has been initiated, e.g., a navigation request triggered exit. The *End Attempt Process* (UP.4) is applied to the current activity and Rollup is performed to ensure the activity tree is properly updated.

When the *Sequencing Exit Action Rules Subprocess* is applied to the current activity, one and only one of its ancestors may terminate. The ancestor that terminates is the one closest to the root of the activity tree that has an *Exit Action Rule* that evaluates to *True*. For example, assume both the parent activity and the grandparent activity of the current activity have *Exit Action Rules* that evaluate to True; only the grandparent activity will terminate. When an ancestor of the current activity terminates, the *Sequencing Exit Action Rules Subprocess* will terminate the attempt on the ancestor and all of its active descendents by applying the *Terminate Descendent Attempts Process* (UP.3) and make the current activity the activity identified for exit. For the example described above, Figure TB.2 shows the activity tree after the *Sequencing Exit Action Rules Subprocess* has completed.

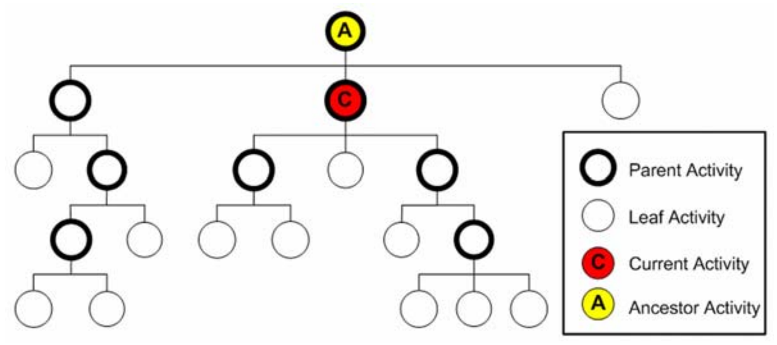

Figure TB.2 - Sequencing Exit Action Rule Subprocess – End State.

The *Sequencing Exit Action Rules Subprocess* evaluates sequencing rules for the ancestors of the current activity, and as a result, one of those activities may terminate.

The *Sequencing Exit Action Rules Subprocess* is specified by the following pseudo code. The pseudo code describes only the exit action rule processing. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Sequencing Exit Action Rules Subprocess* only describes the expected behavior that an implementation will exhibit.

Sequencing Exit Action Rules Subprocess (for the *Current Activity*; may change the *Current Activity*):		
	Reference: Current Activity End Attempt Process Sequencing Rules Check Process Sequencing Rule Description Terminate Descendent Attempts Process	**Section:** AM.1.2 UP.4 UP.2 SM.2 UP.3
1.	Form the activity path as the ordered series of activities from the root of the activity tree to the parent of the *Current Activity*, inclusive	
2.	Initialize exit target to *Null*	
3.	**For** each activity in the activity path	Evaluate all exit rules along the active path, starting at the root of the activity tree.
3.1.	Apply the *Sequencing Rules Check Process* to the activity and the set of *Exit* actions	
3.2.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	
3.2.1.	Set the exit target to the activity	Stop at the first activity that has an exit rule evaluating to true.
3.2.2.	**Break For**	
	End If	

	End For	
4.	**If** exit target is **Not** *Null* **Then**	
4.1.	Apply the *Terminate Descendent Attempts Process* to the exit target	End the current attempt on all active descendents.
4.2.	Apply the *End Attempt Process* to the exit target	End the current attempt on the 'exiting' activity.
4.3.	Set the *Current Activity* to the exit target	Move the current activity to the activity that identified for termination.
	End If	
5.	**Exit** *Sequencing Exit Action Rules Subprocess*	

TB.2.2 Sequencing Post Condition Rules Subprocess

Post condition sequencing rules are applied to the current activity when it terminates, if the current activity was not suspended. These sequencing rules may override the sequencing request generated from a navigation request or produce an additional termination request. The *Sequencing Post Condition Rules Subprocess* evaluates post condition sequencing rules on the current activity and may identify a termination or sequencing request.

Sequencing rule conditions are specified by the Sequencing Rule Definitions. Sequencing rules are evaluated using the *Sequencing Rules Check Process* (UP.2).

- This subprocess assumes the *Sequencing Exit Action Rules Subprocess* has completed, so that the *Current Activity* (AM.1.2) is accurate, given the current state of the activity tree. It evaluates only sequencing rules whose actions are defined as Post Condition rules (see SM 1.2 *Sequencing Rule Description*), on the current activity. Rules are only evaluated if the current activity was not suspended during its previous attempt. The subprocess stops at the first rule that evaluates to True and may return a termination and sequencing request.

The *Sequencing Post Condition Rules Subprocess* evaluates sequencing rules for the identified activity and may optionally return one and only one termination request and one and only one sequencing request, determined by the rule action.

The *Sequencing Post Condition Rules Subprocess* is specified by the following pseudo code. The pseudo code describes only the post condition rule processing. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Sequencing Post Condition Rules Subprocess* only describes the expected behavior that an implementation will exhibit.

Sequencing Post Condition Rules Subprocess (for the C*urrent Activity*; may return a termination request and a sequencing request):		
	Reference: Activity is Suspended Current Activity Sequencing Rules Check Process Sequencing Rule Description	**Section:** AM.1.1 AM.1.2 UP.2 SM.2
1.	**If** *Activity is Suspended* for the C*urrent Activity* is *True* **Then**	Do not apply post condition rules to a suspended activity.
1.1.	**Exit** *Sequencing Post Condition Rules Subprocess*	
	End If	
2.	Apply the *Sequencing Rules Check Process* to the *Current Activity* and the set of *Post Condition* actions	Apply the post condition rules to the current activity.
3.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	

3.1.	**If** the *Sequencing Rules Check Process* returned *Retry, Continue,* **Or** *Previous* **Then**	
3.1.1.	**Exit** *Sequencing Post Condition Rules Subprocess* (**Sequencing Request**: the value returned by the *Sequencing Rules Check Process*; **Termination Request**: *n/a*)	Attempt to override any pending sequencing request with this one.
	End If	
3.2.	**If** the *Sequencing Rules Check Process* returned *Exit Parent* **Or** *Exit All* **Then**	
3.2.1.	**Exit** *Sequencing Post Condition Rules Subprocess* (**Sequencing Request**: *n/a*; **Termination Request**: the value returned by the *Sequencing Rules Check Process*)	Terminate the appropriate activity(s)
	End If	
3.3.	**If** the *Sequencing Rules Check Process* returned *Retry All* **Then**	
3.3.1.	**Exit** *Sequencing Post Condition Rules Subprocess* (**Termination Request**: *Exit All*; **Sequencing Request**: *Retry*)	Terminate all active activities and move the current activity to the root of the activity tree; then perform an 'in-process' start.
	End If	
	End If	
4.	**Exit** *Sequencing Post Condition Rules Subprocess*	

TB.2.3 Termination Request Process

The *Termination Request Process* processes the termination request.

The process uses the *Overall Rollup Process* (RB.1.4), the *Sequencing Exit Action Rules Subprocess* (TB.2.1), the *Sequencing Post Condition Rules Subprocess* (TB.2.2), and the *End Attempt Process* (UP.4). It updates values in the tracking model and the activity state model.

- A termination request results in the current attempt on the current activity ending and status information in the activity tree being updated (through the *End Attempt Process* and the *Overall Rollup Process*). The *Sequencing Exit Action Rules Subprocess* may cause certain ancestors of the current activity to also terminate. The *Sequencing Post Condition Rule Subprocess* evaluates post condition rules on the *Current Activity* (AM.1.2). The result of the *Termination Request Process* may identify a new sequencing request.

Certain termination requests (Suspend All, Abandon, and Abandon All) do not result in any tracking model status change to the current activity; the *End Attempt Process* is not invoked for these termination requests.

The *Termination Request Process* processes the termination request and optionally returns a new sequencing request.

The *Termination Request Process* for a termination request is specified by the following pseudo code. The pseudo code describes only the termination request processing. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Termination Request Process* only describes the expected behavior that an implementation will exhibit.

Termination Request Process (for a termination request, ends the current attempt on the *Current Activity*; returns the validity of the termination request; may return a sequencing request):

Reference:	**Section:**
Activity is Active	AM.1.1
Activity is Suspended	AM.1.1
Current Activity	AM.1.2
End Attempt Process	UP.4
Overall Rollup Process	RB.1.4
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 already begun, there is nothing to terminate.
1.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a*)	
	End If	
2.	**If** *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*)	
	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 *Overall Rollup Process* to the *Current Activity*	Propagate the state of the current activity up the tree.
3.3.	Apply the *Sequencing Exit Action Rules Subprocess* to the *Current Activity*	Check if any of the current activity's ancestors need to terminate.
3.4.	**Repeat**	
3.4.1.	Set the processed exit to *False*	
3.4.2.	**If** the *Current Activity* is **Not** the root of the activity tree **Then**	
3.4.2.1.	Apply the *Sequencing Post Condition Rules Subprocess* to the *Current Activity*	
3.4.2.2.	**If** the *Sequencing Post Condition Rule Subprocess* returned a termination request of *Exit All* **Then**	
3.4.2.2.1.	Change the termination request to *Exit All*	
3.4.2.2.2.	**Break** to the next **Case**	
	End If	
3.4.2.3.	**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.

3.4.2.3.1.	Set the *Current Activity* to the parent of the *Current Activity*	
3.4.2.3.2.	Apply the *End Attempt Process* to the *Current Activity* (AM.1.2)	
3.4.2.3.3.	Set processed exit to *True*	Need to evaluate post conditions on the new current activity.
	End If	
	End If	
3.5.	**Until** processed exit is *False*	
3.6.	**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*)	
	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*	
4.1.2.	Apply the *Overall Rollup 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)	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* **Then**	If the current activity is active, 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 *Activity is Suspended* for the *Current Activity* is *False* **Then**	The current activity has already terminated; check if it was suspended.
5.2.1.1.	**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.
5.2.1.1.1.	Set the *Suspended Activity* to the parent of the *Current Activity*	
5.2.1.2.	**Else**	
5.2.1.2.1.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a*)	Nothing to suspend.
	End If	

	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*)	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*)	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*)	
	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*)	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*)	Inform the sequencer that the sequencing session has ended.
	End Case	
8.	**Exit** *Termination Request Process* (**Termination Request**: *Not Valid;* **Sequencing Request**: *n/a*)	Undefined termination request.

RB. Rollup Behavior Model

Simple Sequencing processes uses information about the results of a learner’s interactions with activities (e.g., complete, incomplete) and the learner’s record for objectives (e.g., satisfaction, score) to control the sequencing of other activities. The data attributes that describe the results of the learner’s interactions, i.e., the elements of the tracking model, may be referenced for any activity in the activity tree. The results data for an activity may be determined from the results data for the children of the activity. The process of computing the results data for an activity from the results data from the children of the activity is called the “rollup process” or just “rollup”.

The rollup process is used to maintain the tracking model data for all activities and objectives in the activity tree. The Termination Request Process (see Termination Behavior **TB**) invokes the rollup process to update the tracking model data when an attempt on an activity terminates. Other processes may invoke the rollup process at other times, but such invocation is not detailed in this specification.

The rollup process is applied to all parts of the tracking model (see Tracking Model **TM**):

- *Objective Information* – information about the results of the learner’s interactions related to an objective.
- *Progress Information* – information about results of the learner’s interaction on an attempt at an activity.

The rollup process is controlled by parts of the sequencing definition model (see Sequencing Definition Model **SM**):

- *Rollup Controls* – controls which data from an activity contributes to the rollup of the parent activity.
- *Rollup Rule Definitions* – rules that indicate how tracking data from an activity or its associated objectives is produced from the results of the children of the activity.
- *Objective Description* – the learning objective(s) associated with an activity.
- *Delivery Controls* – controls when progress and objective information for an activity is recorded.

Rollup behavior is defined by the *Overall Rollup Process* (RB.1.5); the process propagates results data from a given activity up the activity tree. The behavior of the *Overall Rollup Process* is defined in terms of different processes, each of which is applied to one part of the tracking model. Each individual process is applied only if enabled by the corresponding *Rollup Control* (SM.8) elements in the sequencing definition model. Rollup of both Objective and Progress values is applied only to tracked activities. Rollup is not otherwise conditional on any other sequencing rules or conditions:

- *Measure Rollup Process* – determines the normalized measure (e.g., score) of the objective associated with an activity.
- *Objective Rollup Process* – determines the objective progress status of the objective associated with an activity.
- *Activity Progress Rollup Process* – determines the completion status of an activity.

The *Objective Rollup Process* (RB.1.2) and *Activity Progress Rollup Process* (RB.1.3) may evaluate rollup rules associated with the activity to determine the activity’s objective and progress status using the *Rollup Rule Check Process* (RB.1.4).

RB.1 Rollup Behavior

The rollup behavior describes how a sequencing system interprets the elements of the sequencing definition model in combination with instance data from the tracking model to update the tracking model.

An implementation must be capable of representing the processes described and have the implemented process exhibit the behavior described. There are no additional requirements on implementing the rollup behavior model.

The rollup behavior relies on the data from the sequencing definition model (see Sequencing Definition Model **SM**) and the tracking model (see Tracking Model **TM**). These information models also specify default data values that govern the access to activity or tracking data.

RB.1.1 Measure Rollup Process

Measure rollup is used to determine the normalized measure for the objective that is associated with an activity and that has the attribute value *Objective Contributes to Rollup* (SM.6) value of True. The normalized measure is computed from the values of the measures for the objectives that are associated with the children of the activity and that have the attribute value *Objective Contributes to Rollup* value of True.

Measure Rollup computations (i.e., computing the values of *Objective Measure Status* and *Objective Normalized Measure* from TM.1.1) are controlled by the Rollup Control value of *Rollup Objective Measure Weight* (SM.8) and the Delivery Control value of *Tracked* (SM.11).

The measure rollup behavior is defined by a weighted measure-based computation.

- All *Tracked* children with a non-negative *Rollup Objective Measure Weight* are included in the computation. If the *Objective Measure Status* (TM.1.1) is False for the objective associated with a child activity that has the attribute value *Objective Contributes to Rollup* (SM.6) of True (i.e., there is no measure for the child), then the *Objective Measure Status* is False for the objective associated with the activity that has the attribute value *Objective Contributes to Rollup* of True. Otherwise, compute a weighted average measure across all of the activity's children for the objectives. The computation of the *Objective Normalized Measure* (TM.1.1) for the objective that is associated with the child activity is weighted by the *Rollup Objective Measure Weight* for the child activity. The computation is only performed using the child activity's objectives that have the attribute value *Objective Contributes to Rollup* value of True.

The sum of the *Rollup Objective Measure Weight* values for the contributing child activities need not total to 1.0.

The *Measure Rollup Process* for an activity is specified in the following pseudo code. The pseudo code describes only the measure rollup logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Measure Rollup Process* only describes the expected behavior that an implementation will exhibit.

Measure Rollup Process (for an activity; may change the *Objective Information*for the activity):		
	Reference: Objective Contributes to Rollup; Objective Description; Objective Measure Status; Objective Normalized Measure; Rollup Objective Measure Weight; Tracked	**Section:** SM.6; SM.6; TM.1.1; TM.1.1; SM.8; 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.	**If** the *Objective Measure Status* for the rolled-up objective is *True* **Then**	
5.1.1.3.1.1.	**If** the *Rollup Objective Measure Weight* for the child is *Non-negative (>= 0)* **Then**	Only include children with non-negative weights.
5.1.1.3.1.1.1.	Increment counted measures by the *Rollup Objective Measure Weight* for the child	
5.1.1.3.1.1.2.	Add the product of *Objective Normalized Measure* (TM.1.1) for the rolled-up objective multiplied by the *Rollup Objective Measure Weight* for the child to the total weighted measure	
	End If	
5.1.1.3.2.	**Else**	
5.1.1.3.2.1.	Set counted measures to *Zero (0.0)*	No measure defined for the child; incomplete tracking state.
5.1.1.3.2.2.	**Break For**	
	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*	None or incomplete 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.

RB.1.2 Objective Rollup Process

Objective rollup is used to determine the objective status of the objective associated with an activity from the values of the objective information associated with the children of the activity.

Objective Rollup computations (i.e., computing the values of *Objective Progress Status* and *Objective Satisfied Status* from TM.1.1) are controlled by the Rollup Controls value of *Rollup Objective Satisfied Status* (SM.8) and the *Rollup Rules* (SM.5) for the activity.

The objective rollup behavior is defined by a measure-based computation, a rule-based computation or an alternative default rule-based computation, applied in that order.

1) *Measure-Based Computation* – The objective is satisfied (set values for *Objective Progress Status* and *Objective Satisfied Status*) if the *Objective Normalized Measure* (TM.1.1) for the objective is equal to or greater than the *Objective Minimum Satisfied Normalized Measure* (SM.6) for the objective. Measure-based rollup is used only if the *Objective Satisfied by Measure* (SM.6) attribute is True for the activity's objective that has the attribute value *Objective Contributes to Rollup* (SM.6) of True.
2) *Rule-Based Computation* – Rollup rules with *Rollup Actions* (SM.5) of *satisfied or not satisfied* are evaluated for the activity. Child activities that are part of the child set (i.e., child activities that have *Rollup Objective Satisfied* (SM.8) equal to *True* and have *Tracked* (SM.11) equal to *True*) are used to evaluate the *Rollup Conditions* (SM.5) of the rule. No rules are evaluated if there are no child activities that have a *Rollup Objective Satisfied* value of *True.* If the rollup rule evaluates to *True*, the *Rollup Action* sets values for *Objective Progress Status* and *Objective Satisfied Status* for the objective. The computation is performed only if the objective has the attribute value *Objective Contributes to Rollup* value of *True*. All rules with *Rollup Action* of *Not Satisfied* are evaluated before evaluating any rules with *Rollup Action* of *Satisfied*; thus, the rollup rules for *Satisfied* override rollup rules for *Not Satisfied*. If no rules fire, there is no change to the values of *Objective Progress Status* and *Objective Progress Status* for the objective.
3) Default *Rule-Based Computation* – If the activity does not have any defined rollup rules with *Rollup Actions* of *Satisfied* or *Not Satisfied,* the rule-based computation described above is applied using the following default rollup rules:
    - Default *Satisfied* Rule – a *Rollup Child Set* value of *All,* a *Rollup Condition* value of *Satisfied,* and a *Rollup Action* value of *Satisfied.*
    - Default *Not Satisfied* Rule – a *Rollup Child Set* value of *All, Rollup Condition* values of *Not Satisfied* **And** (the *Condition Combination* of *All) Attempted,* and a *Rollup Action* value of *Not Satisfied*.

The *Objective Rollup Process* for an activity is specified in the following pseudo code. The pseudo code describes only the objective rollup logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Objective Rollup Process* only describes the expected behavior that an implementation will exhibit.

Objective Rollup Using Measure Process (for an activity; may change the *Objective Information* for the activity):

Reference:	**Section:**
Objective Contributes to Rollup	SM.6
Objective Description	SM.6
Objective Satisfied by Measure	SM.6
Objective Measure Status	TM.1.1
Objective Normalized Measure	TM.1.1
Objective Progress Status	TM.1.1
Objective Satisfied Status	TM.1.1

1.	Set the target objective to *Undefined*	
2.	**For** each objective associated with the activity	
2.1.	**If** *Objective Contributes to Rollup* for the objective is *True* **Then**	Identify the objective that may be altered based on the activity's children's rolled up measure.
2.1.1.	Set the target objective to the objective	
2.1.2.	**Break For**	
	End If	
	End For	
3.	**If** target objective is *Defined* **Then**	
3.1.	**If** *Objective Satisfied by Measure* for the target objective is *True* **Then**	If the objective is satisfied by measure, test the rolled-up measure against the defined threshold.
3.1.1.	**If** the *Objective Measure Status* for the target objective is *False* **Then**	
3.1.1.1.	Set the *Objective Progress Status* for the target objective to *False*	
3.1.2.	**Else**	
3.1.2.1.	**If** the *Objective Normalized Measure* for the target objective is equal to or greater than the *Objective Minimum Normalized Measure* (SM.6) for the target objective **Then**	
3.1.2.1.1.	Set the *Objective Progress Status* for the target objective to *True*	
3.1.2.1.2.	Set the *Objective Satisfied Status* for the target objective to *False*	
3.1.2.2.	**Else**	
3.1.2.2.1.	Set the *Objective Progress Status* for the target objective to *True*	
3.1.2.2.2.	Set the *Objective Satisfied Status* for the target objective to *True*	
	End If	
	End If	
	End If	
3.2.	**Exit** *Objective Rollup Using Measure Process*	
4.	**Else**	
4.1.	**Exit** *Objective Rollup Using Measure Process*	No objective contributes to rollup, so we cannot set anything.
	End If	

Objective Rollup Using Rules Process (for an activity; may change the *Objective Information* for the activity):		
	Reference: Objective Contributes to Rollup Objective Description Objective Progress Status Objective Satisfied Status Rollup Rule Check Subprocess Rollup Action	**Section:** SM.6 SM.6 TM.1.1 TM.1.1 RB.1.4 SM.5
1.	Set the target objective to *Undefined*	
2.	**For** each objective associated with the activity	
2.1.	**If** *Objective Contributes to Rollup* for the objective is *True* **Then**	Identify the objective that may be altered based on the activity's children's rolled up status.
2.1.1.	Set the target objective to the objective	
2.1.2.	**Break For**	
	End If	
	End For	
3.	**If** target objective is *Defined* **Then**	
3.1.	**Apply** the *Rollup Rule Check Subprocess* to the activity and the *Not Satisfied* rollup action	Process all Not Satisfied rules first
3.2.	**If** the *Rollup Rule Check Subprocess* returned *True* **Then**	
3.2.1.	Set the *Objective Progress Status* for the target objective to *True*	
3.2.2.	Set the *Objective Satisfied Status* for the target objective to *False*	
	End If	
3.3.	**Apply** the *Rollup Rule Check Subprocess* to the activity and the *Satisfied* rollup action	Process all Satisfied rules last.
3.4.	**If** the *Rollup Rule Check Subprocess* returned *True* **Then**	
3.4.1.	Set the *Objective Progress Status* for the target objective to *True*	
3.4.2.	Set the *Objective Satisfied Status* for the target objective to *True*	
	End If	
3.5	**Exit** *Check Child for Rollup Subprocess*	
4.	**Else**	
4.1.	**Exit** *Check Child for Rollup Subprocess*	No objective contributes to rollup, so we cannot set anything.
	End If	

RB.1.3 Activity Progress Rollup Process

Activity Progress rollup is used to determine the completion status of the current attempt on an activity from the completion status of the children of the activity.

Attempt Progress computations (i.e., computing the values of *Attempt Progress Status* (TM.1.2.2) and *Attempt Completion Status* (TM.1.2.2) for the activity) are controlled by the Rollup Controls value of *Rollup Progress Completion* (SM.8) for the activity's children and the Rollup Rules (SM.5) for the activity.

The activity progress rollup behavior is defined by a rule-based computation.

1) *Rule-Based Computation* – Rollup Rules with *Rollup Actions* (SM.5) of *Incomplete* or *Completed* are evaluated for the activity. Child activities that are part of the child set (i.e., child activities that have a *Rollup Progress Completion* equal to *True* and *Tracked* (SM.11) equal to *True*) are used to evaluate the set of *Rollup Conditions* of the rule. No rules are evaluated if there are no child activities that have a *Rollup Progress Completion* value of *True.* If the Rollup Rule evaluates to *True*, the *Rollup Action* sets values for *Attempt Progress Status* and *Attempt Completion Status* for the current attempt of the activity. All rules with *Rollup Action* of *Incomplete* are evaluated before evaluating any rules with *Rollup Action* of *Completed*; thus, rollup rules for *Completed* override rollup rules for *Incomplete*. If no rules fire, there is no change to the values of *Attempt Completion Status* for the activity.
2) *Default Rule-Based Computation* – If the activity does not have any defined Rollup Rules with *Rollup Actions* of *Incomplete* or *Completed,* the rule-based computation described above is applied using the following default rules:
    - Default *Completed* Rule – a *Rollup Child Set* value of *All*, a *Rollup Condition* value of *Completed,* and a *Rollup Action* value of *Completed*.
    - Default *Incomplete* Rule – a *Rollup Child Set* value of *All*, *Rollup Condition* values of *Incomplete* **And** (the *Condition Combination* of *All) Attempted*, and a *Rollup Action* value of *Incomplete*.

The *Activity Progress Rollup Process* for an activity is specified in the following pseudo code. The pseudo code describes only the activity progress rollup logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Activity Progress Rollup Process* only describes the expected behavior that an implementation will exhibit.

Activity Progress Rollup Process (for an activity; may change the *Attempt Information* for the activity):		
	Reference: Attempt Completion Status Attempt Progress Status Rollup Rule Check Subprocess Rollup Action	**Section:** TM.1.2.2 TM.1.2.2 RB.1.4 SM.5
1.	**Apply** the *Rollup Rule Check Subprocess* to the activity and the *Incomplete* rollup action	Process all Incomplete rules first
2.	**If** the *Rollup Rule Check Subprocess* returned *True* **Then**	
2.1.	Set the *Attempt Progress Status* for the activity to *True*	
2.2.	Set the *Attempt Completion Status* for the activity to *False*	
	End If	
3.	**Apply** the *Rollup Rule Check Subprocess* to the activity and the *Completed* rollup action	Process all Completed rules last.
4.	**If** the *Rollup Rule Check Subprocess* returned *True* **Then**	
4.1.	Set the *Attempt Progress Status* for the activity to *True*	
4.2.	Set the *Attempt Completion Status* for the activity to *True*	
	End If	
5.	**Exit** *Activity Progress Rollup Process*	

RB.1.4 Rollup Rule Check Subprocess

The *Rollup Rule Check Subprocess* evaluates a set of rollup rules with a given rollup rule action for an activity. The *Rollup Controls* for each of the activity's children are evaluated by the *Check Child for Rollup Subprocess* to determine which children contribute to the rollup evaluation. The *Evaluate Rollup Conditions Subprocess* is applied to each contributing child to determine if its current tracking status meets the rollup rule criteria. If the appropriate set of children, defined by the rollup rule, successfully contribute to rollup, the *Rollup Rule Check Subprocess* indicates that the status of the activity should be changed. The *Rollup Rule Check Subprocess* ends after evaluating a rule that indicates that the status of the activity should be changed.

The *Rollup Rule Check Subprocess* for an activity is specified by the following pseudo code. The pseudo code describes only the rule-checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Rollup Rule Check Subprocess* only describes the expected behavior that an implementation will exhibit.

Rollup Rule Check Subprocess (for an activity and a *Rollup Action*; returns True if the action applies):

Reference:	**Section:**
Check Child for Rollup Subprocess	RB.1.4.2
Evaluate Rollup Conditions Subprocess	RB.1.4.1
Rollup Action	SM.5
Rollup Child Activity Set	SM.5
Rollup Minimum Count	SM.5
Rollup Minimum Percent	SM.5
Rollup Rule Description	SM.5
Tracked	SM.11
Tracking Model	TM

1.	**If** the activity includes *Rollup Rules* with the specified *Rollup Action* **Then**	Make sure the activity has rules to evaluate.
1.1.	Initialize rules list by selecting the set of *Rollup Rules* for the activity that have one of the specified *Rollup Actions*, maintaining original rule ordering	
1.2.	**For** each rule in the rules list	
1.2.1.	Initialize contributing children bag as an empty collection	
1.2.2.	**For** each child of the activity	
1.2.2.1.	**If** *Tracked* for the child is *True* **Then**	
1.2.2.2.1.	Apply *Check Child for Rollup Subprocess* to the child and the *Rollup Action*	Make sure this child contributes to the status of its parent.
1.2.2.2.2.	**If** *Check Child for Rollup Subprocess* returned *True* **Then**	
1.2.2.2.2.1.	Apply the *Evaluate Rollup Conditions Subprocess* to the child and the *Rollup Conditions* for the rule	Evaluate the rollup conditions on the child activity.
1.2.2.2.2.2.	**If** *Evaluate Rollup Conditions Subprocess* returned *True* **Then**	
1.2.2.2.2.2.1.	Add a *True* value to the contributing children bag	
1.2.2.2.2.3.	**Else**	
1.2.2.2.2.3.1.	Add a *False* value to the contributing children bag	
	End If	
	End If	
	End If	
	End For	
1.2.3.	Initialize status change to *False*	Determine if the appropriate children contributed to rollup; if they did, the status of the activity should be changed.
1.2.4.	**Case:** the *Rollup Child Activity Set* is *All*	
1.2.4.1.	**If** the contributing children bag does not contain a value of *False* **Then**	
1.2.4.1.1.	Set status change to *True*	

	End If	
	End Case	
1.2.5.	**Case:** the *Rollup Child Activity Set* is *Any*	
1.2.5.1.	**If** the contributing children bag contains a value of *True* **Then**	
1.2.5.1.1.	Set status change to *True*	
	End If	
	End Case	
1.2.6.	**Case:** the *Rollup Child Activity Set* is *None*	
1.2.6.1.	**If** the contributing children bag does not contain a value of *True* **Then**	
1.2.6.1.1.	Set status change to *True*	
	End If	
	End Case	
1.2.7.	**Case:** the *Rollup Child Activity Set* is *At Least Count*	
1.2.7.1.	**If** the count of *True* values contained in the contributing children bag equals or exceeds the *Rollup Minimum Count* of the rule **Then**	
1.2.7.1.1.	Set status change to *True*	
	End If	
	End Case	
1.2.8.	**Case:** the *Rollup Child Activity Set* is *At Least Percent*	
1.2.8.1.	**If** the percentage (normalized between 0..1, inclusive) of *True* values contained in the contributing children bag equals or exceeds the *Rollup Minimum Percent* of the rule **Then**	
1.2.8.1.1.	Set status change to *True*	
	End If	
	End Case	
1.2.9.	**If** status change is *True* **Then**	
1.2.9.1.	**Exit** *RollupRule Check Subprocess* (**Evaluation**: *True*)	Stop at the first rule that evaluates to true – perform the associated action.
	End If	
	End For	
	End If	
2.	**Exit** *Rollup Rule Check Subprocess* (**Evaluation**: *False*)	No rules evaluated to true – do not perform any action.

RB.1.4.1 Evaluate Rollup Conditions Subprocess

The *Evaluate Rollup Conditions Subprocess* applies a set of rollup rule conditions to an activity to determine if they apply. Each condition in the set is evaluated against the current tracking information for the activity. The condition combination is applied to the evaluated conditions to determine the overall evaluation of the condition set.

The *Evaluate Rollup Conditions Subprocess* for an activity is specified by the following pseudo code. The pseudo code describes only the condition-checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Evaluate Rollup Conditions Subprocess* only describes the expected behavior that an implementation will exhibit.

Evaluate Rollup Conditions Subprocess (for an activity and a set of *Rollup Conditions*; returns True if the condition(s) evaluate to True):		
	Reference: Condition Combination, Rollup Condition, Rollup Condition Operator, Tracking Model	**Section:** SM.5, SM.5, SM.5, TM
1.	Initialize rollup condition bag as an *Empty* collection	This is used to keep track of the evaluation of the rule's conditions.
2.	**For** each *Rollup Condition* in the set of *Rollup Conditions*	
2.1.	Evaluate the rollup condition by applying the appropriate tracking information for the activity to the *Rollup Condition*	Evaluate each condition against the activity's tracking information.
2.2.	**If** the *Rollup Condition Operator* for the *Rollup Condition* is *Not* **Then**	
2.2.1.	**Negate** the rollup condition	
	End If	
2.3.	Add the value of the rollup condition to the rollup condition bag	Add the evaluation of this condition to the set of evaluated conditions.
	End For	
3.	**If** the rollup condition bag is *Empty* **Then**	If there are no defined conditions for the rule, the rule does not apply.
3.1.	**Exit** *Evaluate Rollup Conditions Subprocess* (**Evaluation**: *False*)	
	End If	
4.	Apply the *Condition Combination* to the rollup condition bag to produce a single combined rule evaluation	'And' or 'Or' the set of evaluated conditions, based on the rollup rule definition.
5.	**Exit** *Evaluate Rollup Conditions Subprocess* (**Evaluation**: the value of rule evaluation)	

RB.1.4.2 Check Child for Rollup Subprocess

The *Check Child for Rollup Subprocess* determines if tracking information for an activity should be included during rollup. The subprocess tests the activity's Rollup Controls (see SM.8) against an intended rollup action.

The *Check Child for Rollup Subprocess* for an activity is specified by the following pseudo code. The pseudo code describes only the checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Check Child for Rollup Subprocess* only describes the expected behavior that an implementation will exhibit.

Check Child for Rollup Subprocess (for an activity and a *Rollup Action*; returns True if the activity is included in rollup):		
	Reference: Rollup Action; Rollup Objective Satisfied; Rollup Progress Completion	**Section:** SM.5; SM.8; SM.8
1.	**If** the *Rollup Action* is *Satisfied* **Or** *Not Satisfied* **Then**	
1.1.	**Exit** *Check Child for Rollup Subprocess* (**Child is Included in Rollup**: the *Rollup Objective Satisfied* value for the activity)	Test the objective rollup control.
	End If	
2.	**If** the *Rollup Action* is *Complete* **Or** *Incomplete* **Then**	Test the progress rollup control.
2.1.	**Exit** *Check Child for Rollup Subprocess* (**Child is Included in Rollup**: the *Rollup Progress Completion* value for the activity)	
	End If	.
3.	**Exit** *Check Child for Rollup Subprocess* (**Child is Included in Rollup**: *False*)	Undefined rollup action.

RB.1.5 Overall Rollup Process

The *Overall Rollup Process* is used to propagate tracking results upward through the entire activity tree. The rollup process is applied to:

- An activity tree, designated by the root activity of the tree.
- The activity within the tree that terminated and triggered rollup, i.e., the activity whose tracking data has changed.

The root of the tree and the current activity within the tree specify a unique single path through the tree, denoted the rollup path. Rollup is applied to all activities along that path, starting from the parent of the activity that triggered the rollup and proceeding backward along the path of parents of the activities to the root of the activity tree.

At each step along the rollup path, all rollup processes, *Measure Rollup*, *Objective Rollup*, and *Activity Progress Rollup*, are applied. *Measure Rollup* must be applied before any other rollup process, but there are no other ordering constraints on the rollup processes. If rollup results in a change to an activity's objective information and a 'write' objective map is defined on the activity that transfers objective information to a shared global objective, the shared global objective's information must be set prior to continuing rollup. Whenever the rollup process is initiated, it is performed on all activities along the path; the process does not stop if rollup does not change any values for an activity (other actions may have changed values of other children along different paths that contribute to the rolled up values at some level in the tree).

The *Overall Rollup Process* is specified by the following pseudo code. The pseudo code describes only the rollup logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Overall Rollup Process* only describes the expected behavior that an implementation will exhibit.

Overall Rollup Process (for an activity; may change the tracking information for the activity and its ancestors):		
	Reference: Activity Progress Rollup Process Measure Rollup Process Objective Rollup Process Tracked Tracking Model	**Section:** RB.1.3 RB.1.1 RB.1.2 SM.11 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.	Apply the *Measure Rollup Process* to the activity	Rollup the activity's measure.
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*	

SR. Selection and Randomization Behavior Model

Simple Sequencing identifies activities for delivery to the learner from those defined in the activity tree. The structure of the activity tree may be statically defined, during content development, or may be dynamically generated while a learner is involved in a learning experience. Simple Sequencing places no constraints on when the structure of the activity tree may change. The Selection and Randomization controls define how an activity's children are selected and reordered before and while being used in sequencing.

The overall sequencing process (see Overall Sequencing Process **OP**) does not relate the selection and randomization behaviors to the navigation, termination, rollup, sequencing, and delivery processes. An implementation must invoke the selection and randomization processes at the appropriate times during the overall sequencing process to exhibit the following behaviors:

1) The selection and randomization processes have no effect if applied to a leaf activity.
2) The selection and randomization processes may result in an ordered list of available children for an identified activity. If a process does result in a list of available children, that list replaces the activity's current set of *Available Children* (AM.1.1).
3) The selection process must be invoked prior to the randomization process, if both apply at the same time.
4) The children in an activity may not be selected or randomized if the activity or any of its descendents are currently active (*Activity is Active* – AM.1.1) or suspended (*Activity is Suspended* – AM.1.1).
5) *Available Children* must persist, unaltered, for the duration the activity is active or suspended.
6) Unless it is specified, through *Selection Timing* (SM.9) or *Randomization Timing* (SM.10), that the selection or randomization process should be applied on each new attempt, the list of *Available Children* must persist as long as there is a possibility of another attempt on the activity, even if the activity is no longer active, regardless of whether the activity is suspended or not. Typically this will be the duration of the attempt on the entire activity tree.
   Note: If the activity tree is dynamic, the selection and randomization processes should be applied to an activity each time the activity's descendents are altered.
7) In a directed traversal of the activity tree, through the *Flow Subprocess* (SB.3), any activity for which *Selection Timing* or *Randomization Timing* is specified, must have the selection and/or randomization processes invoked prior to the activity's children being traversed, in accordance to the specified timing. For such an activity, the traversal only applies the activity's list of *Available Children*, in the order defined by the list.
   Note: The selection and/or randomization process must take place whether or not the activity is activated but before its children can be traversed; the process must not be applied to an activity whose children are not being traversed.
8) A *Choice* sequencing request, through the *Choice Sequencing Request Process* (SB.9), may target an activity that is a descendent of an activity for which selection or randomization is required, only if the selection and/or randomization processes have been applied as required for every ancestor of the target activity.

The selection and randomization processes are controlled by parts of the sequencing definition model:

- *Selection Controls* – controls how activities are selected.
- *Randomization Controls* – controls how activities are reordered.

The selection and randomization processes do not require data from the tracking model.

The selection and randomization processes use data from the activity state model.

- *Activity State Information* – information about the results of the learner's state for an activity.

The selection and randomization processes may change the *Available Children* (AM.1.1) attribute for activities processed.

The selection and randomization behavior is defined in terms of two processes:

- *Select Children Process* – selects some number of an activity's children.
- *Randomize Children Process* – reorders the available children of an activity.

SR.1 Select Children Process

Sequencing selects child activities of an activity prior to selecting the activity to deliver to the learner. The process is applied to an activity's children.

The sequencing definition attributes *Selection Controls* control activity selection. The process is defined as the *Select Children Process*.

- From the current set of all child activities, select some number of them, without replacement, retaining relative order.

The *Select Children Process* is applied to the identified activity's children. It does not return any value. It may change the *Available Children* (AM.1.2) for the identified activity.

The *Select Children Process* is specified by the following pseudo code. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Select Children Process* only describes the expected behavior that an implementation will exhibit.

Select Children Process (for an activity; may change the *Available Children* for the activity):

Reference:	**Section:**
Activity is Active	AM.1.1
Activity is Suspended	AM.1.1
Available Children	AM.1.1
Activity Progress Status	TM.1.2.1
Selection Count	SM.9
Selection Count Status	SM.9
Selection Timing	SM.9

1.	**If** the activity does not have children **Then**	Cannot apply selection to a leaf activity.
1.1.	**Exit** *Select Children Process*	
	End If	
2.	**If** *Activity is Suspended* for the activity is *True* **Or** the *Activity is Active* for the activity is *True* **Then**	Cannot apply selection to a suspended or active activity.
2.1.	**Exit** *Select Children Process*	
	End If	
3.	**Case:** the *Selection Timing* for the activity is *Never*	
3.1.	**Exit** *Select Children Process*	
	End Case	
4.	**Case:** the *Selection Timing* for the activity is *Once*	
4.1.	**If** the *Activity Progress Status* for the activity is *False* **Then**	If the activity has not been attempted yet.
4.1.1.	**If** the *Selection Count Status* for the activity is *True* **Then**	
4.1.1.1.	Initialize child list as an *Empty* ordered list	
4.1.1.2.	**Iterate** *Selection Count*, for the activity, times	
4.1.1.2.1.	Randomly select (without replacement) an activity from the children of the activity	
4.1.1.2.2.	Add the selected activity to the child list, retaining the original (from the activity) relative order of activities	
	End Iterate	

4.1.1.3.	Set *Available Children* for the activity to the child list	
	End If	
	End If	
4.2.	**Exit** *Select Children Process*	
	End Case	
5.	**Case:** the *Selection Timing* for the activity is *On Each New Attempt*	
5.1.	**Exit** *Select Children Process*	Undefined behavior.
	End Case	
6.	**Exit** *Select Children Process*	Undefined timing attribute.

SR.2 Randomize Children Process

Randomizes the available child activities of a activity. The process is applied to an activity's children.

The sequencing definition attributes *Randomize Controls* controls randomization. The process is defined as the *Randomize Children Process*.

- Randomly reorder the activity's available children.

The *Randomize Children Process* is applied to the identified activity's available children. It does not return any value. It may change the *Available Children* (AM.1.2) for the identified activity.

The *Randomize Children Process* is specified by the following pseudo code. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Randomize Children Process* only describes the expected behavior that an implementation will exhibit.

Randomize Children Process (for an activity; may change the *Available Children* for the activity):

Reference:	Section:
Activity is Active	AM.1.1
Activity is Suspended	AM.1.1
Available Children	AM.1.1
Activity Progress Status	TM.1.2.1
Randomize Children	SM.10
Randomization Timing	SM.10

1.	**If** the activity does not have children **Then**	Cannot apply randomization to a leaf activity.
1.1.	**Exit** *Randomize Children Process*	
	End If	
2.	**If** *Activity is Suspended* for the activity is *True* **Or** the *Activity is Active* for the activity is *True* **Then**	Cannot apply randomization to a suspended or active activity.
2.1.	**Exit** *Randomize Children Process*	
	End If	
3.	**Case:** the *Randomization Timing* for the activity is *Never*	
3.1.	**Exit** *Randomize Children Process*	
	End Case	

4.	**Case:** the *Randomization Timing* for the activity is *Once*	
4.1.	**If** the *Activity Progress Status* for the activity is *False* **Then**	If the activity has not been attempted yet.
4.1.1.	**If** *Randomize Children* for the activity is *True* **Then**	
4.1.1.1.	Randomly reorder the activities contained in *Available Children* for the activity	
	End If	
	End If	
4.2.	**Exit** *Randomize Children Process*	
	End Case	
5.	**Case:** the *Randomization Timing* for the activity is *On Each New Attempt*	
5.1.	**If** *Randomize Children* for the activity is *True* **Then**	
5.1.1.	Randomly reorder the activities contained in *Available Children* for the activity	
	End If	
5.2.	**Exit** *Randomize Children Process*	
	End Case	
6.	**Exit** *Randomize Children Process*	Undefined timing attribute.

SB. Sequencing Behavior Model

The Simple Sequencing process is driven in part by a “sequencing request” – a request to identify an element of content for eventual delivery to the learner. The Simple Sequencing process evaluates the request in terms of the content model described by the activity tree, the learner’s current location in the content experience, and determines what content object should be delivered to the learner. This is essentially a mapping or resolution process that converts a sequencing request into the identification of a content object. The process of evaluating the request and identifying the content object (or returning an error) is called the “sequencing process”.

The sequencing process makes no assumptions as to how or when a sequencing request is generated. The sequencing process must insure that sequencing requests are valid. Additionally, the sequencing process makes no assumptions about how or when the identified content object is actually delivered to the learner. In most cases, a delivery request for the identified content object will be immediately delivered to the learner. As part of identifying the content object, the sequencing process uses elements of the tracking model and overall activity tree structure.

The overall sequencing process (see Overall Sequencing Process **OP**) relates the sequencing process to the navigation, termination, rollup, selection and randomization, and delivery processes.

The sequencing process is controlled by parts of the sequencing definition model:

- *Sequencing Control Modes* – controls for types of sequencing requests that may apply to a collection of activities.
- *Sequencing Rule Definitions* – rules, applied to activities, that are used to specify sequencing behaviors for the activity.
- *Limit Conditions* – limits on how many times, how long and when an activity is allowed.
- *Objective Description* – the learning objectives associated with an activity.

The sequencing process uses all parts of the tracking model:

- *Objective Information* – information about the results of the learner’s interactions related to an objective.
- *Activity / Attempt Progress Information* – information about a learner’s *attempt* at an activity.

The sequencing process uses data from the activity state model:

- *Activity State Model* – information about the results of the learner’s state for an activity and the global state of the activity tree.

The behavior of the sequencing process is defined in terms of seven processes (one for each sequencing request), an overall sequencing process and seven associated supporting processes:

- *Sequencing Rules Check Process* – applies sequencing rules to determine if some sequencing rule action should be performed.
- *Check Activity Process* – applies limit conditions and disabled rules to determine if the activity is allowed for sequencing
- *Flow Tree Traversal Subprocess* – determines the next activity in a preorder traversal of the activity tree.
- *Flow Activity Sequencing Subprocess* – determines if a single activity should be delivered by checking limit conditions and sequencing rules, optionally traversing the activity tree to the next activity.
- *Flow Subprocess* – traverses the activity tree in a specified direction to find the next activity allowed for delivery.
- *Check Activity Process* – determines if a single activity is disabled or violates limit conditions.
- *Choice Activity Sequence Subprocess* – determines if a *Choice* sequencing request is permitted for a single activity by checking limit conditions and sequencing rules.
- *Start Sequencing Request Process* – processes a *Start* sequencing request.
- *Resume All Sequencing Request Process* – processes a *Resume All* sequencing request.
- *Continue Sequencing Request Process* – processes a *Continue* sequencing request.
- *Previous Sequencing Request Process* – processes a *Previous* sequencing request.

- *Choice Sequencing Request Process* – processes a *Choice* sequencing request.
- *Retry Sequencing Request Process* – processes a *Retry* sequencing request.
- *Exit Sequencing Request Process* – processes an *Exit* sequencing request.

These individual processes are part of the overall *Sequencing Request Process* that controls all sequencing behavior.

The sequencing process traverses the activity tree, in a forward (or reverse) preorder traversal. Tree traversal continues until the traversal identifies a termination point or an activity to deliver. Identifying the activity to deliver is always constrained by limit conditions.

SB.1 Sequencing Requests

The sequencing process responds to one of a set of different sequencing requests. A sequencing request defines the actions to be performed when the request is processed. The request names are tokens in a vocabulary. The names have no semantics or meanings themselves. The definition of the action is the complete definition of the required behavior.

Sequencing Request	Action
Start	Start the sequencing process at the root of the activity tree. Perform a forward preorder tree traversal and evaluate the activities: • If the activity does not have a “Flow” sequencing control mode (value *False*), exit without identifying an activity to deliver – the sequencing system waits for another navigation request. • If the tree traversal reaches a leaf activity, the activity is the candidate to deliver to the learner.
Resume All	Resume the activity where the previous *Suspend All* termination request was processed. The activity to be resumed is retained in the Activity State attribute of *Suspended Activity*. • If the Activity State attribute value of *Suspended Activity* is a defined activity, the attempt on that activity (and all ancestor activities) should be resumed, and the *Suspended Activity* is the candidate to deliver to the learner. • If the Activity State attribute value of *Suspended Activity* is undefined, exit without identifying an activity to deliver – the sequencing system waits for another navigation request.
Continue	A Continue sequencing request is processed only if the “Flow” sequencing control mode (value “True”) is enabled for the parent of the current activity; otherwise, this sequencing request has no effect. Start the sequencing process at the current activity. Perform a forward preorder tree traversal and evaluate the activities: • If the activity does not have a “Flow” sequencing control mode (value “False”), exit without identifying an activity to deliver – the sequencing system waits for another navigation request. • If the tree traversal reaches a leaf activity, the activity is the candidate to deliver to the learner.
Previous	A Previous sequencing request is processed only if a “Flow” sequencing control mode is enabled (value “True”) for the parent of the current activity; otherwise, this sequencing request has no effect. A Previous sequencing request is processed only if the “Forward Only” sequencing control mode for the parent of the current activity is “False”. Start the sequencing process at the current activity. Perform a *reverse* preorder tree traversal and evaluate the activities: • If the activity does not have a “Flow” sequencing control mode (value “False”), exit without identifying an activity to deliver – the sequencing system waits for another navigation request. • If the tree traversal reaches a leaf activity, the activity is the candidate to deliver to the learner.
Choice	A “target” of the request, the activity that is requested as the next activity in the sequence, accompanies this sequencing request. If the parent of the target activity does not have a “Choice” sequencing control mode (value “True”), exit without identifying an activity to deliver – the sequencing system waits for another navigation request. Find the first common ancestor activity of the current activity (or the root if there is no current activity) and the target activity. Form a path from the current activity to the common ancestor activity and then to the target activity. If any activity along the path from the parent of the current activity to the common ancestor, but not including the common ancestor, does not have a “Choice Exit” sequencing control mode (value “True”), exit without identifying an activity to deliver – the sequencing system waits for another navigation request. Start the sequencing process at the current activity. Traverse the path from the current activity to the target activity and evaluate the activities (each step in the traversal may be *forward* or *reverse* relative to a preorder tree traversal): • If traversal is in the *forward* direction, and a *Stop Forward Traversal* rule for the activity evaluates to “True”, exit without identifying an activity to deliver – the sequencing system waits for another navigation request. • If traversal is in the *reverse* direction, and a *Forward Only* rule for the activity evaluates to “True”, exit without identifying an activity to deliver – the sequencing system waits for another navigation request. • If the traversal reaches the target activity, the target activity is the candidate to deliver to the learner.
Retry	The current activity is identified as the activity to deliver (recorded as a new attempt).

Sequencing Request	Action
Exit	Start the sequencing process at the current activity. • If the current activity is the root of the activity tree, exit sequencing. Otherwise, the sequencing system waits for another navigation request.

SB.2 Sequencing Behavior

The sequencing behavior describes how a sequencing processor interprets a sequencing request in combination with the elements of the sequencing definition model and with instance data from the tracking model and activity state model to validate the sequencing request and identify the content object for delivery. Except when the sequencing request is *Exit*, the sequencing process attempts to determine an activity that can be delivered to the learner.

An implementation must be capable of representing the processes described and have the implemented process exhibit the behavior described. There are no additional requirements on implementing the sequencing behavior model.

The sequencing behavior relies on the data descriptions from the sequencing definition model (see Sequencing Definition Model **SM**), the tracking model (see Tracking Model **TM**), and the activity state model (see Activity State Model **AM)**. These information models also specify default data values that govern the access to activity or tracking data.

SB.2.1 Flow Tree Traversal Subprocess

Sequencing performs a preorder traversal of the activity tree. Different sequencing requests traverse either forward or backward through the activity tree. The *Flow Tree Traversal Subprocess* is used to determine the *next* activity in traversal order, starting at a particular activity (or the *previous* activity if traversing in the backward direction).

Traversal is controlled by sequencing rules and Control Modes. Sequencing rule conditions are specified by the Sequencing Rule Descriptions (SM.2). The process uses the *Sequencing Rules Check Subprocess* (UP.2).

The traversal process is defined as a rule-checking and limit-checking traversal of the activity tree.

- The *Flow Tree Traversal Subprocess* traverses the tree, either forward or backward, from the specified activity to identify the next activity (one single step in the tree traversal). Traversal beyond the end of the tree (in forward traversal) or the start of the tree (in backward traversal) is an exception. Only precondition sequencing rules related to traversing the tree in a *Flow* control mode are applied.

The *Flow Tree Traversal Subprocess* is applied to the identified activity in the specified direction. If successful, it returns the *Next* activity in the tree in traversal order.

The *Flow Tree Traversal Subprocess* is specified by the following pseudo code. The pseudo code describes only the tree traversal logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Flow Tree Traversal Subprocess* only describes the expected behavior that an implementation will exhibit.

Flow Tree Traversal Subprocess (for an activity, a traversal direction, a consider children flag, and a previous traversal direction; returns the ‘next’ activity in directed traversal of the activity tree)**:**		
	Reference:	**Section:**
	Available Children	AM.1.1
	Flow Activity Traversal Subprocess	SB.2.2
	Sequencing Control Forward Only	SM.1
	Sequencing Rules Check Process	UP.2
1.	**If** the activity is not the root activity of the activity tree **Then**	
1.1.	**If** *Sequencing Control Flow* for the parent of the activity is *False* **Then**	Confirm a ‘flow’ traversal is allowed from the activity.

1.1.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	
	End If	
	End If	
2.	**If** (previous traversal direction is *Defined* **And** is *Backward*) **And** the activity is the last activity in the activity's parent's list of *Available Children* **Then**	Test if we have skipped all of the children in a forward only cluster moving backward
2.1.	traversal direction is *Backward*	
2.2.	activity is the first activity in the activity's parent's list of *Available Children*	
	End If	
3.	**If** the traversal direction is *Forward* **Then**	
3.1.	**If** the activity is the last activity in a forward preorder tree traversal of the activity tree **Then**	Cannot walk off the activity tree.
3.1.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	
	End If	
3.2.	**If** the activity is a leaf **Or** consider children is *False* **Then**	
3.2.1.	**If** the activity is the last activity in the activity's parent's list of *Available Children* (AM.1.1) **Then**	
3.2.1.1.	Apply the *Flow Tree Traversal Subprocess* to the activity's parent in the *Forward* direction and a previous traversal direction of *n/a* with consider children equal to *False*	Recursion – Move to the activity's parent's next forward sibling.
3.2.1.2.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: the result of the recursive *Flow Tree Traversal Subprocess*)	Return the result of the recursion.
3.2.2.	**Else**	
3.2.2.1.	Apply the *Sequencing Rules Check Process* to the activity and the *Stop Forward Traversal sequencing rules*	Make sure we can move past the activity.
3.2.2.2.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	
3.2.2.2.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	
	End If	
3.2.2.3.	Traverse the tree, forward preorder, one activity to the next activity, in the activity's parent's list of *Available Children*	
3.2.2.4.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *the activity identified by the traversal*)	
	End If	
3.3.	**Else**	Entering a cluster
3.3.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: the first activity in the activity's list of *Available Children*)	
	End If	
	End If	
4.	**If** the traversal direction is *Backward* **Then**	
4.1	**If** the activity is the root activity of the tree **Then**	Cannot walk off the root of the activity tree.
4.1.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	
	End If	

4.2.	**If** the activity is a leaf **Or** consider children is *False* **Then**	
4.2.1.	**If** the activity is the first activity in the activity's parent's list of *Available Children* **Then**	
4.2.1.1.	Apply the *Flow Tree Traversal Subprocess* to the activity's parent in the *Backward* direction and a previous traversal direction of *n/a* with consider children equal to *False*	Recursion – Move the activity's parent's next backward sibling.
4.2.1.2.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: the result of the recursive *Flow Tree Traversal Subprocess*)	Return the result of the recursion.
4.2.2.	**Else**	
4.2.2.1.	**If** *Sequencing Control Forward Only* for the parent of the activity is *True* **Then**	Test the control mode before traversing.
4.2.2.1.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	
	End If	
4.2.2.2.	Traverse the tree, reverse preorder, one activity to the previous activity, from the activity's parent's list of *Available Children*	
4.2.2.3.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *the activity identified by the traversal*)	
	End If	
4.3.	**Else**	Entering a cluster
4.3.1.	**If** *Sequencing Control Forward Only* for the parent of the activity is *True* **Or** the activity is the root of the activity tree **Then**	
4.3.1.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: the first activity in the activity's list of *Available Children*)	Start at the beginning of a forward only cluster.
4.3.2.	**Else**	
4.3.2.1.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: the last activity in the activity's list of *Available Children*)	Start at the end of the cluster if we are backing into it.
	End If	
	End If	
	End If	
5.	**Exit** *Flow Tree Traversal Subprocess* (**Next Activity**: *Nil*)	

SB.2.2 Flow Activity Traversal Subprocess

Sequencing traverses the activity tree, applying sequencing rules and limit conditions. The *Flow Activity Traversal Subprocess* is used to determine if an activity should be delivered, or if the traversal should examine the next activity in the tree. The process is applied to a single activity, checking both limit conditions and precondition sequencing rules.

Traversal is controlled by sequencing rules and Control Modes. Sequencing rule conditions are specified by the *Sequencing Control Modes* (SM.1), the *Sequencing Rule Descriptions* (SM.2), and the Limit Conditions Description (SM.3). The process uses the *Check Process* (UP.4), the *Sequencing Rules Check Subprocess* (UP.2), and the *Flow Tree Traversal Subprocess* (SB.2.1).

The process is defined as a rule- and limit-checking traversal of the activity tree.

- The process traverses the entire tree, in a preorder traversal, from the particular activity. If *Sequencing Control Mode Flow* is not enabled for the parent of the activity, stop without identifying an activity to deliver. Check the *Limit Conditions* for the activity and stop if any limits are exceeded. Stop if the activity is disabled (SM.2 – *Disabled*). Skip activities marked to be skipped (SM.2 – *Skipped*), and traverse to the next activity in the specified direction. If the activity is not a leaf, traverse the tree to find the next activity.

The *Flow Activity Traversal Subprocess* is applied to the identified activity in the specified direction. If successful, it returns the *Next* deliverable activity in the tree in traversal order. If unsuccessful, it returns an undeliverable activity indicating where the ‘flow’ failed.

The *Flow Activity Traversal Subprocess* is specified by the following pseudo code. The pseudo code describes only the activity sequencing logic. How this process is implemented or information is encoded, stored, represented, or bound is outside the scope of this specification. The *Flow Activity Traversal Subprocess* only describes the expected behavior that an implementation will exhibit.

Flow Activity Traversal Subprocess (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):

Reference:	**Section:**
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)	
	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)	
3.3.	**Else**	
3.3.1.	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.	**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)	
	End If	
6.	**If** the activity is not a leaf node in the activity tree **Then**	
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*	Cannot deliver a non-leaf activity; enter the activity looking for a child.

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)	
6.3.	**Else**	
6.3.1.	**If** *Sequencing Control Forward Only* for the parent of the activity identified by the *Flow Tree Traversal Subprocess* is *True* **And** the traversal direction is *Backward* **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*	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*	Make sure the identified activity is OK.
	End If	
	End If	
	End If	
7.	**Exit** *Flow Activity Traversal Subprocess* (**Deliverable**: *True*; **Next Activity**: the activity)	

SB.2.3 Flow Subprocess

The *Flow Subprocess* traverses the activity tree from a designated activity in a specified direction. It returns the activity where the traversal stopped and True if the activity is allowed for delivery.

The process can be described as a tree traversal. Start at the *Current Activity* (AM.1.2), use the *Flow Tree Traversal Subprocess* (SB.2.1) to go forward to the next activity. Exit if the activity should not be delivered. Otherwise, traverse the activity tree. Use the *Flow Activity Sequencing Subprocess* (SB.2.2) to determine if the traversal should terminate, because either the activity should or should not be delivered. Otherwise, continue the traversal with the next activity in tree traversal order.

The process traverses the entire tree, in a preorder traversal, from the particular activity. The traversal is performed by applying the *Flow Activity Traversal Subprocess* and the *Flow Tree Traversal Subprocess*. It returns the position in the activity tree where the traversal stopped and an indicator describing if that activity can be delivered.

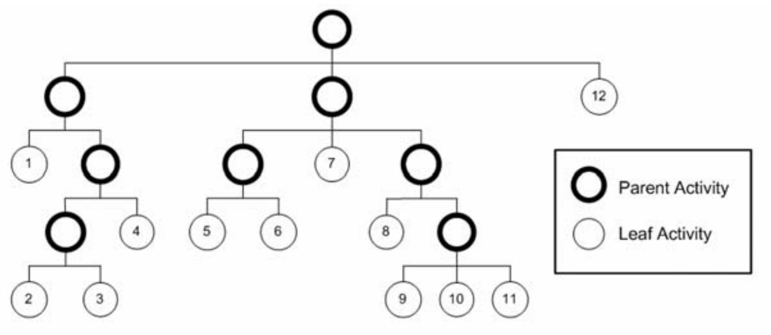

Figure SB.1 - Order of Activity Delivery during a Continue or Previous sequencing request.

The activity tree shown in Figure SB.1 illustrates simple *Forward* or *Backward* activity tree traversals. If the *Sequencing Control Flow* (SM.1) value is *True* for all of the parent activities, and there are no other sequencing definitions, the leaf activities will be encountered in the numeric order shown.

For example, a *Continue* sequencing request will delivery the next leaf activity, in ascending order, e.g., activity 7 will follow activity 6. A *Previous* sequencing request will deliver the preceding leaf activity, in descending order, e.g., activity 4 precedes activity 5.

The *Flow Subprocess* for an activity is specified by the following pseudo code. The pseudo code describes only the traversal logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Flow Subprocess* only describes the expected behavior that an implementation will exhibit.

Flow Subprocess (for an activity, a traversal direction, and a consider children flag; indicates if the flow was successful and at what activity the flow stopped):

Reference:	Section:
Flow Activity Traversal Subprocess	SB.2.2
Flow Tree Traversal Subprocess	SB.2.1

1.	The candidate activity is the activity	
2.	Apply the *Flow Tree Traversal Subprocess* to the candidate activity in the traversal direction and a previous traversal direction of *n/a* with consider children equal to the consider children flag	Attempt to move away from the activity, one activity in the specified direction.
3.	**If** the *Flow Tree Traversal Subprocess* does not identify an activity **Then**	No activity to move to.
3.1.	**Exit** *Flow Subprocess* (**Identified Activity**: candidate activity; **Deliverable**: *False*)	
4.	**Else**	
4.1	candidate activity is the activity identified by the *Flow Tree Traversal Subprocess*	
4.2	Apply the *Flow Activity Traversal Subprocess* to the candidate activity in the traversal direction and a previous traversal direction of *n/a*	Validate the activity and traverse until a valid leaf is encountered.
4.3.	**Exit** *Flow Subprocess* (**Identified Activity**: the activity identified by the *Flow Activity Traversal Subprocess*; **Deliverable**: as identified by the *Flow Activity Traversal Subprocess*)	
	End If	

SB.2.4 Choice Activity Traversal Subprocess

Sequencing traverses the activity tree, applying sequencing rules and limit conditions. The *Choice Activity Traversal Subprocess* is used to determine if a *Choice* sequencing request on an activity is permitted. The process is applied to a single activity, checking both limit conditions and sequencing rules.

Traversal is controlled by sequencing rules and Control Modes. Sequencing rule conditions are specified by the Sequencing Rule Definitions. The process uses the *Check Activity Process* (UP.5) and the *Sequencing Rules Check Process* (UP.2).The process is defined as a rule- and limit-checking process.

- Check the limit conditions for the activity and stop if any limit condition is violated (SM.3). Stop if the activity is disabled (SM.2 – *Disabled*). Stop if traversal is not permitted (if there are *Stop Forward Traversal* (SM.2) rules for traversal in the forward direction or the *Sequencing Control Forward Only* (SM.1) is True for traversal in the backward direction).

The *Choice Activity Traversal Subprocess* is applied to the single identified activity. It returns a Boolean: False if any limit condition or rule is violated; or True the activity can be reached by the traversal.

The *Choice Activity Traversal Subprocess* is specified by the following pseudo code. The pseudo code describes only the checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Choice Activity Traversal Subprocess* only describes the expected behavior that an implementation will exhibit.

Choice Activity Traversal Subprocess (for an activity and a traversal direction; returns True if the activity can be reached):

Reference:	**Section:**
Check Activity Process	UP.5
Sequencing Control Forward Only	SM.1
Sequencing Rules Check Process	UP.2

1.	Apply the *Check Activity Process* to the activity	Make sure the activity is allowed.
2.	**If** the *Check Activity Process* returns *True* **Then**	
2.1.	**Exit** *Choice Activity Traversal Subprocess* (**Reachable**: *False*)	
	End If	
3.	**If** the traversal direction is *Forward* **Then**	
3.1.	Apply the *Sequencing Rules Check Process* to the activity and the *Stop Forward Traversal sequencing rules*	
3.2.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	
3.2.1.	**Exit** *Choice Activity Traversal Subprocess* (**Reachable**: *False*)	
	End If	
3.3.	**Exit** *Choice Activity Traversal Subprocess* (**Reachable**: *True*)	
	End If	
4.	**If** the traversal direction is *Backward* **Then**	
4.1.	**If** the activity has a parent **Then**	
4.1.1.	**If** *Sequencing Control Forward Only* for the parent of the activity is *True* **Then**	
4.1.1.1.	**Exit** *Choice Activity Traversal Subprocess* (**Reachable**: *False*)	
	End If	
4.1.2.	**Else**	
4.1.2.1.	**Exit** *Choice Activity Traversal Subprocess* (**Reachable**: *False*)	Cannot walk backward from the root of the activity tree.
	End If	
4.2.	**Exit** *Choice Activity Traversal Subprocess* (**Reachable**: *True*)	
	End If	

SB.2.5 Start Sequencing Request Process

A *Start* sequencing request is used to identify the first activity to deliver. The *Start Sequencing Request Process* traverses the activity tree, applying sequencing rules and limit conditions to identify the first activity to deliver.

The sequencing process uses the *Flow Subprocess* (SB.2.3). It does not explicitly use any sequencing definitions or tracking model data.

The sequencing request process is defined as a rule-checking and limit-checking traversal of the activity tree.

- The process traverses the entire tree, in a forward preorder traversal, from the root using the *Flow Activity Sequencing Subprocess.* At any activity, if *Sequencing Control Flow* (SM.1) is not enabled for the parent, stop without identifying an activity to deliver. Check the *Limit Conditions* (SM.3) for the activity and stop if any limits are exceeded. Stop if the traversal reaches an activity that is disabled (SM.2 – *Disabled*). Skip (and traverse *Forward*) activities marked to be skipped (SM.2 – *Skipped*). If the activity is not a leaf, traverse the tree to find the next activity. The first leaf activity found is the candidate activity to deliver.

The process can be described as a tree traversal. Start at the root, use the *Flow Subprocess* to determine if which activity should be delivered, if any.

The *Start Sequencing Request Process* either validates the sequencing request and returns the candidate activity to deliver or returns an indication that the there is no activity to deliver.

The *Start Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the activity sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Start Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Start Sequencing Request Process (may return a delivery request):		
	Reference: Current Activity / Flow Subprocess	**Section:** AM.1.2 / SB.2.3
1.	**If** the *Current Activity* is *Defined* **Then**	Make sure the sequencing session has not already begun.
1.1	**Exit** *Start Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
2.	Apply the *Flow Subprocess* to the root of the activity tree in the *Forward* direction with consider children equal to *True*	Attempt to flow into the activity tree.
3.	**If** the *Flow Subprocess* returns *False* **Then**	
3.1.	**Exit** *Start Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
4.	**Else**	
4.1.	**Exit** *Start Sequencing Request Process* (**Delivery Request**: the activity identified by the *Flow Subprocess*)	
	End If	
	End If	

SB.2.6 Resume All Sequencing Request Process

A *Resume All* sequencing request is used to identify the first activity to deliver by attempting to resume the previous sequencing session. The *Resume All Sequencing Request Process* traverses the activity tree, applying sequencing rules and limit conditions, through the *Check Activity Process* (UP.5), to identify the first activity to deliver.

The sequencing process uses the *Suspended Activity* (AM.1.2) attribute to construct a path from the root of the activity tree to the activity where a *Suspend All* navigation request was triggered in the previous sequencing session.

The sequencing request process is defined as a rule-checking and limit-checking traversal of the activity tree.

- The process traverses the entire tree, in a forward preorder traversal, from the root, along a unique path toward the activity designated by the *Suspended Activity*. While traversing the path, *Sequencing Control Modes* (SM.1) are ignored. Check the *Limit Conditions* (SM.3) for each activity and stop if any limits are exceeded. Stop if the traversal reaches an activity that is disabled (SM.2 – *Disabled*).

The *Resume All Sequencing Request Process* either validates the sequencing request and returns the candidate activity to deliver or returns an indication that the there is no activity to deliver.

The *Resume All Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the activity sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Resume All Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Resume All Sequencing Request Process (may return a delivery request):		
	Reference: Current Activity Suspended Activity	**Section:** AM.1.2 AM.1.2
1.	**If** the *Current Activity* is *Defined* **Then**	Make sure the sequencing session has not already begun.
1.1.	**Exit** *Resume All Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
2.	**If** the *Suspended Activity* is not *Defined* **Then**	Make sure there is something to resume.
2.1.	**Exit** *Resume All Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
3.	**Exit** *Resume All Sequencing Request Process* (**Delivery Request**: the activity identified by the *Suspended Activity*)	The Delivery Request Process validates that the Suspended Activity can be delivered.

SB.2.7 Continue Sequencing Request Process

A *Continue* sequencing request is used to identify the next activity to deliver starting at the current activity, traversing the activity tree forward. The *Continue Sequencing Request Process* traverses the activity tree, applying sequencing rules and limit conditions to identify the next activity to deliver.

The sequencing process uses the *Flow Subprocess* (SB.2.3). It does not explicitly use any sequencing definitions or tracking model data.

The sequencing request process is defined as a rule- and limit-checking traversal of the activity tree.

- The process traverses the entire tree, in a forward preorder traversal, from the *Current Activity* (AM.1.2) using the *Flow Subprocess.* As the traversal proceeds, each encountered activity is considered a candidate for delivery. At any activity, if *Sequencing Control Flow* (SM.1) is not enabled for the activity's parent, stop without identifying an activity to deliver. Check the *Limit Conditions* (SM.3) for the activity and stop if any limits are exceeded. Stop if the traversal reaches an activity that is disabled (SM.2 – *Disabled*). Skip (and traverse *Forward*) activities marked to be skipped (SM.2 – *Skipped*). If the activity is not a leaf, traverse the tree to find the next activity. The first leaf activity found is the candidate activity to deliver. See Figure SB.1.

The *Continue Sequencing Request Process* either validates the sequencing request and returns the candidate activity to deliver or returns an indication that the there is no activity to deliver.

The *Continue Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the activity sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Continue Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Continue Sequencing Request Process (may return a delivery request):		
	Reference: Current Activity Flow Subprocess	**Section:** AM.1.2 SB.2.3
1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun.
1.1.	**Exit** *Continue Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
2.	Apply the *Flow Subprocess* to the *Current Activity* in the *Forward* direction with consider children equal to *False*	Flow in a forward direction to the next allowed activity.
3.	**If** the *Flow Subprocess* returns *False* **Then**	
3.1.	**Exit** *Continue Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
4.	**Else**	
4.1.	**Exit** *Continue Sequencing Request Process* (**Delivery Request**: the activity identified by the *Flow Subprocess*)	
	End If	

SB.2.8 Previous Sequencing Request Process

A *Previous* sequencing request is used to identify the next activity to deliver starting at the current activity, traversing the activity tree backward. The *Previous Sequencing Request Process* traverses the activity tree, applying sequencing rules and limit conditions to identify the next activity to deliver.

The sequencing process uses the *Flow Subprocess* (SB.2.3). It does not explicitly use any sequencing definitions or tracking model data.

- The process traverses the entire tree, in a reverse preorder traversal, from the *Current Activity* (AM.1.2) using the *Flow Subprocess*. As the traversal proceeds, each encountered activity is considered a candidate for delivery. At any activity, if *Sequencing Control Flow* (SM.1) is not enabled for the activity's parent, stop without identifying an activity to deliver. Check the *Limit Conditions* (SM.3) for the activity and stop if any limits are exceeded. Stop if the traversal reaches an activity that is disabled (SM.2 – *Disabled*). Skip (and traverse *Backward*) activities marked to be skipped (SM.2 – *Skipped*). If the children of an activity with *Sequencing Control Forward Only* (SM.1) are considered, traverse those activities starting at the first child in a *Forward* direction. If the activity is not a leaf, traverse the tree to find the next (i.e., previous) activity. The first leaf activity found is the candidate activity to deliver. See Figure SB.1.

The *Previous Sequencing Request Process* either validates the sequencing request and returns the candidate activity to deliver or returns an indication that the there is no activity to deliver.

The *Previous Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the activity sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Previous Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Previous Sequencing Request Process (may return a delivery request):		
	Reference: Current Activity, Flow Subprocess	**Section:** AM.1.2, 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*)	Nothing to deliver.
	End If	
2.	Apply the *Flow Subprocess* to the *Current Activity* (AM.1.2) in the *Backward* direction with consider children equal to *False*	Flow in a backward direction to the next allowed activity.
3.	**If** the *Flow Subprocess* returns *False* **Then**	
3.1.	**Exit** *Previous Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
4.	**Else**	
4.1.	**Exit** *Previous Sequencing Request Process* (**Delivery Request**: the activity identified by the *Flow Subprocess*)	
	End If	

SB.2.9 Choice Sequencing Request Process

A *Choice* sequencing request specifies the next activity to deliver. The *Choice Sequencing Request Process* traverses the activity tree, applying sequencing rules and limit conditions to verify the specified next activity to deliver.

The sequencing process uses the *Sequencing Rules Check Process* (UP.2), the *Check Activity Process* (UP.5), and the *Flow Subprocess* (SB.2.3). It does not explicitly use any sequencing definitions or tracking model data.

The sequencing request process is defined as a rule- and limit-checking traversal of the activity tree.

- The process requires a target activity, i.e., the target of the choice. The target must either be the root of the activity tree or the parent of the target activity must have *Sequencing Control Choice* (SM.1) enabled. The target and all of its ancestors must have *Hidden From Choice* (SM.2) disabled. Traverse the path from the *Current Activity* (AM.1.2) to the target activity and verify that the sequencing rules and limit conditions hold, through the *Check Activity Process*, for all activities along the path. There are five ordered cases. When processing a *Choice* sequencing request, the first case that applies is acted upon.

1) The *Current Activity* and target activity are the same: Verify that the target activity can be delivered. See Figure SB.2.

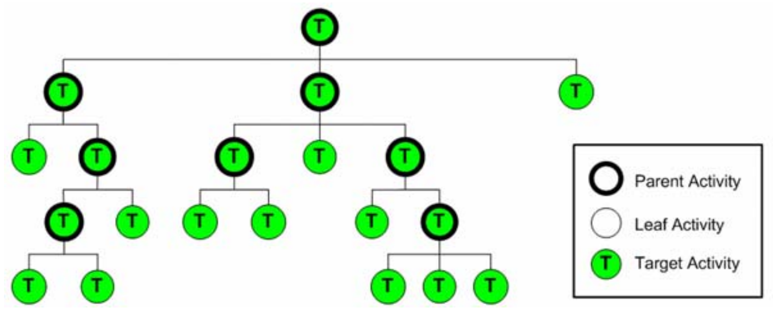

Figure SB.2 - Choice Sequencing Request Case 1.

The activity tree shown in Figure SB.2 illustrates *Choice* sequencing request processing, when the *Current Activity* is the target activity (Case 1). If *Sequencing Control Choice* is *True* for the of the activity's parent, *Sequencing Control Choice Exit* (SM.1) is *True* for all of the ancestor activities, and there are no other sequencing definitions that prevent an activity from being the target of a *Choice* sequencing request, all of the "target" activities (labeled **T**) are valid targets for the sequencing request.

2) The *Current Activity* and the target activity are siblings: Traverse the tree from the *Current Activity* toward the target activity (either *Forward* or *Backward*), applying sequencing rules and limit conditions by using the *Choice Activity Sequence Subprocess* (SB.2.4) to validate each activity on the path. See Figure SB.3.

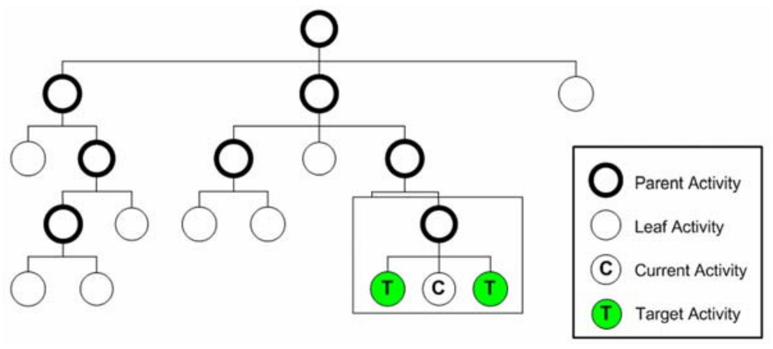

Figure SB.3 - Choice Sequencing Request Case 2.

The activity tree shown in Figure SB.3 illustrates *Choice* sequencing request processing, when the *Current Activity* and the target activity are siblings (Case 2). If *Sequencing Control Choice* is *True* for the parent activity shown in the box (there may be other or different sequencing definitions for other activities), any sibling of the *Current Activity* is a valid "target" activity (labeled **T**) for the sequencing request.

If *Sequencing Control Forward Only* (SM.1) is *True* for the parent activity shown in the box, a sequencing request that targets the predecessor of the current activity will be rejected. If the current activity has a sequencing rule that includes a *Stop Forward Traversal* (SM.2) precondition that evaluates to *True*, a sequencing request that targets the successor of the current activity will be rejected.

3) The target activity is a descendent of the *Current Activity* (move down the tree): Traverse the tree in the *Forward* direction from the *Current Activity* to the target activity, applying sequencing rules and limit conditions by using the *Choice Activity Traversal Subprocess* to validate each activity on the path. See Figure SB.4.

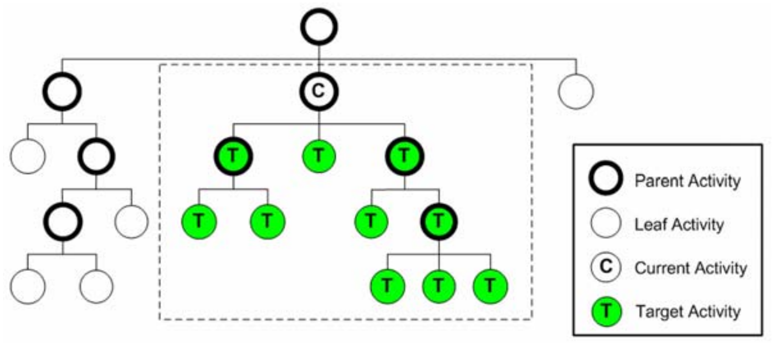

Figure SB.4 - Choice Sequencing Request Case 3.

The activity tree shown in Figure SB.4 illustrates *Choice* sequencing request processing, when the target activity is a descendent of the current activity (Case 3). If *Sequencing Control Choice* (SM.1) is *True* for all of the parent activities in the box, and there are no other sequencing definitions that prevent an activity from being the target of a *Choice* sequencing request, all of the "target" activities (labeled **T**) are valid targets for the sequencing request.

4) The target activity is an ancestor of the *Current Activity* (move up the tree): Traverse the tree in the *Backward* direction from the *Current Activity* to the target activity, applying sequencing rules and limit conditions by using the *Check Activity Process* to validate each activity on the path. See Figure SB.5.

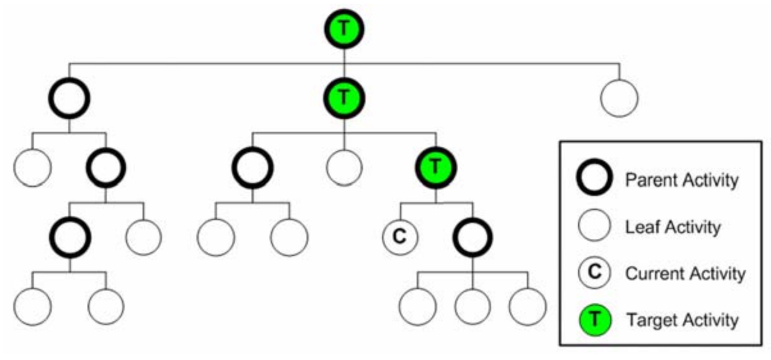

Figure SB.5 -Choice Sequencing Request Case 4.

The activity tree shown in Figure SB.5 illustrates *Choice* sequencing request processing, when the target activity is an ancestor of the *Current Activity* (Case 4). If *Sequencing Control Choice* is *True* for all of the parent of the *Current Activity*, *Sequencing Control Choice Exit* is *True* for all of the ancestors of the *Current Activity*, and there are no other sequencing definitions that prevent an activity from being the target of a *Choice* sequencing request, all of the “target” activities (labeled **T**) are valid targets for the sequencing request.

To process the request, the activity tree is traversed (backward) from the *Current Activity* toward the target, ignoring the value of *Sequencing Control Forward Only*. The target activity is always a parent. If *Sequencing ControlFlow* for the target activity is *False*, nothing is delivered. If *Sequencing Control Flow* for the target activity is *True*, then the request processing continues as a *Continue* sequencing request from the target activity.

5) The target activity is a descendent of the common ancestor (move up to the common ancestor, down to the target): Traverse the tree backward from the *Current Activity* to the common ancestor activity, applying sequencing rules and limit conditions by using the *Check Activity Subprocess* to validate each activity on the path. Traverse the tree from the common ancestor activity to the target activity, applying sequencing rules and limit conditions by using the *Choice Activity Traversal Subprocess* or *Check Activity Process* (depending on the direction the target activity is relative to the *Current Activity*) to validate each activity on the path. See Figure SB.6.

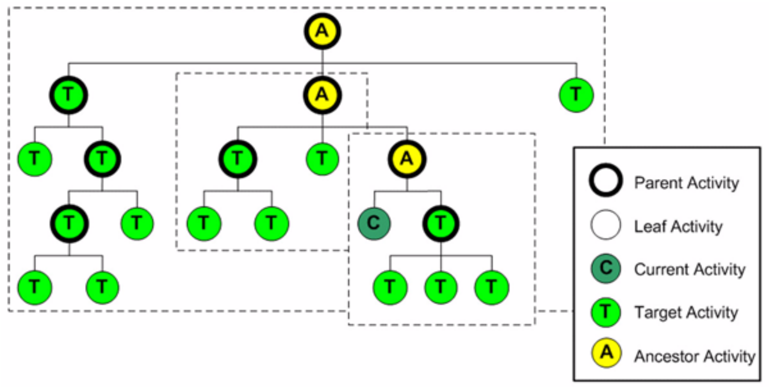

Figure SB.6 - Choice Sequencing Request Case 5.

The activity tree shown in Figure SB.6 illustrates *Choice* sequencing request processing, when the target activity is forward in the activity tree relative to the common ancestor of the *Current Activity* and the target activity (Case 5). If *Sequencing Control Choice* is *True* for the parent of the target activity, *Sequencing Control Mode Choice Exit* is *True* for all of the ancestor activities, and there are no other sequencing definitions that prevent an activity from being the target of a *Choice* sequencing request, all of the “target” activities (labeled **T**) are valid targets for the sequencing request.

Each box in Figure SB.6 contains the set of target activities for one of the common ancestors of the current activity.

To process the request, the activity tree is traversed (backward) from the current activity toward the common ancestor activity, ignoring the value of *Sequencing Control Forward Only*. The activity tree is traversed (forward) from the common ancestor activity to the target activity.

The target activity of choice may be an activity that has children and does not have any resources (a cluster). If the target activity can be reached by the appropriate case above, there are two options:

- The target activity has *Sequencing Control Flow* equal to *True* – The *Flow Subprocess* is invoked from the target activity. Traversal proceeds until a candidate activity is identified for delivery, or the process fails.
- The target activity has *Sequencing Control Flow* equal to *False* – There is no candidate activity to deliver.

The *Choice Sequencing Request Process* either validates the sequencing request and returns the candidate activity to deliver or returns an indication that the there is no activity to deliver.

The *Choice Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the activity sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Choice Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Choice Sequencing Request Process (for a target activity; may return a delivery request; may change the *Current Activity*)

Reference:	**Section:**
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

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*)	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*)	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*)	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*)	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.	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.	Apply the *Check Activity Process* to the *Current Activity*	
8.2.	**If** the *Check Activity Process* returns *True* **Then**	
8.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
8.3.	**Break 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, inclusive	
9.2.	**If** the activity list is *Empty* **Then**	Nothing to choose.
9.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	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*)	Nothing to deliver.
	End If	
	End For	
9.6.	**Break Case**	
	End Case	
10.	**Case:** *Current Activity* and common ancestor are the same Or *Current Activity* is undefined	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, inclusive	
10.2.	**If** the activity path is *Empty* **Then**	
10.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	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*)	Nothing to deliver.
	End If	
	End For	
10.4.	**Break Case**	
	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*)	Nothing to deliver.
	End If	
11.3.	**For** each activity on the activity path	
11.3.1.	Apply the *Check Activity Process* to the activity	Make sure each activity is not disabled or violates limit conditions
11.3.2.	**If** the *Check Activity Process* returns *True* **Then**	
11.3.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
113.3.	**If** the activity is not the last activity in the activity path **Then**	
11.3.3.1.	**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.

11.3.3.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
	End If	
	End For	
11.4.	**Break Case**	
	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 activity, inclusive	
12.2.	**If** the activity path is *Empty* **Then**	
12.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
12.3.	**For** each activity on the activity path	Walk up the tree to the common ancestor.
12.3.1.	Apply the *Check Activity Process* to the activity	Make sure each activity is not disabled or violates limit conditions
12.3.2.	**If** the *Check Activity Process* returns *True* **Then**	
12.3.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
12.3.3.	**If** the activity is not the last activity in the activity path **Then**	
12.3.3.1.	**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.
12.3.3.1.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
	End If	
	End For	
12.4	Form the activity path as the ordered series of activities from the common ancestor to the target activity, inclusive	
12.5.	**If** the activity path is *Empty* **Then**	
12.5.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
12.6.	**If** the target activity is forward in the activity tree relative to the *Current Activity* **Then**	Walk toward the target activity.
12.6.1.	**For** each activity on the activity path	
12.6.1.1.	Apply the *Choice Activity Traversal Subprocess* to the activity in the *Forward* direction	
12.6.1.2.	**If** the *Choice Activity Traversal Subprocess* returns *False* **Then**	
12.6.1.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
	End For	

12.7.	**Else**	
12.7.1.	**For** each activity on the activity path	
12.7.1.1.	Apply the *Check Activity Process* to the activity in the traverse direction	
12.7.1.2.	**If** the *Check Activity Process* returns *True* **Then**	
12.7.1.2.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
	End For	
	End If	
12.8	**Break Case**	
	End Case	
13.	**If** the target activity is a leaf activity **Then**	
13.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: the target activity)	
	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, 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*)	Nothing to deliver.
16.	**Else**	
16.1.	**Exit** *Choice Sequencing Request Process* (**Delivery Request**: for the activity identified by the *Flow Subprocess*)	
	End If	

SB.2.10 Retry Sequencing Request Process

A *Retry* sequencing request is used to restart the current activity with a new attempt.

The *Navigation Process* should have issued a *Termination Request* to the *Termination Request Process* to terminate the activity before the sequencing request is processed

The sequencing process uses the *Check Activity Process.* It does not explicitly use any sequencing definitions or tracking model data.

The sequencing request process is defined as a rule- and limit-checking process.

- The *Retry Sequencing Request Process* checks the status of the current activity, using the *Check Activity Process.* If the current activity is allowed and it is not a leaf, the *Flow Subprocess* is applied to determine which activity should be delivered. If the current activity is a leaf, the current activity as the target activity for the delivery request.

The *Retry Sequencing Request Process* either validates the sequencing request and returns the candidate activity to deliver, returns an indication that the there is no activity to deliver, or returns an error.

The *Retry Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the retry sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Retry Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Retry Sequencing Request Process (may return a delivery request):

Reference:	**Section:**
Activity is Active	AM.1.1
Activity is Suspended	AM.1.1
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** *Retry Sequencing Request Process*	Nothing to deliver.
	End If	
2.	**If** the *Activity is Active* for the *Current Activity* is *True* **Or** the *Activity is Suspended* for the *Current Activity* is *True* **Then**	Cannot retry an activity that is still active or suspended.
2.1.	**Exit** *Retry Sequencing Request Process* (**Delivery Request**: *n/a*)	Nothing to deliver.
	End If	
3.	**If** the *Current Activity* is not a leaf **Then**	
3.1.	Apply the *Flow Subprocess* to the *Current Activity* in the *Forward* direction with consider children equal to *True*	
3.2.	**If** the *Flow Subprocess* returned *False* **Then**	
3.2.1.	**Exit** *Retry Sequencing Request Process*	Nothing to deliver.
3.3.	**Else**	
3.3.1.	**Exit** *Retry Sequencing Request Process* (**Delivery Request**: the activity identified by the *Flow Subprocess*)	
	End If	
4.	**Else**	
4.1.	**Exit** *Retry Sequencing Request Process* (**Delivery Request**: the *Current Activity*)	
	End If	

SB.2.11 Exit Sequencing Request Process

The *Exit* sequencing request is used to exit sequencing when applied to the root of the activity tree. Otherwise the request has no effect.

The *Navigation Process* should have issued a *Termination Request* to the *Termination Request Process* to terminate the activity before the sequencing request is processed.

- An Exit sequencing request does not result in a candidate activity for delivery. If the current activity is the root, sequencing session ends, and control is returned to the LTS.

The *Exit Sequencing Request Process* may return an indication that the sequencing session has ended.

The *Exit Sequencing Request Process* is specified by the following pseudo code. The pseudo code describes only the exit sequencing logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Exit Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Exit Sequencing Request Process (indicates if the sequencing session has ended):		
	Reference: Activity is Active; Current Activity	**Section:** AM.1.1; AM.1.2
1.	**If** the *Current Activity* is **Not** *Defined* **Then**	Make sure the sequencing session has already begun.
1.1.	**Exit** *Exit Sequencing Request Process* (**End Sequencing Session**: *False*)	
	End If	
2.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	Make sure the current activity has already been terminated.
2.1.	**Exit** *Exit Sequencing Request Process* (**End Sequencing Session**: *False*)	
	End If	
3.	**If** the *Current Activity* is the root of the activity tree **Then**	
3.1.	**Exit** *Exit Sequencing Request Process* (**End Sequencing Session**: *True*)	The sequencing session has ended, return control to the LTS
	End If	
4.	**Exit** *Exit Sequencing Request Process* (**End Sequencing Session**: *False*)	

SB.2.12 Sequencing Request Process

The *Sequencing Request Process* is the overall control process that handles all sequencing requests. It simply dispatches a request to an appropriate processor. The individual request processors will do one of the following:

- Indicate that the sequencing process does not identify a candidate activity to deliver:
  - These is no candidate – the overall sequencing system will wait for the next navigation request.
  - There is a candidate, but the limit conditions are exceeded – The overall sequencing system will wait for the next navigation request.
- Indicate that the entire sequencing process should terminate.
- Return a candidate activity to deliver.
- Return an error (generally a malformed request).

The *Sequencing Request Process* for a sequencing request is specified by the following pseudo code. The pseudo code describes only the request and tree traversal logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Sequencing Request Process* only describes the expected behavior that an implementation will exhibit.

Sequencing Request Process (for a sequencing request; validates the sequencing request; may return a delivery request; may indicate control be returned to the LTS):		
	Reference: Choice Continue Exit Previous Resume All Retry Start Sequencing Request Process	**Section:** SB.2.9 SB.2.7 SB.2.11 SB.2.8 SB.2.6 SB.2.10 SB.2.5
1.	**Case:** sequencing request is *Start*	
1.1.	Apply the *Start Sequencing Request Process*	
1.2.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: the result of *Start Sequencing Request Process*; **End Sequencing Session**: *n/a*)	
	End Case	
2.	**Case:** sequencing request is *Resume All*	
2.1.	Apply the *Resume All Sequencing Request Process*	
2.2.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: *Resume All Sequencing Request Process*; **End Sequencing Session**: *n/a*)	
	End Case	
3.	**Case:** sequencing request is *Exit*	
3.1.	Apply the *Exit Sequencing Request Process*	
3.2.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: *n/a* **End Sequencing Session**: the result of *Exit Sequencing Request Process*)	
	End Case	
4.	**Case:** sequencing request is *Retry*	
4.1.	Apply the *Retry Sequencing Request Process*	
4.2.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: the result of *Retry Sequencing Request Process*; **End Sequencing Session**: *n/a*)	
	End Case	
5.	**Case:** sequencing request is *Continue*	
5.1.	Apply the *Continue Sequencing Request Process*	
5.2.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: the result of *Continue Sequencing Request Process*; **End Sequencing Session**: *n/a*)	
	End Case	
6.	**Case:** sequencing request is *Previous*	
6.1.	Apply the *Previous Sequencing Request Process*	
6.2.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: the result of *Previous Sequencing Request Process*; **End Sequencing Session**: *n/a*)	
	End Case	
7.	**Case:** sequencing request is *Choice*	
7.1.	Apply the *Choice Sequencing Request Process*	
7.2	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Valid*; **Delivery Request**: the result of *Choice Sequencing Request Process*; **End Sequencing Session**: *n/a*)	
	End Case	
8.	**Exit** *Sequencing Request Process* (**Sequencing Request**: *Not Valid*; **Delivery Request**: *n/a*; **End Sequencing Session**: *n/a*)	Invalid sequencing request.

DB. Delivery Behavior Model

The Simple Sequencing process identifies an activity whose content resources are to be delivered to the learner (a delivery request). The identified content resources should then be delivered to the learner. Before the content is actually delivered, the Simple Sequencing process must validate that the content resources from the identified activity may be delivered, i.e., all of the conditions that apply to the delivery of the content for the activity and attempt still hold. The process of validating and then initiating the content delivery is called the “delivery process”.

The delivery process makes no assumptions as to how or when an activity for delivery is identified, including if or when the delivery request resulted from a sequencing request. The delivery process must insure that delivery requests are valid. Once the delivery request is validated, the content resources for the activity are delivered. The delivery process must ensure that certain elements of the tracking model and the activity state model are updated as part of content delivery.

The mechanisms used to actually deliver the content resources for the activity to the learner are not specified. The mechanisms used to update the tracking data for objectives and activities for the learner as the result of delivering the content resources for the activity are not specified.

The overall sequencing process (see Overall Sequencing Process **OP**) relates the delivery process to the navigation, sequencing, exit, selection and randomization, and rollup processes.

The delivery process is controlled by parts of the sequencing definition model:

- *Delivery Controls* – actions applied when or while the content resources for the activity are delivered.

The delivery process uses all parts of the tracking model:

- *Objective Information* – information about the results of the learner’s interactions related to an objective.
- *Progress Information* – information about a learner’s *attempt* at an activity, updated by the delivery process.

The delivery process uses data from the activity state model.

The behavior of the delivery process is defined in terms of two different processes:

- *Delivery Request Process* – validates a pending delivery request to ensure that the identified activity is allowed for delivery.
- *Content Delivery Environment Process* – prepares the activity tree and tracking models for the delivery of an activity and informs the LTS of the activity to be launched.

These individual processes are part of the overall process that controls all delivery behavior. The actual delivery is governed by a set of delivery controls and tracking requirements. Subsequent to delivery, the content operates in a managed delivery environment.

DB.1 Delivery Behavior

The delivery behavior describes how a sequencing system interprets a delivery request in combination with the elements of the sequencing definition model and with instance data from the tracking model to validate the delivery request and initiate the delivery of the content resources for the activity and tracking of the activity.

Only leaf activities are valid targets for delivery requests. Delivery requests are validated using only the disabled sequencing rules and limit conditions. No other limits, rules, conditions or control modes are applied when validating a delivery request.

An implementation must be capable of representing the processes described and have the implemented process exhibit the behavior described. There are no additional requirements on implementing the delivery behavior model.

The delivery behavior relies on the data descriptions from the sequencing definition model (see Sequencing Definition Model **SM**) and the tracking model (see Tracking Model **TM**). These information models also specify default data values that govern the access to activity or tracking data.

DB.1.1 Delivery Request Process

The *Delivery Request Process* is used to validate the identified activity before delivery. The process is applied to:

- An activity tree, designated by the root activity of the tree.
- The leaf activity within the tree that is identified for delivery.

The root of the tree and the identified activity within the tree specify a unique path through the tree, designated the delivery path. The delivery validation checks are applied to all activities along that path, starting from the root activity to the activity identified for delivery.

The *Delivery Request Process* is specified by the following pseudo code. The pseudo code describes only the delivery validation logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Delivery Request Process* only describes the expected behavior that an implementation will exhibit.

Delivery Request Process (for a delivery request; returns the validity of the delivery request):		
	Reference: Check Activity Process	**Section:** UP.5
1.	**If** the activity specified by the delivery request is not a leaf **Then**	Can only deliver leaf activities.
1.1.	**Exit** *Delivery Request Process* (**Delivery Request**: *Not Valid*)	
	End If	
2.	Form the activity path as the ordered series of activities from the root of the activity tree to the activity specified in the delivery request, inclusive	
3.	**If** the activity path is *Empty* **Then**	Nothing to deliver.
	Exit *Delivery Request Process* (**Delivery Request**: *Not Valid*)	
	End If	
4.	**For** each activity in the activity path	Make sure each activity along the path is allowed.
4.1.	Apply the *Check Activity Process* to the activity	
4.2.	**If** the *Check Activity Process* return *True* **Then**	
4.2.1.	**Exit** *Delivery Request Process* (**Delivery Request**: *Not Valid*)	
	End If	
	End For	
5.	**Exit** *Delivery Request Process* (**Delivery Request**: *Valid*)	

DB.2 Content Delivery Environment Process

Once the delivery request has been validated, the activity is delivered. The delivery environment must deliver the content resources for the activity to the learner. The process of delivering the content, e.g., retrieving it from a content repository, delivering it via an HTTP response, is not specified.

The *Content Delivery Environment Process* controls how the activity is started or attempted and how data for the activity is recorded. The *Content Delivery Environment Process* assumes the request being processed has been validated for delivery.

If the activity is *Tracked*, then tracking information should be recorded. Tracking information should not change if the activity is not tracked.

If the activity was suspended, the previous attempt is resumed when the activity is delivered; otherwise, the delivery is a new attempt.

When an activity is delivered, any previous suspended sequencing session state will be cleared; the learner can no longer resume a previously suspended session.

If auxiliary resources are associated with the activity to be delivered, a set of auxiliary resource descriptions are provided to the delivery environment. How the delivery environment processes those resource descriptions is not defined in this model.

How the delivery environment tracks data, including activity and attempt durations, is not specified. The delivery environment may be able to determine if the content delivery was attempted, but may not be able to determine if the content was actually delivered to the learner (e.g., an HTTP server responds to the request for content but there may be no mechanism to determine if the client browser received and displayed the content).

The delivery environment is also responsible for recording any necessary data to support tracking and navigation requests.

The *Content Delivery Environment Process* for a delivery request is specified by the following pseudo code. The pseudo code describes only how the activity state is maintained and what should happen once the content has been delivered to the learner. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Content Delivery Environment Process* only describes the expected behavior that an implementation will exhibit.

Content Delivery Environment Process (for a delivery request):

Reference:	**Section:**
Activity Attempt Count	TM.1.2.1
Activity is Active	AM.1.1
Activity is Suspended	AM.1.1
Attempt Absolute Duration	TM.1.2.2
Attempt Experienced Duration	TM.1.2.2
Attempt Progress Information	TM.1.2.2
Clear Suspended Activity Subprocess	DB.2.1
Current Activity	AM.1.2
Objective Progress Information	TM.1.1
Suspended Activity	AM.1.2
Terminate Descendent Attempts Process	UP.4
Tracked	SM.11

1.	**If** the *Activity is Active* for the *Current Activity* is *True* **Then**	If the attempt on the current activity has not been terminated, we cannot deliver new content.
1.1.	**Exit** *Content Delivery Environment Process*	Delivery request is invalid – The *Current Activity* has not been terminated.
	End If	
2.	If the activity identified for delivery is not equal to the *Suspended Activity* Then	Content is about to be delivered, clear any existing suspend all state.
2.1	Apply the *Clear Suspended Activity Subprocess* to the activity identified for delivery	
	End If	
3.	Apply the *Terminate Descendent Attempts Process* to the activity identified for delivery	Make sure that all attempts that should end are terminated.

4.	Form the activity path as the ordered series of activities from the root of the activity tree to the activity identified for delivery, inclusive	Begin all attempts required to deliver the identified activity.
5.	**For** each activity in the activity path	
5.1.	**If** *Activity is Active* for the activity is *False* **Then**	
5.1.1.	**If** *Tracked* for the activity is *True* **Then**	
5.1.1.1.	**If** *Activity is Suspended* for the activity is *True* **Then**	If the previous attempt on the activity ended due to a suspension, clear the suspended state; do not start a new attempt
5.1.1.1.1.	Set *Activity is Suspended* for the activity to *False*	
5.1.1.2.	**Else**	
5.1.1.2.1.	Increment the *Activity Attempt Count* for the activity	Begin a new attempt on the activity.
5.1.1.2.2.	Initialize *Objective Progress Information* and *Attempt Progress Information* required for the new attempt	Initialize tracking information for the new attempt.
	End If	
5.1.1.3.	Set *Activity is Active* for the activity to *True*	
	End If	
	End If	
	End For	
6.	Set *Current Activity* to the activity identified for delivery	The activity identified for delivery becomes the current activity.
7.	Once the delivery of the activity's content resources and auxiliary resources begins	The delivery environment is assumed to deliver the content resources associated with the identified activity. While the activity is assumed to be active, the sequencer may track learner status.
7.1.	**If** *Tracked* for the activity identified for delivery is *False* **Then**	
7.1.1.	The Objective and Attempt Progress information for the activity should not be recorded during delivery	
7.1.2.	The delivery environment begins tracking the *Attempt Absolute Duration* and the *Attempt Experienced Duration*	
	End If	
8.	**Exit** *Content Delivery Environment Process*	

DB.2.1 Clear Suspended Activity Subprocess

The *Clear Suspended Activity Subprocess* clears the suspended state of the *Suspended Activity* and all of its ancestors, ensuring that the state of the activity tree is current.

The *Clear Suspended Activity Subprocess* does not use data from the sequencing definition model.

The *Clear Suspended Activity Subprocess* does not use data from the tracking model.

The *Clear Suspended Activity Subprocess* uses parts of the activity state model:

Suspended Activity – describes the current activity of the previous sequencing session, which ended with a suspend all navigation request.

The *Clear Suspended Activity Subprocess* for an activity is specified by the following pseudo code. The pseudo code describes only the state-change logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Clear Suspended Activity Subprocess* only describes the expected behavior that an implementation will exhibit.

Clear Suspended Activity Subprocess (for an activity; may change the *Suspended Activity*):		
	Reference: Activity is Suspended Suspended Activity	**Section:** AM.1.1 AM.1.2
1.	**If** the *Suspended Activity* is *Defined* **Then**	Make sure there is something to clear.
1.1.	Find the common ancestor of the identified activity and the *Suspended Activity*	
1.2.	Form an activity path as the ordered series of activities from the parent of the *Suspended Activity* to the common ancestor, inclusive	
1.3.	**If** the activity path is **Not** *Empty* **Then**	
1.3.1.	**For** each activity in the activity path	Walk down the tree setting each of the identified activities to ‘not suspended’.
1.3.1.1.	**If** the activity is a leaf **Then**	
1.3.1.1.1.	Set *Activity is Suspended* for the activity to *False*	
1.3.1.2.	**Else**	
1.3.1.2.1.	**If** the activity does not include any child activity whose *Activity is Suspended* attribute is *True* **Then**	
1.3.1.2.1.1.	Set *Activity is Suspended* for the activity to *False*	
	End If	
	End If	
	End For	
	End If	
1.4.	Set *Suspended Activity* to not *Defined*	Clear the Suspended Activity attribute.
	End If	
2.	**Exit** *Clear Suspended Activity Subprocess*	

UP. Utility Processes

UP.1 Limit Conditions Check Process

The *Limit Conditions Check Process* applies the limit conditions from the sequencing definition model to determine if any of the activity's limit conditions have been violated. The *Limit Conditions Check Process* is controlled by parts of the sequencing definition model:

- *Limit Conditions* – limits on how many times, how long, and when an activity is allowed.
- *Delivery Controls* – describe if the activity is tracked.

The *Limit Conditions Check Process* uses parts of the tracking model:

Activity / Attempt Progress Information – information about a learner's *attempt(s)* at an activity.

The *Limit Conditions Check Process* does not use the activity state model.

Limit conditions are specified by all of the Limit Condition attributes in the sequencing definition model. These attributes constrain the number of access attempts, the duration of access or time limit on access and the time frame of access. Limit conditions are evaluated using data values for Activity and Attempt Progress attributes of the tracking model.

- If the activity is tracked, the *Limit Conditions Check Process* determines if any of the activity's limit conditions have been violated based on the current tracking information available. The process need only identify the first limit condition that is violated. The evaluation order of checking limit conditions is arbitrary.

The *Limit Conditions Check Process* for an activity is specified by the following pseudo code. The pseudo code describes only the limit condition logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Limit Conditions Check Process* only describes the expected behavior that an implementation will exhibit.

Limit Conditions Check Process (for an activity; returns *True* if any of the activity's limit conditions have been violated):

Reference:	**Section:**
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 *Limit Condition Attempt Control* for the activity is *True* **Then**	
2.1.	**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**	
2.1.1.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *True*)	Limit conditions have been violated.
	End If	
	End If	
3.	**If** the *Limit Condition Activity Absolute Duration Control* for the activity is *True* **Then**	
3.1.	**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**	
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 Experienced Duration Control* for the activity is *True* **Then**	
4.1.	**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**	
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 Attempt Absolute Duration Control* for the activity is *True* **Then**	
5.1.	**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**	
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 Experienced 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 Experienced Duration* for the activity is greater than or equal to the *Limit Condition Attempt Experienced 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 Begin Time Limit Control* for the activity is *True* **Then**	
7.1.	**If** the current time point is before the *Limit Condition Begin Time Limit* for the activity **Then**	
7.1.1.	**Exit** *Limit Conditions Check Process – True*	Limit conditions have been violated.
	End If	
	End If	
8.	**If** the *Limit Condition End Time Limit Control* for the activity is *True* **Then**	
8.1.	**If** the current time point is after the *Limit Condition End 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.	**Exit** *Limit Conditions Check Process* (**Limit Condition Violated**: *False*)	No limit conditions have been violated.

UP.2 Sequencing Rules Check Process

The *Sequencing Rules Check Process* evaluates a set of sequencing rules for an activity, attempting to identify an action that must be applied.

The *Sequencing Rules Check Process* is controlled by parts of the sequencing definition model:

- *Sequencing Rules* – Describes the sequencing rules for the activity.
- The *Sequencing Rule Check Subprocess* does not use the tracking model.
- The *Sequencing Rule Check Subprocess* does not use the activity state model.

Sequencing Rules Check Process utilizes the *Sequencing Rule Check Subprocess* to evaluate each rule in the set. The process ends at the first rule that evaluates to true.

- The designated set sequencing rules are evaluated in the order defined by the Sequencing Rule Description (SM.3) for the activity. The process stops at the first rule that evaluates to True. The process returns the Rule Action of the first sequencing rule that evaluated to True, or Nil if none of the rules apply.

The *Sequencing Rules Check Process* for an activity is specified by the following pseudo code. The pseudo code describes only the rule-checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Sequencing Rules Check Process* only describes the expected behavior that an implementation will exhibit.

Sequencing Rules Check Process (for an activity and a set of *Rule Actions*; returns the action to apply or *Nil*):		
	Reference: Rule Action / Sequencing Rule Check Subprocess / Sequencing Rule Description	**Section:** SM.2 / UP.2.1 / SM.2
1.	**If** the activity includes *Sequencing Rules* with any of the specified *Rule Actions* **Then**	Make sure the activity has rules to evaluate.
1.1.	Initialize rules list by selecting the set of *Sequencing Rules* for the activity that have any of the specified *Rule Actions*, maintaining original rule ordering	
1.2.	**For** each rule in the rules list	
1.2.1.	Apply the *Sequencing Rule Check Subprocess* to the activity and the rule	Evaluate each rule, one at at a time.
1.2.2.	**If** the *Sequencing Rule Check Subprocess* returns *True* **Then**	
1.2.2.1.	**Exit** *Sequencing Rules Check Process* (**Action**: *Rule Action* for the rule)	Stop at the first rule that evaluates to true – perform the associated action.
	End If	
	End For	
	End If	
2.	**Exit** *Sequencing Rules Check Process* (**Action**: *Nil*)	No rules evaluated to true – do not perform any action.

UP.2.1 Sequencing Rule Check Subprocess

The *Sequencing Rule Check Subprocess* evaluates a set of conditions against an activity's current tracking information.

The *Sequencing Rule Check Subprocess* is controlled by parts of the sequencing definition model:

- *Sequencing Rules* – Describes the sequencing rules for the activity.

The *Sequencing Rule Check Subprocess* uses all parts of the tracking model:

- *Objective Information* – information about the results of the learner's interactions related to an objective.

Activity / Attempt Progress Information – information about a learner's attempt(s) at an activity.

The *Sequencing Rule Check Subprocess* does not use the activity state model.

Evaluating the conditions uses data values for Objective and Attempt Progress attributes of the tracking model.

- The activity's tracking information is applied against a set of rule conditions. The *Sequencing Rule Check Subprocess* returns True if the set of rule conditions are met, or False if they are not met.

The *Sequencing Rule Check Subprocess* for an activity is specified by the following pseudo code. The pseudo code describes only the rule-checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Sequencing Rule Check Subprocess* only describes the expected behavior that an implementation will exhibit.

Sequencing Rule Check Subprocess (for an activity and a *Sequencing Rule*; returns *True* if the rule applies):		
	Reference: Rule Combination; Rule Condition; Rule Condition Operator; Sequencing Rule Description; Tracking Model	**Section:** SM.2; SM.2; SM.2; SM.2; TM
1.	Initialize rule condition bag as an *Empty* collection	This is used to keep track of the evaluation of the rule's conditions.
2.	**For** each *Rule Condition* for the *Sequencing Rule* for the activity	
2.1.	Evaluate the rule condition by applying the appropriate tracking information for the activity to the *Rule Condition*	Evaluate each condition against the activity's tracking information.
2.2.	**If** the *Rule Condition Operator* for the *Rule Condition* is *Not* **Then**	
2.2.1.	**Negate** the rule condition	
	End If	
2.3.	Add the value of rule condition to the rule condition bag	Add the evaluation of this condition to the set of evaluated conditions.
	End For	
3.	**If** the rule condition bag is *Empty* **Then**	If there are no defined conditions for the rule, the rule does not apply.
3.1.	**Exit** *Sequencing Rule Check Subprocess* (**Result**: *False*)	No rule conditions
	End If	
4.	Apply the *Rule Combination* for the *Sequencing Rule* to the rule condition bag to produce a single combined rule evaluation	'And' or 'Or' the set of evaluated conditions, based on the sequencing rule definition.
5.	**Exit** *Sequencing Rule Check Subprocess* (**Result**: the value of rule evaluation)	

UP.3 Terminate Descendent Attempts Process

The *Terminate Descendent Attempts Process* ends the current attempt on a set of activities to ensure the state of the activity tree is current.

The *Terminate Descendent Attempts Process* does not use data from the sequencing definition model.

The *Terminate Descendent Attempts Process* does not use data from the tracking model.

The *Terminate Descendent Attempts Process* uses parts of the activity state model:

Current Activity – describes the most recently delivered or terminated activity in the activity tree.

The *Terminate Descendent Attempts Process* assumes the current attempt on the current activity has already ended, so it does not end the attempt on the current activity.

- The *Terminate Descendent Attempts Process* does not return any value. It determines the common ancestor between an identified activity and the current activity. It applies the *End Attempt Process* (UP.4) to each active descendent of the common ancestor.

The *Terminate Descendent Attempts Process* for an activity is specified by the following pseudo code. The pseudo code describes only the termination logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Terminate Descendent Attempts Process* only describes the expected behavior that an implementation will exhibit.

Terminate Descendent Attempts Process (for an activity):		
	Reference:	**Section:**
	Current Activity	AM.1.2
	End Attempt Process	UP.4
1.	Find the activity that is the common ancestor of the *Current Activity* and the identified activity	
2.	Form the activity path as the ordered series of activities from the *Current Activity* to the common ancestor, exclusive of the *Current Activity* and the common ancestor	
3.	**If** the activity path is **Not** *Empty* **Then**	There are some activities that need to be terminated.
3.1.	**For** each activity in the activity path	
3.1.1.	Apply the *End Attempt Process* to the activity	End the current attempt on each activity.
	End For	
	End If	
4.	**Exit** *Terminate Descendent Attempts Process*	

UP.4 End Attempt Process

The result of a learner's interactions with an activity is reflected in a set of objective and attempt progress data. Simple Sequencing makes no assumptions as to how or when these results are set. A content object may directly set the appropriate objective and activity progress data. However, not all content objects may have the ability to set the data. The delivery controls *Completion Set by Content* and *Objective Set by Content* are used to indicate whether the content object will or will not provide data used to set the corresponding activity's objective and progress status information.

Simple Sequencing makes no provisions for the exact mechanics of how content can set the data values in the tracking model. Simple Sequencing requires that if content sets values in the tracking model, the elements in the track model must be set prior to invoking the *End Attempt Process*.

The *End Attempt Process* uses data values from the Delivery Controls. It may set data values for Objective and Attempt Progress information attributes of the tracking model.

- The *End Attempt Process* specifies values to use when content objects do not set values. Values are only set when the activity is being *Tracked*.
  - An activity that terminates is considered complete.
  - The objective associated with the activity is considered satisfied (but without a satisfaction measure).

Setting of values in the tracking model is controlled only by the delivery controls. When specified by the Delivery Controls, values set by the *End Attempt Process* set the corresponding tracking values. However, the *End Attempt Process* will not overwrite any values set by the content object. If the delivery controls indicate that the attempt termination process should not set values, and if the content object also does not set values, the default values from the tracking model are used in subsequent rollup and sequencing.

When an activity terminates or is suspended, actions are required to properly accumulate the time spent on the activity and the time spent on the attempt at the activity. Determining and accumulating such times could be done as part of the described completion subprocess (time is accumulated independently only if the activity is *Tracked*). This behavior is not described here, but an implementation must record and update the absolute and experienced durations of both the attempt and the activity. The meaning of these data attributes (and the implicit behavior required to accumulate the times) are described in the tracking model (see Tracking Model **TM**).

If a 'write' objective map is defined for the activity being processed that transfers objective information to a shared global objective, the shared global objective's information must be set prior to the completion of the *End Attempt Process*.

The *End Attempt Process* does not explicitly return a result.

The *End Attempt Process* is specified by the following pseudo code. The pseudo code describes only the *End Attempt Process* termination logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *End Attempt Process* only describes the expected behavior that an implementation will exhibit.

End Attempt Process (for an activity):		
	Reference: Activity is Active Activity is Suspended Attempt Completion Status Attempt Progress Status Completion Set by Content Objective Contributes to Rollup Objective Progress Status Objective Satisfied Status Objective Set by Content Tracked	**Section:** AM.1.1 AM.1.1 TM.1.2.2 TM.1.2.2 SM.11 SM.6 TM.1.1 TM.1.1 SM.11 SM.11
1.	**If** the activity is a leaf **Then**	
1.1.	**If** *Tracked* for the activity is *True* **Then**	
1.1.1.	**If** the *Activity is Suspended* for the activity is *False* **Then**	The sequencer will not affect the state of suspended activities.
1.1.1.1.	**If** the *Completion Set by Content* for the activity is *False* **Then**	Should the sequencer set the completion status of the activity?
1.1.1.1.1.	**If** the *Attempt Progress Status* for the activity is *False* **Then**	Did the content inform the sequencer of the activity's completion status?
1.1.1.1.1.1.	Set the *Attempt Progress Status* for the activity to *True*	
1.1.1.1.1.2.	Set the *Attempt Completion Status* for the activity to *True*	
	End If	
	End If	
1.1.1.2.	**If** the *Objective Set by Content* for the activity is *False* **Then**	Should the sequencer set the objective status of the activity?
1.1.1.2.1.	**For** all objectives associated with the activity	
1.1.1.2.1.1	**If** the *Objective Contributes to Rollup* for the objective is *True* **Then**	
1.1.1.2.1.1.1.	**If** the *Objective Progress Status* for the objective is *False* **Then**	Did the content inform the sequencer of the activity's rolled-up objective status?
1.1.1.2.1.1.1.1.	Set the *Objective Progress Status* for the objective to *True*	

1.1.1.2.1.1.1.2.	Set the *Objective Satisfied Status* for the objective to *True*	
	End If	
	End If	
	End For	
	End If	
	End If	
	End If	
2.	**Else**	The activity has children.
2.1.	**If** the activity includes any child activity whose *Activity is Suspended* attribute is *True* **Then**	The suspended status of the parent is dependent on the suspended status of its children.
2.1.1.	Set the *Activity is Suspended* for the activity to *True*	
2.2.	**Else**	
2.2.1.	Set the *Activity is Suspended* for the activity to *False*	
	End If	
	End If	
3.	Set the *Activity is Active* for the activity to *False*	The current attempt on the activity has ended.
4.	**Exit** *End Attempt Subprocess*	

UP.5 Check Activity Process

The *Check Activity Process* is used to determine if an activity is allowed. The process is applied to a single activity, checking both limit conditions and *Disabled* sequencing rules. The *Check Activity Process* applies both the *Limit Conditions Check Process* and the *Sequencing Rules Check Process* to the activity.

- The *Check Activity Process* is applied to the single identified activity. It returns a Boolean: True if any limit condition or *Disabled* rule is violated; or False if no limit condition or *Disabled* rule is violated.

The *Check Activity Process* is specified by the following pseudo code. The pseudo code describes only the checking logic. How this process is implemented or how information is encoded, stored, represented, or bound is outside the scope of this specification. The *Check Activity Process* only describes the expected behavior that an implementation will exhibit.

Check Activity Process (for an activity; returns *True* if the activity is disabled or violates any of its limit conditions):		
	Reference: Disabled Rules; Limit Conditions Check Process; Sequencing Rules Check Process	**Section:** SM.2; UP.1; UP.2
1.	Apply the *Sequencing Rules Check Process* to the activity and the *Disabled sequencing rules*	Make sure the activity is not disabled.
2.	**If** the *Sequencing Rules Check Process* does not return *Nil* **Then**	
2.1.	**Exit** *Check Activity Process* (**Result**: *True*)	
	End If	
3.	Apply the *Limit Conditions Check Process* to the activity	Make the activity does not violate any limit condition.
4.	**If** the *Limit Conditions Check Process* returns *True* **Then**	
4.1.	**Exit** *Check Activity Process* (**Result**: *True*)	
	End If	
5.	**Exit** *Check Activity Process* (**Result**: *False*)	Activity is allowed.

E. Extensibility

E.1 Extensibility of Information Model and Behaviors

The Simple Sequencing Specification does not explicitly define formal extensibility mechanisms to represent additional sequencing descriptions or additions to the Simple Sequencing Information Model[2] components and their associated sequencing behaviors. This specification does not explicitly exclude extensions to the Information Model or the specified behaviors.

Simple Sequencing does not represent all possible sequencing behaviors. Implementers may explore different or additional sequencing behaviors by defining additional capabilities and by adding elements to the Sequencing Definition Model and describing their associated behaviors. To maximize interoperability of systems that implement this specification, extended data elements should not replace data elements in the Information Model, i.e., new data elements should not provide the same functionality as existing elements or override or disable existing sequencing behaviors.

Specifications for extensions maybe developed by communities of practice and specified in application profiles. Such extensions may define additional requirements for interoperability. An implementation of Simple Sequencing should not depend on the presence or implementation of any particular extension and associated behavior when processing an instance of a Sequencing Definition Model that does not rely on or include the extensions.

Extensions to the Information Model should be accompanied by corresponding extensions or modifications to the behavior model. Implementers should attempt to keep all extensions consistent with all of the assumptions that underlie this specification and the functionality within the existing behavior model.

Extensions and behaviors that are outside the scope of describing sequencing, e.g., rendering behaviors, navigation controls, or runtime control behaviors, should be kept separate from sequencing behaviors. Future specifications may address such out-of-scope issues.

E.2 Extensibility of Bindings

The *XML Binding for Sequencing Definition Model* document [5] describes normative criteria for extending the XML binding of the Simple Sequencing Specification.

- The IMS XML Schema Definition File (xsd) for Simple Sequencing may not be modified.
- Elements defined within the IMS Simple Sequencing namespace may not be modified.
- Extensions shall be placed in a different XML namespace.

2. The Information Model includes the Sequencing Definition Model, Tracking Model and Activity State Model.

C. Conformance

A system that implements this specification may conform to the behavior described in this specification or it may conform to the XML Binding of the Sequencing Definition Model. An implementation may conform to either part independently, i.e., a system may process sequencing definitions and sequence learning activities without being able to exchange sequencing definitions via the specified XML binding.

C.1 Behavior Conformance

A *strictly conforming* implementation instance shall implement only the specified Information Model elements and behaviors. A *conforming* implementation may extend the specified Information Model elements and behaviors.

The process or processes that implement the behavior model of this specification operate on an unspecified internal representation of the Information Model and activity tree. For any external[3] *navigation request*, a strictly conforming implementation shall identify the activity and the content resources to be delivered to the learner in accordance to the behavior specified.

Given a specific navigation request, a particular description of activities and sequencing behaviors as defined by the Sequencing Definition Model for the particular activity tree, and a defined set of values for all elements in the Tracking Model and Activity State Model for the corresponding activity tree, any two strictly conforming implementations of this specification shall identify the same activity and content resources to be delivered to the learner.

For a given sequence of timing-independent navigation requests[4], a particular description of activities and sequencing behaviors as defined by the Sequencing Definition Model for the particular activity tree, and a defined set of *initial* values for all elements in the Tracking Model and Activity State Model for the corresponding activity tree, any two strictly conforming implementations of this specification shall identify the same sequence of activities and content resources to be delivered to the learner. After processing any navigation request in the sequence of requests, any two strictly conforming implementations of this specification shall have equivalent values for all elements in the Tracking Model and Activity State Model for the corresponding activity tree.

A strictly conforming implementation that provides the capability to processes external navigation requests, exits requests, sequencing requests, rollup requests and delivery requests shall process such requests and produce the results in accordance with the behavior specified. A strictly conforming implementation must process external navigation requests. There is no requirement that a strictly conforming implementation process any other external requests.

C.2 Binding and Interchange Conformance

The *XML Binding for Sequencing Definition Model* document [5] describes normative criteria for conforming to the XML binding of the Simple Sequencing Specification.

To conform to the XML binding of this specification, an implementation that exports a Sequencing Definition Model maps its unspecified internal representation of the Sequencing Definition Model and activity tree to the corresponding elements of the XML binding embedded within the organization structure of an IMS Content Package manifest. An implementation that imports an XML file that includes an instance of an IMS Content Package manifest that includes Simple Sequencing elements shall map these elements to its unspecified internal representation of the Sequencing Definition Model and activity tree.

A strictly conforming implementation shall map all elements of its internal representation to the elements of the XML representation for export and for import, shall map all elements of an XML definition into its internal representation.

An IMS Content Package manifest that does not include any sequencing elements is strictly conforming and shall be processed using the defaults for Sequencing Definition Model.

3. A request that comes from outside of the sequencing processor.

4. One navigation request must be fully processed before the next is processed and there must be no timing dependencies in processing the requests.

G. Glossary

The following terms are used through out the Simple Sequencing document set to describe parts of the information model or sequencing behavior.

Absolute Duration	The cumulative amount of time that a learner spends on an activity, from when the activity starts until it ends (see also *Experienced Duration*).
Activity	A discrete unit of learning that is sequenced. Activities may have associated *resources* that represent content that can be delivered to the learner. Activities can be organized in aggregations to form higher-level activities and may be composed of multiple levels of sub-activities. *Alternative*: An instantiation of a node in the activity tree for a learner.
Activity List	An ordered subset of activities from an activity's set of Available Children.
Activity Path	See *Path*.
Activity State Model	An information model containing state data for learner-related interactions with activities.
Activity Tree	The hierarchical collection of content objects (activities) with associated rules and specifications of learning behaviors, conditions and limits. The activity tree describes a complex learning experience. *Alternative:* The representation of the parent-child relationships between activities.
Aggregation	A subtree of the activity tree with a root at any activity.
Attempt	A tracked interaction of a learner with an activity. The attempt begins when the activity is delivered and continues until the activity terminates without being suspended (see also *resume*, *suspend*). An attempt may span multiple sessions (see also *session*).
Content Aggregation	See *Aggregation*.
Content Resource	See *Resource*.
Content Package	A collection of learning content assets and resources, and supporting data, including a manifest, as defined by the IMS Content Packaging Specification [3].
Cluster	A node in the activity tree and all direct (first level) descendents of the node.
Delivery Behavior Model	The process that validates that the content resources for the identified activity may be delivered, i.e., none of the conditions that apply to the delivery of the content for the activity and attempt have been or are being violated.
Delivery Request	The identification of an activity whose content resources are to be delivered to the learner.
Exit Request	The identification of the action to take to exit an activity or set of activities.
Experienced Duration	The cumulative amount of time that a learner spends on an activity. The time is measured from the start of the activity to the end of the activity and excludes any time when the activity is suspended (see also *Absolute Duration*).
External Event	An event that comes from outside a Simple Sequencing System.
External Request	A request (navigation, sequencing, delivery, exit, etc.) that comes from outside a Simple Sequencing System.
Learner	The agent for whom interaction data is maintained.
Most Recent	Current or last fully completed. For example, the most recent attempt at an activity is the current interaction or current attempt if the activity is being experienced, it is the last fully completed interaction with the activity (if there was one).

Navigation Behavior Model	The process that evaluates a navigation request and determines the sequencing and exit requests that should be processed to identify and deliver content to the learner.
Navigation Event	An event resulting from an external user interaction that triggers a navigation request.
Navigation Request	The identification of a navigation action.
Objective	A (global) data item that may be associated with one or more elements of the activity tree. The Tracking Model maintains certain data items for an objective (for each learner) including an indication of the learner having satisfied the objective and a measure of satisfaction. While the objective may be associated with a learning objective and the tracking data may represent a pass/fail status and a score, the data item can be used for any Boolean state value with an associated optional measure.
Objective Information	Information about the results of the learner's interactions related to an objective.
Overall Sequencing Process	The overall sequencing process that relates the navigation, exit, sequencing, selection and randomization, delivery, and rollup processes.
Path	The set of activities encountered walking parent-child associations in the activity tree, from some starting activity to some target activity.
Progress Information	Information about a learner's *attempt* at an activity.
Resource	Content to be delivered to the learner (e.g., web page, media file, text file, assessment object) as defined by the IMS Content Packaging Specification [3].
Resume	Restarting an activity that had been previously suspended (see also *attempt*, *suspend*).
Rollup Behavior Model	The process that computes the results data for an activity from the results data from the children of the activity.
Selection and Randomization Behavior Model	The process that selects, randomizes and orders a set of learning activities.
Sequencing	What a Learning Technology System does when it sequences content.
Sequencing Behavior Model	The process that evaluates a sequencing request in terms of the content model described by the activity tree and determines what actual content object should be delivered to the learner.
Sequencing Definition Model	An information model describing intended sequencing behaviors. Rules, limit conditions, and sequencing behaviors defined for activities.
Sequencing Request	The identification of a sequencing action.
Session	A learner's interaction with an activity not interrupted by suspending the activity (see also *suspend*).
Suspend	Exiting an activity with the intent of returning to it. The activity is not marked as complete when exited (see also *attempt*, *resume*).
Termination Behavior Model	The process that evaluates an termination request to end the current attempt on an activity.
Tracking Model	An information model containing results data for a learner related to objectives and progress on learning activities.

UC. Core Use Case Descriptions

The Simple Sequencing Specification development was based on several use cases that served to defined certain basic requirements. Four of these cases are considered to be core cases and are presented here.

UC.1 Boeing Fuel Valve Removal Use Case

The purpose of this discussion is to describe a use case presented by Boeing to be used as a test for the IMS Simple Sequencing Working Group.

The Fuel Valve Removal course is a fictitious example representing part of a maintenance technician's curriculum. It is assumed that the student will have completed courses that familiarize the student with the vehicle and its systems prior to taking this course. The fuel valve removal course teaches how to remove a fuel valve from a fictitious aircraft's wing in order to service the valve. To enable access to valve, the proper door and the fuel quantity transmitter must be removed.

UC.1.1 Terminology

Although this discussion uses the term "course" the term is meant only as a convenient way to describe the group of instruction in the example. The example may also be considered in terms of an AICC's conceptual notion of a "Learning Object." This discussion refers to the first block and its instructional modules as an introduction. The second block is referred to as the lesson block and the modules are called lessons. The lesson and exam blocks also allow the student to view the Interactive Electronic Technical Manual (IETM). The final block is the test block with the first two tests grouped so that the student only sees them as a unit called the knowledge test. The last test in the test block is a simulation-based test called a performance test.

UC.1.2 Content Structure

The course is comprised of three blocks. The first block serves as an introduction, the second block contains the actual lessons, and the third block consists of tests.

```
Fuel Valve Introduction
   Fuel Valve Lesson Intro
   Fuel Valve Theory of Operation
Fuel Valve Lessons
   Fuel Valve and Quantity Transmitter Components
   Fuel System Hazards
   Fuel Valve Removal Procedure
      Preparation
      Remove Door
      Remove Transmitter
      Remove Valve
Tests
   Knowledge Test
      Fuel System Components
      Fuel System Hazards
   Performance Test (simulation)
      Removal Simulation (Prep, Remove Door, Remove Transmitter, Remove Fuel Valve)
```

Note: Underlined means user can see in menu (isvisible=true).

UC.1.3 Narrative Description

The overall course is ordered sequentially where each block must be completed before proceeding to the next block. The first block is taken as a whole, started when the student chooses the *Fuel Valve Introduction* menu item.

The second block is displayed as three choices. When the student has completed the introduction block, only the first two lessons in the second block are enabled, allowing the student to take either of the first two lessons in any order. Although he may take these two lessons in any order, he must complete both before proceeding to the third lesson (i.e., the first two lessons, which may be taken in any order, are prerequisites to the third lesson).

The third lesson teaches the actual steps needed to perform the fuel valve removal procedure in the required order. At any step in the lesson, he may view the IETM job performance aid for that step in the procedure.

The tests are not enabled until all three lessons have been completed. The first test is a standard question/answer knowledge test consisting of two sections which are ordered in the same sequence that the student selected to take the first two lessons. For example, if the student views the hazards lesson first, then the test questions on the hazards are ordered first. If he fails the test, he is taken back to the lesson related to the section of the test he failed for remediation. After remediation, he is allowed to take the test a second time. If he passes the second time, he is allowed to take the second test. If he fails the first test a second time, he is re-mediated again and fails the course.

After the first test is completed successfully, the student may take the performance-based test. This test consists of a “simulation” of the environment in which the student must “perform” the correct steps in the correct order within a certain time limit. During the simulation, the user may launch and use the IETM much like an open book exam.

UC.2 NETg Precision Learning Use Case

Precision Learning is a NETg sequencing feature based on the mastery status of individual content objects (NETg topics) derived from a ‘block testing’ assessment, wherein many content objects are tested in a single pre-assessment or post-assessment experience. (Block assessment is not unique to NETg, though we were the first to promote it in the commercial training sector). The content objects assessed by an assessment object are typically members of the same content aggregation as the assessment object (i.e., a NETg unit, where a unit consists of lessons, and lessons consist of presentation content objects).

An assessment experience itself is delivered as a content object. Each assessed content object is related to a bank of assessment items through a shared learning objective. Hence, a ‘presentation’ content object has a 1:1 relationship with a learning objective, while an ‘assessment’ content object has a many:1 relationship with learning objectives, and through them, ‘presentation’ content objects. Therefore, to properly indicate the mastery state of a presentation content object, an LMS must be able to associate a score earned in an assessment of a learning objective in an assessment object with the content object that shares the learning objective.

To ensure an appropriate learning experience, and to accommodate the fundamental principle that the locus of control always rests with the learner, NETg has defined a set of behaviors necessary to implement the above ideas. However, current specifications do not allow the specification of any of these behaviors in an interoperable way. It is not possible to specify that a particular unit of content is a pre-assessment, and that another unit of content is a post-assessment, and thus that if the post-assessment has been attempted, the pre-assessment is off limits to the learner. NETg desires an interoperable way to do this, as well as the other behaviors necessary to implement the above behaviors.

UC.2.1 Terminology

NETg needs a specification that provides a generic language for the description of assessments, content, and their related objectives, as well as the rules for entry and sequencing, to allow Precision Learning to interoperate on multiple platforms. This specification should allow for an interoperable description of:

Different types of learning content:
Pre-assessments
Post-assessments
Presentation content
Units, Lessons, or other groups of content

Sequencing rules relating to Precision Learning:
Conditions under which a given content object is inaccessible;
Conditions under which the default is to skip a content object (but the object is still accessible).

UC.2.2 Content Structure

The course consists of a single unit subdivided into a pre-assessment, a post-assessment, and multiple lessons, each lesson having multiple learning objects. Each of the scenarios described below describes a different way of sequencing through these.

```
Unit 1
   Unit 1 Pre-assessment
   Unit 1 Lesson 1
      Unit 1 Lesson 1 Topic 1
      Unit 1 Lesson 1 Topic 2
      …
      Unit 1 Lesson 1 Topic N
   Unit 1 Lesson 2
      Unit 1 Lesson 2 Topic 1
      …
   …
   Unit 1 Lesson N
      …
   Unit 1 Post Assessment
```

UC.2.3 Narrative Description

The Precision Learning use case requires a mechanism for content and the delivery engine to communicate results that correspond to particular learning objectives; this mechanism is assumed to be out of scope for the Simple Sequencing Specification, although it may turn out to be in scope.

Scenario #1

The learner starts a unit of content. The delivery engine presents the learner with the pre-assessment for the unit. The learner successfully answers every question in the pre-assessment. The delivery engine knows that the learner has passed the unit, and moves the learner on to the next unit.

Scenario #2

The learner starts a unit of content. The delivery engine presents the learner with the pre-assessment for the unit. The learner successfully answers some of the questions in the pre-assessment, but is unsuccessful at answering other questions in the unit. The delivery engine knows what objectives the learner has achieved, and as the learner asks for the next item of content (presumably by hitting the ‘next’ button, or similar) presents the learner with only the content corresponding to the unachieved objectives. When the learner has exhausted the content corresponding to the unachieved objectives, the delivery engine presents the learner with the post-assessment.

Scenario #3

The learner starts a unit of content. The delivery engine presents the learner with the pre-test for the unit. The learner fails every question in the pre-assessment. The delivery engine knows that the learner has not achieved any objectives, and presents the learner with all of the content in the unit. When all of the content has been presented, the delivery engine presents the learner with the post-assessment for the unit.

Scenario #4

The learner starts a unit of content by entering at an arbitrary point in the middle of the content. As the learner asks for the next item, the engine provides the learner with the next sequential item in the unit. When the learner reaches the end of the unit, the delivery engine presents the post-assessment for the unit.

UC.3 Click2Learn Blended Content Use Case

This use case describes a learning experience for a soft skill in a learning environment that combines structured canned content with other content resources.

The Sexual Harassment (SH) Behavior Diagnostic course is a fictitious example representing part of a mandatory employee curriculum in a fictional company. It is assumed that the learner will have completed the introduction part of the course in which the learner has been sensitized to the issues covered by the course and given information about the topic. The SH Behavior Diagnostic course teaches the learner how to identify improper behavior and discriminate between innocent and inappropriate or actionable behavior. To successfully complete this course, the learner must correctly classify different behaviors shown in a series of scenarios, and also select the appropriate rationale for each decision.

Because the rationale choices actually contain additional clues, they act as a teaching device as well as for testing. If a learner explores all the choice he or she should be able to deduct the correct answer. The learner can also ask questions from online experts who typically respond within 24 hours; they can also view responses given by experts to questions asked by previous learners. The responses never directly indicate the correct response. The learner can also access the corporate sexual harassment policy documents and a corporate policy glossary at any time while attempting to respond. Access to experts and reference materials is on demand without losing one’s place in the sequence of scenarios. The learner can explore any scenario in any order and change responses as often as desired until the completed course is submitted for evaluation, and can do this over as many sessions as desired, but must finish the assignment within 10 days of beginning it. Once submitted, the choices are evaluated, a score is shown and the learner is invited to review the answers. The review does not allow the learner to change the answers, but includes explanatory feedback for each answer given.

UC.3.1 Terminology

An *activity* consists of several sub-activities. The first is an introduction explaining the rules of the game. The second is the actual diagnostic exercise. The third is the review after submitting the completed exercise. The diagnostic activity consists of viewing and classifying a randomly selected subset of a large collection of scenarios.

UC.3.2 Content Structure

The content consists of introduction material, a collection of scenarios from which a different subset is drawn for each attempt by a learner, different summary feedback to be selected depending on the outcome of the diagnostic exercise, a glossary and reference texts. A collection of questions and responses is provided for delivery situations where the expert community as a real service is not available, as in offline use or where a LMS does not support that feature.

UC.3.3 Narrative Description

The overall learner activity is ordered sequentially. Each major activity must be completed before proceeding to the next major activity. During the scenario exercise, and during the review process, the learner can choose freely among scenarios and revisit the same scenario as many times as desired. If any expert question/responses are available, the learner can choose the question/responses freely. These can be presented discreetly or as a large, scrolling document. If the learner is allowed to retake this course, this is another attempt and all activities must be completed anew; a different set of scenario is presented in the new attempt; this set may contain items already presented in a previous attempt but it may not contain the same set.

UC.4 GIUNTI Conservation of Paintings Use Case

This example envisions a situation in which some students want to obtain a first level certification on conservation of paintings on wood. So if evaluation tests at the end of the modules (see later for explanation) are not passed, the Course is considered failed and the student cannot have a second chance until the Tutor / Administrator of the Course (a.k.a. the Content Provider) allows for a second chance.

The Course titled A general introduction on conservation of paintings on wood is an example of a Course finalized to the first level certification of a restorer. As a first level Course, no particular background is required, except a humanistic one.

The Course teaches how to understand, after a systematic analysis, the best approach to the planning of the type of intervention on a damaged painting on a wood support.

UC.4.1 Terminology

As in the Boeing Use Case, the term Course must be intended in the general assumption of a Learning Object, or Unit of Study. The main blocks are called modules, each module can have one or more lessons. A lesson can be constituted by one or more paragraph. At the end of each lesson there will be a particular paragraph called exercise test (with no score) to allow the student to self evaluate his acquired skills.

At the end of each module there will be an evaluation test (with score) to verify the student's know how and to allow the prosecution to the other modules.

UC.4.2 Content Structure

The Course is comprised of three main blocks (called modules), which are an Introduction to what are paintings on wood, a General overview of the conservation of paintings, and Procedural aspects of planning intervention on paintings.

Two additional blocks (Glossary and Bibliography) are present, and can be launched by the student in any moment. The Glossary will be also accessible from each lesson/paragraph (by means of a button).

```
Introduction on paintings on wood
   Involved materials
      Exercise test
   Historical issues
      Exercise test
   Consumption factors
      Thermo-hygrometric variations
      Atmospheric pollution
      Biological attacks
      Wood growing old
      Light effects
      Previous interventions
      Accidental events
      Exercise test
   Evaluation test
General aspects of conservation of paintings on wood
   What is conservation
      Maintenance
      Prevention
      Restoration
      Exercise test
   Main aspects of the theory of restoration
      Exercise test
   Evaluation test
      Procedural aspects of conservation
   How to know the painting and the environment
      Surveying on previous interventions
      Surveying on materials
      Surveying on the environment
      Exercise test
   The interventions
      Interventions on the environment
      Interventions on the painting
      Exercise test
   Evaluation test
Glossary
Bibliography
```

Note: Underline means that the student can see the item in the menu or Table of Contents.

UC.4.3 Narrative Description

The Course is ordered sequentially at the modules level. Each module must be completed before proceeding to the next block. One module is completed when the evaluation test is passed (the mastery score must be achieved).

A User sequencing behavior is supposed, assuming that the Table of Contents (the underscored items in the Course structure displayed above) is displayed. Initially, only the first module can be chosen. The items of the Table of Contents that actually cannot be delivered have names displayed, e.g., with gray fonts (changed to black fonts when they can be delivered, i.e., when pre-conditions are satisfied).

The first module has three lessons, the first two can be chosen in any order, while the third lesson can be delivered only if the first two are completed. The third lesson has eight paragraphs (not sequenced, they can be delivered in any order). The last paragraph of each lesson is an exercise test, which can be attempted any times. It is useful for the student to check the skills achieved in the lesson.

The evaluation test can be attempted only if the three lessons are completed. The Author decides the mastery score. If the test is passed, the student can start launching the second module. If not, a maximum number of attempts (decided by the Author) can be allowed. At the last attempt, if not passed, the student is invited to send a mail to the Tutor / Administrator asking for a new chance of trying the Course.

The second module has two lessons that can be chosen in any order. Also the paragraphs can be chosen in any order. The same approach of the first module is followed for the delivery of the exercise tests. The evaluation test can be attempted only if the two lessons are completed. The Author decides the mastery score. If the test is passed, the student can start launching the third module. If not, a maximum number of attempts (decided by the Author) can be allowed. At the last attempt, if not passed, the student is invited to send a mail to the Tutor / Administrator asking for a new chance of trying the Course.

The third module has two lessons. The second can be chosen only if the first is completed. The first lesson is composed of three non-visible paragraphs, automatically sequenced by the LMS. The fourth paragraph is the exercise test (visible). The second lesson is composed of three paragraphs, not sequenced and chosen in any order.

The evaluation test can be attempted only if the two lessons are completed. The Author decides the mastery score. If the test is passed, the student has completed the course and will receive by the Tutor / Administrator the certification (this is not a sequencing issue). The event of submit of the evaluation test in this third module is the last event in the last SCO. This is an example of a last event in a Course.

Again, if the evaluation test is not passed, a maximum number of attempts (decided by the Author) can be allowed. At the last attempt, if not passed, the student is invited to send a mail to the Tutor / Administrator asking for a new chance of trying the Course.

About this Document

Title	IMS Simple Sequencing Information and Behavior Model
Editors	Mark Norton (IMS), Angelo Panar (ADL)
Team Co-Leads	Ron Ball (ADL), Brandt Dargue (Boeing)
Version	1.0
Version Date	03 March 2003
Status	**Final Specification**
Summary	This document describes the information and behavior models for the IMS Simple Sequencing Specification
Purpose	Defines the IMS Simple Sequencing Specification
Document Location	http://www.imsglobal.org/simplesequencing/ssv1p0/imsss_infov1p0.html

List of Contributors

The following individuals contributed to the development of this document:

Name	**Organization**
Thor Anderson	IMS Global Learning Consortium, Inc.
Bill Blackmon	Carnegie Mellon University
Fabrizio Cardinali	Giunti Interactive Labs
Travis Carlton	Boeing
Thomas Dinger	IBM
Philip Dodds	Advanced Distributed Learning (ADL)
Leonard Greenberg	Pathlore
Michael Korcuska	DigitalThink
Jon McCarty	Docent
Boyd Nielsen	Thomson NETg
Land Ormiston	Saba
Claude Ostyn	Click2learn
Dan Rehak	Carnegie Mellon University
GianLuca Rolandelli	Giunti Interactive Labs
Brian Taliesin	Microsoft
Schawn Thropp	Advanced Distributed Learning (ADL)
Robert Todd	DigitalThink
Brendon Towle	Thomson NETg
Ed Walker	IMS Global Learning Consortium, Inc.
Michael Vax	WebCT
Jonathan Zempel	IBM

Revision History

Version No.	Release Date	Comments
0.7.5 Public Draft	12 April 2002	First Public Draft released to the public.
1.0 Public Draft	17 October 2002	Numerous editoral updates and additional clarifications. Updated pseudo code. Added introduction section and several diagrams.
1.0 Final	03 March 2003	Made many changes to the document to resolve various technical issues. See the Issues List document for a detailed description of the changes. Added section UC describing the core use cases.

Index