

SCORM 2004 Content Development Guide

Version 1.04

March 2006

The e-Learning Consortium, Japan

Document Revision History

Date	Version	Remarks
December 2005	1.01	Translated from the Japanese document.
February 2006	1.03	Editorial correction
March 2006	1.04	Editorial correction

Table of Contents

<i>Document Revision History</i>	ii
<i>Table of Contents</i>	i
1 Introduction.....	3
2 SCORM 2004 Overview	4
2.1 What is SCORM?	4
2.2 Origins of the SCORM Standard	4
2.3 The LMS Model.....	5
2.4 SCORM 2004 Overview.....	5
2.5 Changes from SCORM 1.0 to SCORM 1.2.....	7
2.6 Changes from SCORM 1.2 to SCORM 2004.....	8
2.6.1 Change to the Versioning of SCORM	8
2.6.2 Addition of the Sequencing Feature	8
2.6.3 Triggering Navigation Requests from SCOs	9
2.6.4 Changes to the SCORM Run-Time Environment	9
2.6.5 Changes to the SCORM Content Aggregation Model.....	10
2.7 Future Evolution of SCORM.....	11
3 Sequencing.....	12
3.1 Content Structures and Learning Objectives	12
3.2 Tracking Information.....	14
3.2.1 Tracking Objective Status and Completion Status	14
3.2.2 Information about Learning Time and Attempt Count.....	15
3.3 Navigation Requests, Sequencing Requests, and Termination Requests	16
3.4 Sequencing Rules.....	18
3.4.1 Sequencing Control Modes.....	19
3.4.2 Limit Conditions	21
3.4.3 Precondition Sequencing Rules	21
3.4.4 Post-Condition Rules and Exit Action Rules.....	24
3.4.5 Rollup Rules.....	25
3.4.6 Local Objectives and Shared Global Objectives	32
3.4.7 Shared Global Objectives and Rule Evaluations	34
3.5 Attempts.....	35
4 Navigation.....	36
4.1 Navigation Control Overview.....	36
4.1.1 SCO Navigation in SCORM 1.2.....	36
4.1.2 SCO Navigation in SCORM 2004.....	37
4.2 Triggering a Navigation Event and SCO Termination	37
4.2.1 SCO Navigation Event Triggered by an SCO	37
4.2.2 Navigation Request Event and SCO Termination	38
4.2.3 Validity of Navigation Request Events.....	39
4.3 Controlling LMS-Provided Navigation Devices	40
5 Run-Time Environment (RTE).....	42
5.1 SCORM 2004 Run-Time Environment Overview	42
5.2 Launching Content Objects.....	43
5.2.1 Assets	43
5.2.2 SCOs	43
5.3 API.....	44
5.3.1 API Overview	44

5.3.2	API Instance Overview	44
5.3.3	Using the API Instance	45
5.3.4	API Method Overview	47
5.3.5	API Instance State Transitions.....	51
5.3.6	API Error Code Overview	52
5.4	Data Model.....	58
5.4.1	Data Model Overview.....	58
5.4.2	Data Model Basics	58
5.4.3	SCORM Run-Time Environment Data Model	63
6	Features of SCORM 2004 Content	72
6.1	Sequencing Features	72
6.1.1	Specifying Sequencing Strategies.....	72
6.1.2	Specifying Sequencing Control Modes	72
6.1.3	Specifying Sequencing Rules	75
6.1.4	Specifying Rollup Rules	80
6.1.5	Specifying Limit Conditions.....	85
6.1.6	Specifying Objectives	86
6.1.7	Specifying Other Controls	91
6.2	Features of SCORM 2004 SCOs	94
6.2.1	RTE Usage Examples	94
6.2.2	Example Uses of Navigation	104
7	Migration from SCORM 1.2 to SCORM 2004.....	107
7.1	Manifest File and SCOs	107
7.2	Converting the Manifest File	107
7.2.1	Basic Structure	107
7.2.2	Content Packaging	108
7.2.3	Changes to the Prerequisites and Masteryscore Elements.....	108
7.2.4	Defining Sequencing Control Modes.....	110
7.2.5	Metadata Description	111
7.3	SCO Migration.....	112
7.3.1	API Instance Name Change	112
7.3.2	API Method Name Changes	112
7.3.3	Changes to Data Model Elements.....	112
7.4	Changes to Error Codes	117
7.5	The Potential of SCORM 2004.....	118
	<i>Index</i>	119
	<i>About This Document</i>	121

1 Introduction

The Sharable Content Object Reference Model (SCORM) has been adopted in Japan as the standard for Web-based e-learning content and learning system development for a few years. During this time, learning management systems (LMS) products, content products and tools that are conformant to SCORM have been widely used both in Japan and overseas. The SCORM standard that has been generally used so far is SCORM Version 1.2, which was released in 2000. Although large quantities of SCORM 1.2 conformant products are widely used, some concerns have been raised as to their functional deficiencies, ambiguities in the specifications and other matters. In 2004, the ADL (Advanced Distributed Learning Initiative) released a new standard called SCORM 2004 in an effort to resolve such concerns and deficiencies in the standard. The purpose of this handbook is to provide a technical overview of this new standard and an explanation of the newly added features and the differences from SCORM 1.2 and its intended audience is those who have some knowledge of SCORM.

SCORM 2004 not only introduces the new features of sequencing and navigation, but it also provides an entire range of detailed specifications. And it is believed that the new standard will satisfy most of the requirements expected by the e-learning industry and the user community. However, the amount of technical content of SCORM 2004 exceeds some 800 pages in all, and it is not easy to grasp the big picture from the individual technical books. To assist those who have some knowledge of SCORM 1.2 and wish to know about SCORM 2004, this handbook covers an overview of the SCORM 2004 standard, the newly added features, and the differences from SCORM 1.2. It is anticipated that the readers will find it easier to understand the standard if they read this handbook first and then scan through the standard books.

Although this document discusses useful points in implementing SCORM 2004 to give a better understanding of the standard mainly to content developers, it is expected to be a useful guide to those who are involved in the planning, design and development of e-learning, not just to the content developers.

The list below shows the SCORM 2004 technical books and their acronyms that this handbook uses as references.

OV: *SCORM 2004 2nd Edition Overview*
CAM: *SCORM Content Aggregation Model Version 1.3.1*
RTE: *SCORM Run-Time Environment Version 1.3.1*
SN: *SCORM Sequencing and Navigation Version 1.3.1*
CR: *SCORM Conformance Requirements Version 1.3*
ADD: *SCORM 2004 2nd Edition Addendum Version 1.2*

2 SCORM 2004 Overview

This section describes the significance of the e-learning standard, emphasizing the newly added features in SCORM 2004 and the changes from SCORM Version 1.2.

2.1 What is SCORM?

SCORM is the Sharable Content Object Reference Model documented and maintained by the Advanced Distributed Learning Initiative (ADL) of the United States of America. It is intended to provide a common standard that will enable the sharing of learning content by making it

- Durable
- Interoperable
- Accessible
- Reusable

SCORM is a foundation reference model. E-learning content based on SCORM can be used without change regardless of any changes to the hardware and software environment (durability), can run in any operating system and Web browser environment (interoperability), can be searched for and discovered whenever required (accessibility) and can be used to develop new learning content (reusability).

2.2 Origins of the SCORM Standard

The SCORM standard and specifications are derived from work done by various industry and technology organizations, including the IMS Global Learning Consortium (IMS), the Aviation Industry CBT Committee (AICC), and the Institute of Electrical and Electronics Engineers Learning Technology Standards Committee (IEEE LTSC). Using these specifications and guidelines, SCORM defines a framework for application to learning content, its aggregation, and its packaging. SCORM also defines a set of conformance requirements for systems that will deliver such content to the learner. SCORM has been influenced by the following:

SCORM 2004 CAM

- IEEE Learning Object Metadata (LOM)
- IMS Content Packaging
- IEEE Extensible Markup Language (XML) Schema Binding for Learning Object Metadata Data Model

SCORM 2004 RTE

- IEEE Data Model For Content Object Communication
- IEEE ECMAScript Application Programming Interface for Content to Runtime Services Communication

SCORM 2004 SN

- IMS Simple Sequencing.

2.3 The LMS Model

Figure 2.1 shows a generalized learning management system (LMS) model. As shown, an LMS provides various services such as a learner profile service and a content management service. However, SCORM does not address the specific implementation of such services. It provides a set of specifications only for the interface points between learning content and the LMS. It merely defines the rules for registering content to the LMS, launching the content and exchanging data between the content and the LMS.

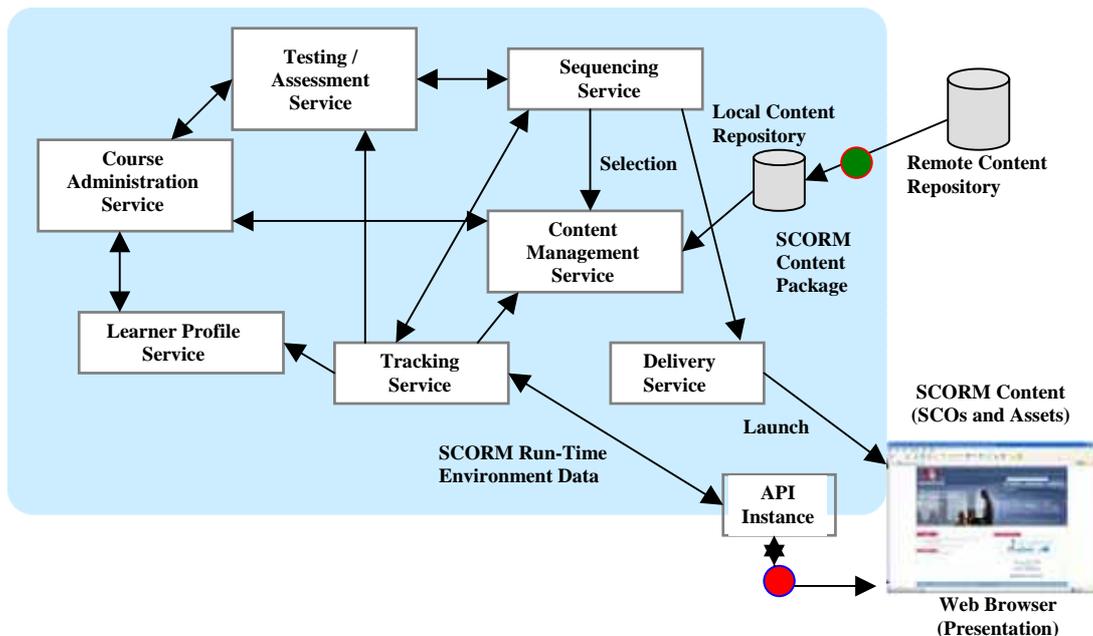


Figure 2.1 An LMS Model

2.4 SCORM 2004 Overview

The ADL released SCORM 2004 in 2004 as the successor standard to SCORM 1.2. SCORM 2004's most significant difference from its predecessor is the addition of sequencing and navigation features. Thanks to the enhancements introduced in the new standard, content developers can enjoy more freedom in their content design and development. For example, the dynamic behavior of content in response to the learning experience sequence and the learner's progress – which could not be described under the earlier versions – can now be controlled by the content. To complement LMS-provided user interface devices for navigation, content can now trigger navigation request events to allow use of the content's own navigation command buttons like [Next] and [Back].

The specifications and standards of SCORM 1.2 were bundled into three technical books: the *SCORM Overview* book, the *SCORM Content Aggregation Model* book and the *SCORM Run-Time Environment* book. For SCORM 2004, the *SCORM Sequencing and Navigation* book has been added to cover the sequencing and navigation specifications, so there are now four technical books.

(1) The SCORM 2004 Overview Book

This book describes the history and objectives of the ADL Initiative and SCORM. It also covers the specifications and standards the SCORM has borrowed, and explains how the SCORM books relate to each other.

(2) The SCORM Content Aggregation Model (CAM) Book

This book covers what content developers should know when they design SCORM compliant learning content; that is, it provides a set of guidelines on how to describe the identity of learning content components, and explains how to assemble and package the components. These guidelines are based on IEEE Learning Object Model (LOM) 1484.12, the AICC guidelines on content structure, IMS Content Packaging and IMS Simple Sequencing.

The major SCORM technical topics covered in this book are sharable content objects (SCOs), assets, content aggregation, content packaging, the package interchange file (PIF), metadata, the manifest file, sequencing and navigation.

(3) The SCORM Run-Time Environment (RTE) Book

This book covers the requirements of the learning management system (LMS) for managing a Web-based Run-Time Environment in terms of launching a learning content object, exchanging data with the content object, and tracking the learner's progress. These guidelines are based on IEEE Application Programming Interface (API) 1484.11.2 and IEEE Data Model 1484.11.1.

The major SCORM technical topics covered in the book include the Application Programming Interface (API), the API Instance, content object launch, session data support methods, and the Run-Time Environment Data Model.

(4) The SCORM Sequencing and Navigation (SN) Book

This technical book has been added for SCORM 2004, and it marks the most significant evolution of SCORM. This book provides guidelines on sequencing; i.e., how to deliver content to the learner. These guidelines are based on the IMS Sequencing Information and Behavior Model. Guidelines on the navigation and user interface devices are also based on this specification.

The major SCORM technical topics covered in this book include activity trees, objectives, sequencing information, navigation information, and the run-time navigation data model.

While each of the SCORM technical books focuses on specific aspects of SCORM, there are some overlapping areas among the books, and those areas are described in such a way that they can be easily referred to from each other.

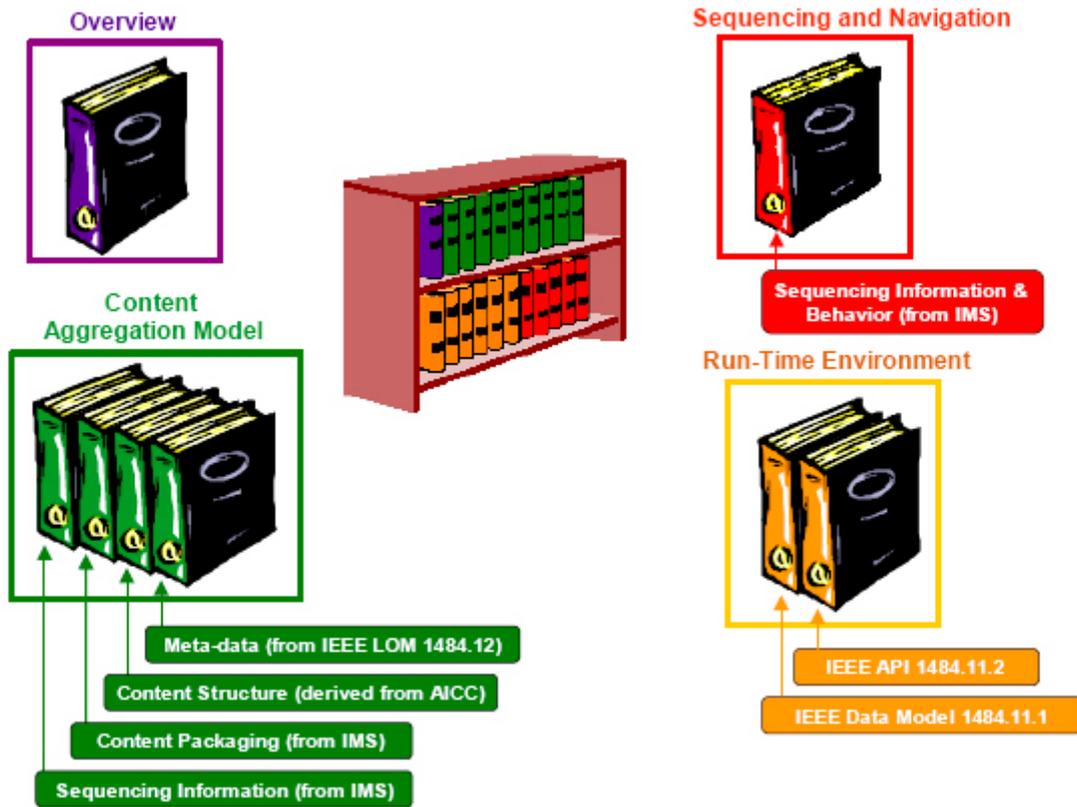


Figure 2.2 SCORM Books (Source: *SCORM 2004 2nd Edition Overview* by ADL)

2.5 Changes from SCORM 1.0 to SCORM 1.2

Many changes have been made in the SCORM specification from the previous SCORM versions. The forces behind this revision include the need to clarify concepts and requirements, changes resulting from standardization efforts, the adoption of best practices from the ADL community, and the provision of enhancements and bug fixes.

SCORM entered its experimentation and evaluation step with SCORM 1.0. The participants in the experimentation and evaluation effort regarding SCORM 1.0 raised a number of questions and issues based on what they encountered during the implementation.

Instead of expanding the scope of SCORM 1.0 in the new version, the specifications and guidelines were modified and improved in SCORM 1.1 on the basis of the feedback from the early participants.

The most noticeable change introduced in SCORM 1.1 was a name change. That is, while SCORM was an acronym for the Sharable Courseware Object Reference Model in SCORM 1.0, it has stood for the Sharable Content Object Reference Model from SCORM 1.1 onwards. This reflects the reality that the standard and specifications in SCORM are applied to various levels of content rather than just courseware as a whole. Another significant change in SCORM 1.1 was that the standards and specifications were subdivided into different functional groups to improve ease of use.

There were two more notable changes introduced in SCORM 1.1. First, a few data model elements were removed from the run-time data model as a result of a simplification introduced to the AICC CMI Recommendations and Guidelines, upon which the SCORM run-time data model was based. Second, a major improvement was made in the API for the Run-Time Environment.

In SCORM 1.2, the SCORM content package application profile was added on the basis of the IMS Content Packaging specifications. In addition, metadata was updated to reflect the latest specifications developed by IMS and IEEE LTSC. This update included changes made to the data model and XML binding specifications. Furthermore, from this version onwards the metadata application profile was renamed the SCORM Content Aggregation Model and a new naming convention was adopted to match the IMS Content Packaging specifications.

2.6 Changes from SCORM 1.2 to SCORM 2004

This section describes the changes and improvements made in SCORM 2004 compared to SCORM 1.2.

2.6.1 Change to the Versioning of SCORM

To improve the independence and maintainability of the SCORM technical books, the versioning of SCORM has been changed as from SCORM 2004 to allow each SCORM book to have its own version, as in CAM and RTE “Version 1.3”. Future changes will apply only to those books affected and will be reflected in their version numbers only.

2.6.2 Addition of the Sequencing Feature

For SCORM 2004, sequencing and navigation specifications have been added as a new SCORM technical book. Learning sequence controls were not part of the SCORM specifications up to SCORM 1.2, but they can now be described. For example, the delivery sequence of learning content can now be controlled by changing the type and sequence of the content to be learned based on the result of pretesting done before the start of a lesson. In this case, a learner is regarded as having completed a course after successfully answering questions A and B, while a learner who does not answer successfully is asked to repeat the lesson or repeatedly go through explanation screens before taking the test again; this continues until a learning objective is met.

Thus, it is now possible for content developers to control content behavior by describing a content structure and its associated sequencing rules in the manifest file.

As a learning path or status can be defined by combining various conditions, such as how the learner satisfies an objective and how he or she progresses through the lesson, content developers can now create adaptable content or a simulation package specific to a particular learner.

2.6.3 Triggering Navigation Requests from SCOs

The SCORM 2004 Sequencing and Navigation book introduces a specification regarding the navigation requests that can be made from SCOs.

SCOs are allowed to trigger navigation request events such as [continue] and [previous] from within the SCOs. In addition, SCOs can now request whether LMS-provided navigation user interface (UI) devices are to be shown or hidden.

To enable an SCO to trigger a navigation event, the content developer should add an API method call in the SCO that will set a value to a run-time data model element. The content developer should also add a description in the manifest file regarding how to control LMS-provided UI devices.

The new specification allows content developers to design a navigation function, which is an important content factor, without regard to the type of LMS under which the content will run.

2.6.4 Changes to the SCORM Run-Time Environment

The Run-Time Environment specification has been greatly changed in SCORM 2004 from SCORM 1.2. This section briefly summarizes these changes.

(1) API Instance Name Change

The API Instance has been renamed from API to API_1484_11.

(2) API Method Name Change

Table 2.1 Changes to API Methods

SCORM1.2	SCORM2004
LMSInitialize(“”)	Initialize(“”)
LMSFinish(“”)	Terminate(“”)
LMSGetValue(parameter)	GetValue(parameter)
LMSSetValue(parameter_1,parameter_2)	SetValue(parameter_1,parameter_2)
LMSCommit(“”)	Commit(“”)
LMSGetLastError()	GetLastError()
LMSGetErrorString(parameter)	GetErrorString(parameter)
LMSGetDiagnostic(parameter)	GetDiagnostic(parameter)

(3) Data Model Change

The major changes are as follows:

- All data model elements defined in SCORM must be implemented and supported by the LMS.
- The data model has been flattened in SCORM, and the *cmi.core* and *cmi.student_data* elements have been removed.
- The data model element for the interactions between an SCO and its Run-Time Environment has become more sophisticated with the introduction of a more precise format for describing data concerning the learner’s responses to tasks and responses to questions.

- As data model element names and their values are bound to character strings encoded in Unicode (ISO 10646), multi-language implementation has been realized, including double-byte languages.
- The *lesson_status* data model element has been removed, and in its place the *completion_status* and *success_status* elements have been introduced. The possible state token values of the *completion_status* data model element are *completed*, *incomplete*, *not_attempted* or *unknown* to indicate whether the learner has completed an SCO. The possible state token values of the *success_status* data model element are *passed*, *failed* or *unknown* to indicate whether the learner has mastered an objective. The *browsed* state token is no longer used.
- The *score.scaled* data model element has been introduced to indicate the performance level of the learner for an objective. The old *score.raw* data model element with a range from 0 to 100 points has been removed.
- The *objectives* data model element is mapped with the learning objectives of an activity, and shared global objectives can now be defined.
- The introduction of a wider range of error codes makes it possible to check the status of the API Instance and the validity of data.

2.6.5 Changes to the SCORM Content Aggregation Model

The SCORM content aggregation model has been changed to reflect the introduction of sequencing and navigation specifications, and the affected XML schemas have been modified.

In addition, the following ADL content packaging extension elements were removed:

- `<adlcp:prerequisites>`
- `<adlcp:masteryscore>`
- `<adlcp:maxtimeallowed>`

The conditions that were previously defined with these elements are now described in terms of corresponding sequencing rules.

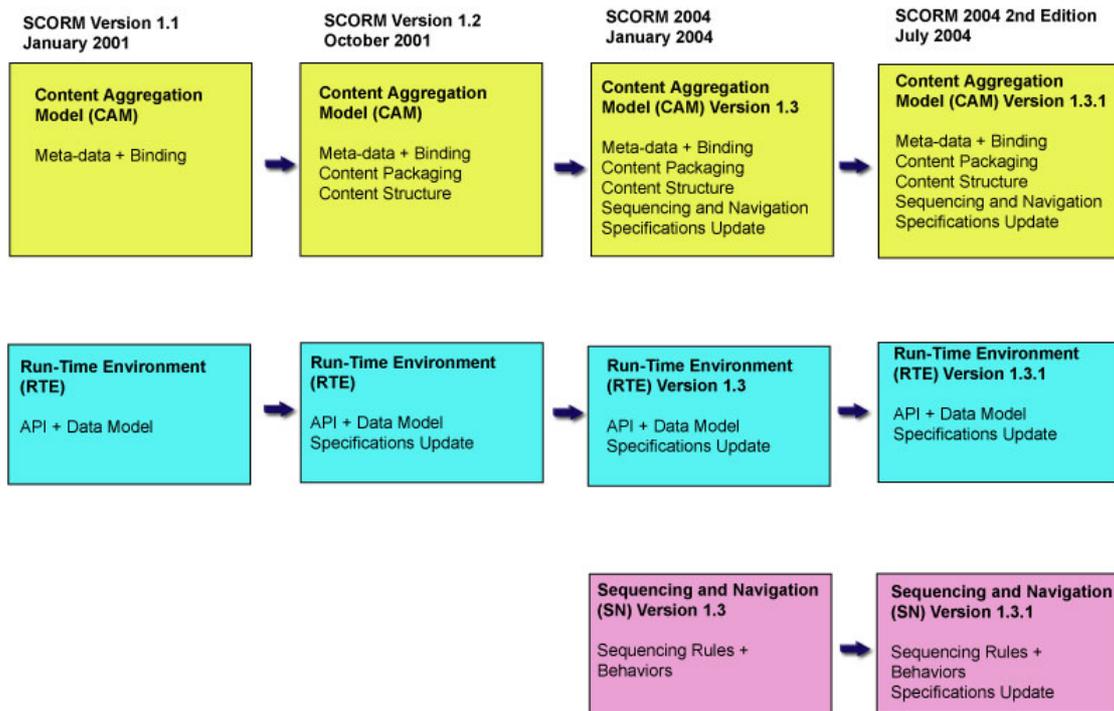


Figure 2.3 SCORM Evolution

2.7 Future Evolution of SCORM

The ADL lists the following as future tasks for Web-based learning functions:

- Designing new run-time and content data model architectures
- Incorporating simulations
- Incorporating electronic performance support objects
- Implementing SCORM-based intelligent tutoring capabilities
- Designing a new content model
- Incorporating gaming technologies

Note, however, that the ADL has no current plans to release the next version.

3 Sequencing

This section explains the sequencing feature that is the main enhancement in SCORM 2004. It covers the basic concept of sequencing and its relationship with other components, and explains how sequencing behaviors are described.

Figure 3.1 shows an overview of a sequencing behavior. Content developers control the behavior of content by describing a content structure and a set of sequencing rules associated with the structure in the manifest file (*imsmanifest.xml*). The LMS reads the manifest file and performs actions described in the file. When it receives a navigation request from the learner, the LMS updates the learner's status data (the tracking data) to reflect the learner's progress status, interprets the sequencing rules, decides on the next content to be delivered, and then delivers it to the learner's environment. The LMS repeats this process until the lesson is terminated.

The major components of sequencing and external functions are described with the following:

- Content structure and learning objectives
- Tracking information
- Navigation requests and sequencing requests
- Sequencing rules

Run-time sequencing behaviors are described as a set of processes shown on the right-hand side of Figure 3.1. The behavior of each process is defined as pseudo-code. Section 6 gives a detailed explanation of the processes and pseudo-code. This section explains the external functions of sequencing with the above four components.

3.1 Content Structures and Learning Objectives

In SCORM 2004, a content structure is described as a hierarchical tree structure. Each node in the tree is called an activity, and the content structure is represented as an activity tree. An activity that has no child activity attached to it is called a leaf activity and is associated with a learning resource (an SCO or asset) that can be delivered to the learner's screen.

When an activity has one or more child activities attached to it, the set is called a cluster. For example, Activities 1.1.3, 1.1.3.1, and 1.1.3.2 in Figure 3.1 constitute a cluster with Activity 1.1.3 as the parent, while Activities 1.1, 1.1.1, 1.1.2, and 1.1.3 constitute another cluster with Activity 1.1 as the parent. A cluster is the basic unit for sequencing behavior, and in many cases the sequencing rules that are defined for a parent activity are applied to the cluster.

Each activity is always assigned with at least one objective, and this is called the primary objective¹. The role of the primary objective is discussed in the section on

¹ The primary objective is also called a rollup objective. This name is given because it is represented by the PrimaryObjective element in the manifest file. In the SN book, this is called the rollup objective because among the objectives associated with an activity, this objective has the *Objective Contributes*

rollup below. Activities and objectives hold tracking information, which will be discussed in Section 3.2. In addition to the default objective, content developers can associate any number of global objectives to each activity. These global objectives can be shared between multiple activities. Therefore an activity can be associated with multiple shared global objectives in addition to the default objective, and each shared global objective may be shared by multiple activities. A read and write relationship can be defined between a shared global objective and activities. The tracking information status of a shared global objective is determined by the value of the tracking information written from an activity. The activity can read the tracking information of a shared global objective, and refer to it in the application of sequencing rules. Shared global objectives are discussed in detail in Section 3.4.6.

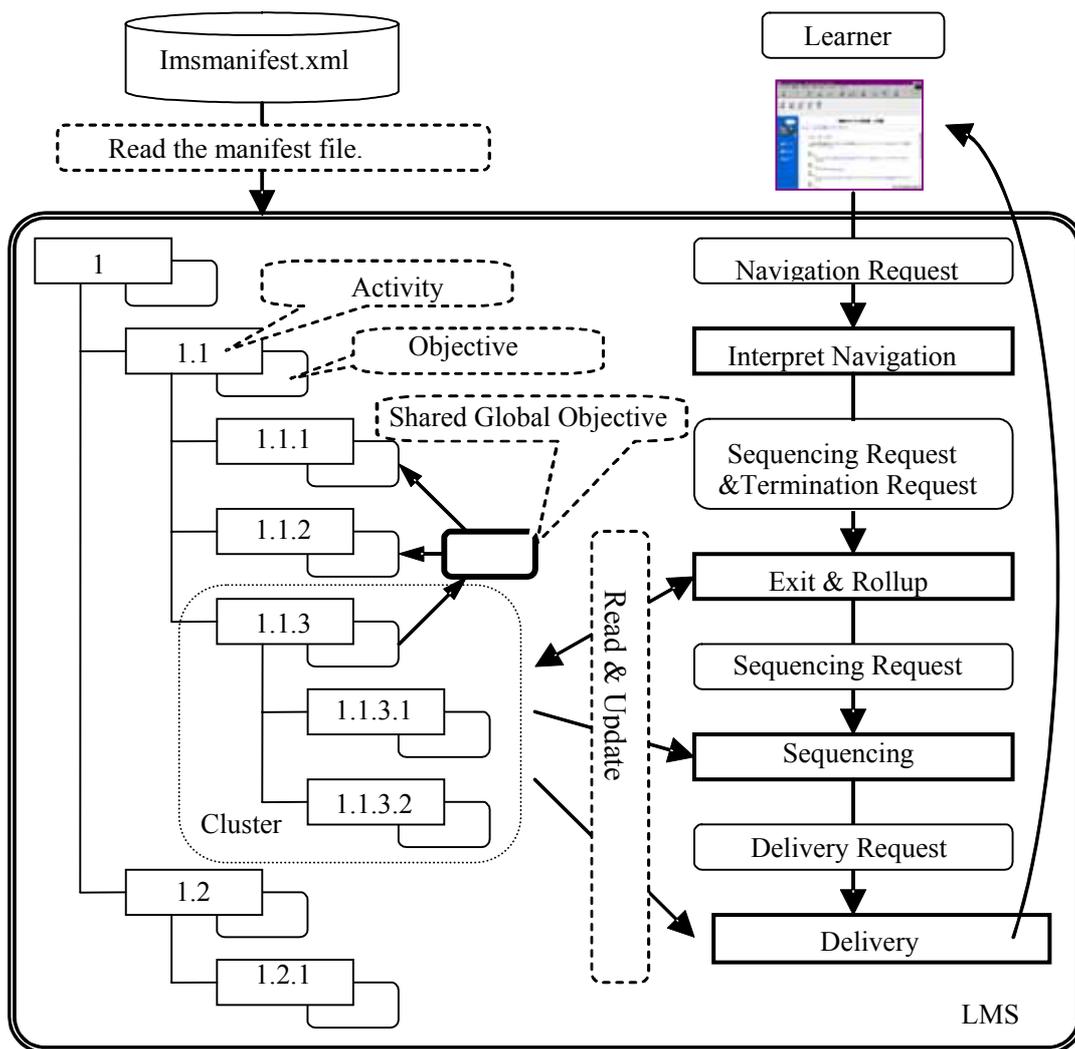


Figure 3.1 Sequencing Behavior Overview

To *Rollup* attribute set to *True*. They represent the same entity. The term “primary objective” is used in the document.

3.2 Tracking Information

The tracking information is information that reflects the learner’s learning status, and it is associated with each activity and its objectives. Table 3.1 shows the details of tracking information.

The tracking information can be divided into data concerning learning performance and completion on one hand and data concerning the learning time and number of attempts on the other. An attempt here means a learner’s effort to complete an activity. It refers to an effort from the time when an SCO is launched to the time when it is terminated. It may also refer to a learning effort that starts with a child activity of an intermediate parent activity and continues until the learner navigates out of that cluster. An activity may be attempted more than once, so multiple attempts may exist for an activity. The details of attempts are discussed in 3.5.

Table 3.1 Tracking Model

Objective Progress Information	Activity Progress Information	Attempt Progress Information
<i>Objective Satisfied Status</i>		<i>Attempt Completion Status</i>
<i>Objective Normalized Measure</i>		<i>Attempt Completion Amount</i>
	<i>Activity Absolute Duration</i>	<i>Attempt Absolute Duration</i>
	<i>Activity Experienced Duration</i>	<i>Attempt Experienced Duration</i>
	<i>Activity Attempt Count</i>	

3.2.1 Tracking Objective Status and Completion Status

SCORM 2004 makes it possible to manage “completion” and “satisfaction” independently. This is to accommodate a situation where a learner completes an activity from start to finish but does not succeed in attaining its objectives, or a situation where the learner masters the content without completing the learning activity.

“Satisfaction” is related to objectives, indicating whether the learner has satisfied or not satisfied the objective or to what extent the learner has satisfied the objective. These indications correspond to the *Objective Satisfied Status* and the *Object Measure Status* elements of the Tracking Model shown in Table 3.1.

“Completion” is related to attempts on activities. It indicates whether the learner has completed the attempt or not or to what extent the learner has completed the attempt. These indications correspond to the *Attempt Completion Status* and *Attempt Completion Amount* elements of the Tracking Model shown in Table 3.1.

These data elements of a leaf activity are updated by the associated SCO using the Run-Time Environment Data Model. Table 3.2 shows the correspondence between the Tracking Model and the Run-Time Environment Data Model.

In the case of a cluster, however, the status information about the parent activity is updated based on its children's status information. For the whole activity tree, tracking information is propagated from SCO to leaf activity, then its parent, and finally to the root activity of the activity tree. This recursive process is called a rollup² behavior. A content developer decides how a parent activity's information is to be updated through a rollup process. Section 3.4.5 describes this rollup process.

Table 3.2 Correspondence between Tracking Model and Run-Time Environment Data Model

Tracking Model		Run-Time Environment Data Model
<i>Attempt Completion Status</i>		<i>cmi.completion.status</i>
<i>Attempt Completion Amount</i>		<i>cmi.progress.measure</i>
<i>Objective Satisfied Status</i>	Primary Objective	<i>cmi.success.status</i>
	Other Objectives	<i>cmi.objectives.n.status</i>
<i>Objective Normalized Measure</i>	Primary Objective	<i>cmi.score.scaled</i>
	Other Objectives	<i>cmi.objectives.n.score.scaled</i>

3.2.2 Information about Learning Time and Attempt Count

The amount of learning time with an activity is managed with the *Attempt Absolute Duration*, *Attempt Experienced Duration*, *Activity Absolute Duration*, and *Activity Experienced Duration* data elements.

The *Attempt Absolute Duration* refers to the duration from the start of one attempt to the end of the attempt. The *Attempt Experienced Duration* refers to the duration from the start of one attempt to the end of the attempt, excluding the elapsed time while the attempt was suspended. If the attempt was not suspended, the values of these two elements are identical.

The *Activity Absolute Duration* refers to the cumulative duration of all attempts on the activity by a specific learner, and the *Activity Experienced Duration* refers to the cumulative duration of all attempts (excluding suspensions) on the activity by a specific learner.

² An attempt's completion amount is not subject to rollup in the current specification.

The *Activity Attempt Count* refers to the number of attempts on the activity by a specific learner.

This information is collected at run-time by the LMS.

3.3 Navigation Requests, Sequencing Requests, and Termination Requests

Navigation requests refer to requests like [Continue] and [Previous] invoked by a learner. Table 3.3 shows the types of navigation requests. When a navigation request is made by the learner from the Web browser, LMS-provided user interface devices may be used or a request from the SCO may be honored. Section 5 explains how to issue a navigation request from an SCO.

A navigation request is translated into a termination request and a sequencing request by the navigation interpretation process shown in Figure 3.1. Table 3.3 lists navigation requests and the corresponding sequencing requests. Tables 3.4 and 3.5 explain each of them. A sequencing request triggers the launching of a whole learning process, and traversal from one activity to another. A termination request triggers the termination and suspension of a learning process.

Some sequencing requests and termination requests may be translated into other sequencing requests and termination requests by the post-condition rules shown in Section 3.4.4. The *Retry* sequence request shown in Table 3.4 is generated by the post-condition rules rather than with a navigation request. In response to a sequencing request, the LMS switches from the current activity to another activity and decides on the next activity to be delivered to the learner. In this case, the LMS processes the limit conditions and precondition rules that are described in Sections 3.4.2 and 3.4.3.

Table 3.3 Navigation Requests

Name	Description	Sequence Request	Termination Request
<i>Start</i>	Start a new learning process on the activity tree	<i>Start</i>	
<i>Resume All</i>	Resume a suspended learning process on the activity tree	<i>Resume All</i>	
<i>Continue</i>	Proceed to the next activity	<i>Continue</i>	<i>Exit</i>
<i>Previous</i>	Go back to the previous activity	<i>Previous</i>	<i>Exit</i>
<i>Forward</i>	Not specified in the current version of SCORM		
<i>Backward</i>	Not specified in the current version of SCORM		
<i>Choice</i>	Proceed to the selected activity	<i>Choice</i>	<i>Exit</i>
<i>Exit</i>	Terminate the current activity	<i>Exit</i>	<i>Exit</i>
<i>Exit All</i>	Terminate the current activity and all of its ancestors in the tree	<i>Exit</i>	<i>Exit All</i>
<i>Suspend All</i>	Terminate the current attempt on the current activity and all of its ancestors after saving the tracking information so that the learning process may be resumed in the future	<i>Exit</i>	<i>Suspend All</i>
<i>Abandon</i>	Abandon the current attempt on the current activity	<i>Exit</i>	<i>Abandon</i>
<i>Abandon All</i>	Abandon the current activity and all of its ancestors in the tree	<i>Exit</i>	<i>Abandon All</i>

Table 3.4 Sequencing Requests

Name	Description
<i>Start</i>	Start a new activity
<i>Resume All</i>	Resume a suspended activity
<i>Continue</i>	Proceed to the next activity
<i>Previous</i>	Go back to the previous activity
<i>Choice</i>	Proceed to the selected activity
<i>Exit</i>	Terminate the current activity
<i>Exit All</i>	Terminate the current activity and all of its ancestors in the tree

Table 3.5 Termination Requests

Name	Description
<i>Exit</i>	Terminate the current activity
<i>Exit All</i>	Terminate the current activity and all of its ancestors in the tree
<i>Suspend All</i>	Terminate the current attempt on the current activity and all of its ancestors after saving the tracking information so that the activities may be resumed in the future
<i>Abandon</i>	Abandon the current attempt on the current activity
<i>Abandon All</i>	Abandon the current activity and all of its ancestors in the tree

3.4 Sequencing Rules

Sequencing rules are used by content developers to define sequencing behaviors. The sequencing rules are broadly classified as explained below. Note that these sequencing rules are defined for each activity.

- Rules limiting sequencing requests and transition behavior between activities. These rules are applied on the basis of either a predetermined condition or the tracking information. The former case is called a sequencing control mode. For example, one of the sequencing control modes specifies whether each child activity in an activity tree can be accessed in the forward direction only and reverse navigation is prohibited. The latter type of rules consist of precondition rules, such as *“if the Objective Satisfied Status is True, then the activity is skipped,”* and control condition rules, such as *“the total accumulated learning time for the activity must be less than 30 minutes.”*
- Rules for issuing a specific sequencing request when the tracking information satisfies a certain condition. The rules of this type are called post-condition rules. For example, *“if the activity’s objective status is not satisfied, then retry the activity.”* Post-condition rules are evaluated at the Termination and Rollup stage shown in Figure 3.1.
- Rules for updating tracking information. As discussed in Section 3.2, the objective progress information for an activity is updated with a trigger event generated when a learner provides some input to the current SCO. The information is rolled up from the activity associated with the SCO through to the root activity. This update process is called the rollup behavior. In these rules, it is possible to describe whether an activity is to contribute to the rollup, conditions under which a rollup occurs, and an appropriate action to be taken when a condition is met. For example, it is possible to describe a rule indicating that *“if three of its child activities are completed, then the parent activity is regarded as completed.”* These rules are evaluated at the Termination and Rollup stage shown in Figure 3.1.

The types of sequencing rules discussed above can be summarized from the sequencing behavior standpoint as follows:

- (1) Tracking information update
Rollup rules are evaluated at the Termination and Rollup stage shown in Figure 3.1, and the tracking information at each activity of the activity tree is updated.
- (2) Confirming a sequencing request
Post-condition rules are evaluated at the Termination and Rollup stage shown in Figure 3.1, and if the post-condition rules are satisfied, the sequencing request based on the navigation request from the learner is replaced with the sequencing request generated by the precondition rule.
- (3) Deciding on an activity for delivery
At the Sequencing and Delivery stages shown in Figure 3.1, a target activity is selected for delivery. At this time, the next activity is selected with reference to the sequencing control mode, precondition rules, and the limiting conditions.

Each type of sequencing rule is explained in detail in the sections below.

3.4.1 Sequencing Control Modes

Sequencing control modes control the sequencing behavior for a cluster, and they are classified broadly into the following types:

- Those used to make specific navigation requests effective (*Sequencing Control Choice, Sequencing Control Flow*)
- Those to add a limitation to transition between activities (*Sequencing Control Choice Exit, Sequencing Control Forward Only*)
- Those to control how to evaluate tracking information (*Use Current Attempt Objective Information, Use Current Attempt Progress Information*).

Table 3.6 describes these sequencing control modes.

Table 3.6 Sequencing Control Modes

Name	Description
<i>Sequencing Control Choice</i>	A <i>Choice</i> navigation request is permitted to target the children of the activity
<i>Sequencing Control Choice Exit</i>	If false, it is prohibited to move from the activity or its descendent to another activity through the <i>Choice</i> navigation request
<i>Sequencing Control Flow</i>	Continue and Previous navigation requests are valid in the cluster
<i>Sequencing Control Forward Only</i>	Backward movement is prohibited in the cluster
<i>Use Current Attempt Objective Information</i>	The Objective Progress Information for the current attempt of the activity will be used in rule evaluations and rollup
<i>Use Current Attempt Progress Information</i>	The Attempt Progress Information for the current attempt of the activity will be used in rule evaluations and rollup

3.4.1.1 Sequencing Control Choice and Sequencing Control Flow

The *Sequencing Control Choice* element is used to allow the learner to freely choose an activity from a list of activities when the learner proceeds to the next activity. If the *Sequencing Control Choice* element of a parent activity is set to *True*, then the learner can proceed to any of the child activities with a *Choice* sequencing request. If the sequencing control mode value is *False*, the learner is not allowed to proceed to any of the child activities with a *Choice* sequencing request.

The *Sequencing Control Flow* element is used to decide the delivery sequence of the child activities of the target parent with a *Continue* or *Previous* sequencing request. If the value of the *Sequencing Control Flow* element is *True*, the learner is allowed to proceed forward or backward to each child activity with, respectively, a *Continue* or a *Previous* sequencing request. If the sequencing control mode value is *False*, the learner is not allowed to move within the cluster with a *Continue* or *Previous* sequencing request.

3.4.1.2 Sequencing Control Choice Exit

The *Sequencing Control Choice Exit* element is used to limit movement from the activity or its child activities to another activity with a *Choice* sequencing request. If the value of the *Sequencing Control Choice Exit* element of an activity is *False*, it is not possible to proceed to another activity from that activity or its descendent activities with a *Choice* sequencing request. For a *Choice* sequencing request to be effective, the value of the *Sequencing Control Choice Exit* element defined for the current activity and all its ancestor activities must be *True*. Therefore, this sequencing control mode makes it possible to prohibit an exit from the set of activities below a particular parent activity through a *Choice* sequencing request.

3.4.1.3 Sequencing Forward Only

The *Sequencing Forward Only* element is used to direct movement between activities to forward only within a cluster, and to prohibit reverse movement. If the value of the *Sequencing Control Forward Only* element of an activity is *True*, *Previous* and *Choice* sequencing requests are not allowed among its child activities. However, these sequencing requests are valid if the value is *False*.

3.4.1.4 Use Current Attempt Objective Information and Use Current Attempt Progress Information

These data elements are used to indicate whether the *Attempt Objective Information* and *Attempt Progress Information* for the cluster will reflect only the information gathered from the current attempt or use the latest information, including that from the most recent previous attempt on the cluster's child activities. If the values of these elements of the parent activity are *True*, the information from the current attempt only is reflected. At the current attempt on the associated cluster, the objective information and progress information about the child activities that have not yet been delivered are considered "*Unknown*". If the values of the *Attempt Objective Information* and *Attempt Progress Information* elements are *False*, then the latest information, including that from the most recent previous attempt, is used. At the current attempt on the associated cluster, the objective information and progress information regarding the child activities that have not yet been delivered are adjusted to the information at the end of the most recent previous attempt.

Figure 3.2 illustrates the above process. In Figure 3.2, the *Attempt Objective Information* and *Attempt Progress Information* elements of the parent are set to *True* in a) and *False* in b).

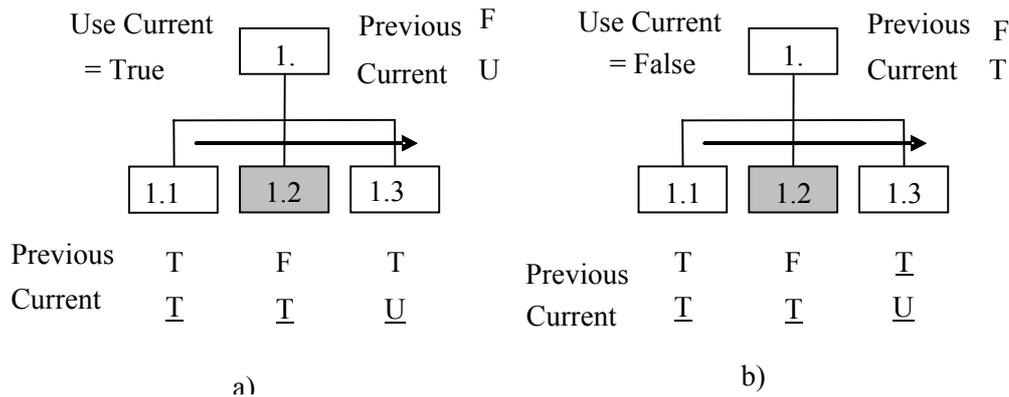


Figure 3.2 Use Current Attempt Objective and Progressive Information

In both cases, the objective satisfied and attempt completion status of the child activities for the previous attempt are set to *True* for 1.1, *False* for 1.2 and *True* for 1.3. In the case of Activity 1, the rolled up status is *False*, based on the default rollup rule explained in Section 3.4.5 for the *And* combination of 1.1, 1.2, and 1.3. At the current attempt, activities 1.1 and 1.2 have been completed with 1.1 and 1.2 being set to *True*.

How should the status information of parent activity 1 be determined in this situation? If it is to be based on the current attempt information, the status of Activity 1 should be set to *Unknown* (as shown in a) above) with the *And* combination of 1.1, 1.2, and 1.3 as Activity 1.3 has not been attempted. If the past information should also be considered, on the other hand, the latest information not only about the current attempt but also about the most recent previous attempt is considered, and the status value of Activity 1 becomes *True* (as shown in b) in the diagram) because the status of 1.3 was *True* at the previous attempt. Content developers may choose which information is used for each cluster.

3.4.2 Limit Conditions

A limit condition can be defined to indicate that an activity is not allowed to be delivered. The only means of implementing a limit condition under the current SCORM specifications is to use the *Limit Condition Attempt Count* element. If an attempt count limit is defined for an activity, that activity cannot be attempted for more than the specified number of times.

3.4.3 Precondition Sequencing Rules

Precondition sequencing rules are used to define conditions for limiting the delivery of activities. These rules are similar to limit conditions in the sense that they are used to control the delivery of activities.

Precondition rules are defined for each activity, and more than one rule may be defined for one activity. The precondition sequencing rules are described in the following format:

If [*condition set*] **Then** [*action*].

The condition set is a collection of conditions to be evaluated as *True* or *False* with respect to the tracking information for the activity. The action part indicates what limit is to be imposed on the delivery of the activity. Some example precondition sequencing rules are shown below:

If Satisfied Then Skip

If the objectives of the activity are satisfied, it is to be skipped.

If Attempted Then Disabled

If the activity has been attempted, the delivery of the activity is to be disabled.

If Always Then Hidden from Choice

At all times, this activity must not be used as a Choice navigation target.

3.4.3.1 Condition Set of Precondition Sequencing Rules

The condition set of a precondition sequencing rule is defined in the following format:

condition_combination ((condition_operator, condition_element),)

Therefore, a condition set is one or more pairs of a condition operator and a condition element combined with a condition combination. Each of these elements is explained below.

- Condition combination: There are two condition combination elements: *All* and *Any*. When *All* is used as a condition combination, the condition set is evaluated as *True* only if all the subsequent condition elements are evaluated as *True*. If the *Any* condition combination is used, the result of the condition set becomes *True* if any one condition element is evaluated as *True*. The default condition combination is *Any*.
- Condition operator: There are two types of rule condition operator: *NO-OP* and *Not*. The *NO-OP* condition operator does not change the Boolean value of the corresponding condition element. The *Not* condition operator negates the Boolean value of the condition element.
- Condition element: The evaluation result of a rule condition element depends on the tracking information for the activity. Table 3.7 shows a list of rule condition elements. If the target tracking information is the *Objective Satisfied Status* information or the *Objective Normalized Measure* information regarding the activity, the target objective is specified by the *Rule Condition Referenced Objective* element. For the *Objective Normalized Measure* element, the target threshold is specified by the *Rule Condition Measure Threshold*.

Table 3.7 List of Rule Condition Elements

Condition	Tracking information	Description
<i>Satisfied</i>	<i>Objective Satisfied Status</i>	The condition is evaluated as <i>True</i> if the <i>Objective Satisfied Status</i> for the target objective is <i>True</i>
<i>Objective Status Known</i>	<i>Objective Satisfied Status</i>	The condition is evaluated as <i>True</i> unless the <i>Objective Satisfied Status</i> for the target objective is <i>Unknown</i>
<i>Objective Measure Known</i>	<i>Objective Normalized Measure</i>	The condition is evaluated as <i>True</i> unless the <i>Objective Normalized Measure</i> for the target activity is <i>Unknown</i>
<i>Objective Measure Greater Than</i>	<i>Objective Normalized Measure</i>	The condition is evaluated as <i>True</i> if the <i>Objective Normalized Measure</i> for the target objective is greater than the <i>Rule Condition Measure Threshold</i>
<i>Objective Measure Less Than</i>	<i>Objective Normalized Measure</i>	The condition is evaluated as <i>True</i> if the <i>Objective Normalized Measure</i> for the target objective is less than the <i>Rule Condition Measure Threshold</i>
<i>Completed</i>	<i>Attempt Completion Status</i>	The condition is evaluated as <i>True</i> if the <i>Attempt Completion Status</i> for the activity is <i>True</i>
<i>Activity Progress Known</i>	<i>Attempt Completion Status</i>	The condition is evaluated as <i>True</i> unless the <i>Attempt Completion Status</i> for the activity is <i>Unknown</i>
<i>Attempted</i>	<i>Activity Attempt Count</i>	The condition is evaluated as <i>True</i> if the <i>Activity Attempt Count</i> for the activity is 1 or more
<i>Attempt Limit Exceeded</i>	<i>Activity Attempt Count</i>	The condition is evaluated as <i>True</i> if the <i>Activity Attempt Count</i> for the activity is equal to or greater than the <i>Limit Condition Attempt Limit</i> for the activity
<i>Always</i>	<i>None</i>	The condition is always evaluated as <i>True</i>

3.4.3.2 Actions for Precondition Sequencing Rules

Table 3.8 shows a list of actions for precondition rules. These actions are applied when the next activity for delivery is decided as shown in Figure 3.1.

Table 3.8 List of Actions for Precondition Rules

Action	Description
<i>Skip</i>	This action is used for a situation where an activity for delivery is to be selected during a traversal of an activity tree with a <i>Continue</i> or <i>Previous</i> sequencing request. If the condition for an activity is met, that activity is skipped and the next in line activity in that direction is checked for delivery possibility.
<i>Disabled</i>	This action is used to prohibit the delivery of an activity. If the condition is <i>True</i> , that activity cannot be delivered even when selected.
<i>Hidden from Choice</i>	This action is used to hide an activity from the list of available activities for selection with a <i>Choice</i> sequencing request. If the condition is <i>True</i> , the activity is not included in the target for a <i>Choice</i> sequencing request.
<i>Stop Forward Traversal</i>	This action is used when the next activity is to be selected during a forward traversal of an activity tree. If the condition is evaluated as <i>True</i> , the activities following it will not be considered candidates for delivery.

3.4.4 Post-Condition Rules and Exit Action Rules

Post-condition rules and exit action rules can be used to ignore a navigation request from the learner and instead generate a sequencing request or termination request designed by the content developer.

These rules are defined for each activity. One or more rules may be defined for one activity. As with the precondition rules, these rules are defined in the form of

If [*condition set*] **Then** [*action*]

The condition set is a collection of conditions to be evaluated as *True* or *False* with respect to the tracking information for the activity. The action part indicates a sequencing request or a termination request.

Two examples of post-condition sequencing rules follow:

If Not Satisfied Then Retry

If the objective of the activity is not satisfied, then that activity is to be retried.

If All (Attempted, Satisfied) Then Exit All

If the activity has been attempted and the objective is satisfied, the whole learning process is to be terminated.

3.4.4.1 Condition Set of Post-Condition Sequencing Rules and Exit Action Rules

The condition part is defined in the same way as for precondition rules.

3.4.4.2 Actions for Post-Condition Sequencing Rules and Exit Action Rules

Table 3.9 shows a list of actions for post-condition rules and exit action rules. These actions are applied when a new sequencing request or a termination request is to be generated in place of the learner's navigation request during the termination and rollup process shown in Figure 3.1.

Table 3.9 Actions for Post-Condition Rules and Exit Action Rules

Action	Description	Sequencing Request	Termination Request
<i>Exit Parent</i>	Terminate the parent of the current activity		<i>Exit Parent</i>
<i>Exit All</i>	Terminate the whole learning process		<i>Exit All</i>
<i>Exit</i>	Terminate the current activity		<i>Exit</i>
<i>Retry</i>	Retry the current activity. If it is not a leaf activity, retry the first child activity of the cluster.	<i>Retry</i>	
<i>Retry All</i>	Terminate the whole learning process and resume	<i>Retry</i>	<i>Exit All</i>
<i>Continue</i>	Move forward	<i>Continue</i>	
<i>Previous</i>	Move backward	<i>Previous</i>	

3.4.5 Rollup Rules

In the Rollup process, the tracking information of each activity in an activity tree is successively rolled up from each leaf activity (SCO) towards the root activity. Rollup rules are used to decide how the tracking information of a parent activity is rolled up from the tracking information of its child activities.

Rollup rules are related to the *Objective Satisfied Status* data, the *Objective Normalized Measure* data, and the *Attempt Completion Status* data. Figure 3.3 shows the relationship between these three types of data.

In the measure rollup process, the *Objective Normalized Measure* data of the parent's primary objective is determined from the *Objective Normalized Measure* data for its child activities' primary objectives. Only one primary objective is associated with each activity as described in Section 3.1.

The satisfied status of the primary objective of a parent activity is determined through a rollup process using the following data: its own *Objective Normalized Measure* data that is rolled up from its child activities, the *Objective Satisfied Status* data of the

primary objective, the *Attempt Completion Status* data, and the *Activity Attempt Count* data of each child activity.

In the progress status rollup, the *Attempt Completion Status* data of a parent activity is determined from the *Objective Satisfied Status* data, the *Attempt Completion Status* data, and the *Activity Attempt Count* data of each child activity.

Each of these is explained below.

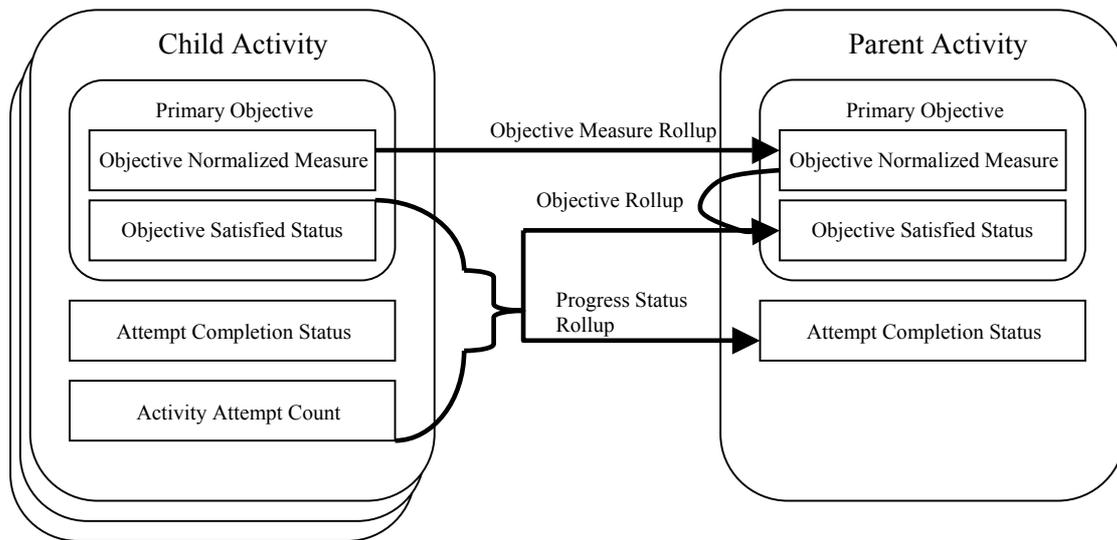


Figure 3.3 Tracking Information Relationship in a Rollup Process

3.4.5.1 Measure Rollup Process

The objective measure of a parent activity’s primary objective is determined by calculating the weighted average of the objective measure for the primary objective of each child activity. The weight for an objective measure is specified by the content developer using the *Rollup Objective Measure Weight* data item. The formula is as follows.

The Objective Measure of a Parent

$$= \frac{\sum_{\text{child activity}} (\text{RollupObjectiveMeasureWeight} \times \text{ObjectiveNormalizedMeasure})}{\sum_{\text{child activity}} \text{RollupObjectiveMeasureWeight}}$$

If the *Objective Normalized Measure* value of a child activity’s primary objective is *Unknown*, the calculation is performed as if the value is 0.

3.4.5.2 Objective Rollup

The objective rollup process determines the *Rollup Objective Satisfied* status of a parent activity through the following sequence:

(1) Using Objective Measure

If the *Objective Satisfied by Measure* element of the parent activity is evaluated as *True*, the *Objective Satisfied Status* element of the parent is determined by comparing its *Objective Normalized Measure* value that has been calculated through a measure rollup process against the value specified for the *Objective Minimum Satisfied Normalized Measure*. If the rollup measure equals or exceeds the value of the *Objective Minimum Satisfied Normalized Measure*, the *Rollup Objective Satisfied* status is *True*; otherwise, it is *False*. The rollup process ends here.

If the *Objective Satisfied by Measure* element of the parent activity is evaluated as *False*, the activity's status does not change and the rollup process proceeds to (2) below.

(2) Using Rollup Rules

If the rollup rules defined for the activity contain an action *Satisfied* or *Not Satisfied*, the satisfied status of the primary objective is determined by evaluating the *Not Satisfied* rule first and then the *Satisfied* rule. Therefore, the result of the *Not Satisfied* rule evaluation may sometimes be overwritten by the result of the *Satisfied* rule evaluation. The objective rollup process ends here. The rollup rules will be discussed in more detail later.

If the actions of the rollup rules for the parent activity do not include *Satisfied* or *Not Satisfied*, the rollup process proceeds to the evaluation of the default rules as shown in (3) below.

(3) Using Default Rules

The following default rules are evaluated in the same way as in (2) above.

If all (attempted or not satisfied), Then not satisfied
If all satisfied, Then satisfied

That is,

If all the activities are attempted or their *Objective Satisfied Status* values are not *True*, the *Rollup Objective Satisfied* status is *False*.

If the *Objective Satisfied Status* values of all child activities are *True*, then the *Rollup Objective Satisfied* status of the parent activity is *True*.

3.4.5.3 Activity Progress Rollup Process

In the progress rollup process, the Rollup Completion Status of a parent activity is evaluated through the following sequence.

(1) Using Rollup Rules

If the rollup rules defined for the activity contain an action *Completed* or *Incomplete*, the attempt completion status is determined by evaluating the *Incomplete* rule first and then the *Completed* rule. Therefore, the result of the *Incomplete* rule evaluation may sometimes be overwritten by the result of the *Completed* rule evaluation. The progress rollup process ends here. The rollup rules are discussed below in more detail.

If the actions of the rollup rules for the parent activity do not include *Completed* or *Incomplete*, the rollup process proceeds to the evaluation of the default rules as described in (2) below.

(2) Using Default Rules

The following default rules are evaluated in the same way as in (1) above.

If all (attempted or incomplete), Then incomplete
If all completed, Then completed

That is,

If the *Attempt Completed Status* values of all child activities are *True*, then the *Rollup Completion Status* is *True*.

If all the activities are attempted or their *Attempt Completion Status* values are *True*, the *Rollup Completion Status* is *True*.

3.4.5.4 Rollup Rules in Detail

The *Satisfied* and *Not Satisfied* rollup rules for an objective rollup and the *Completed* and *Incomplete* rollup rules for an activity progress rollup process are both defined in the following format:

If [*condition_set*] **For** [*child_activity_set*] **Then** [*action*]

The above rule format means that each rollup rule consists of a *child_activity_set*, which is a set of child activities to consider, a *condition_set*, which is a set of conditions that are to be evaluated against the tracking information of the included child activities, and an *action*, which is a corresponding action that sets the cluster's tracking status information if the final result of applying the set of conditions to the child activity set is evaluated as *True*.

Examples of the rollup rules are shown below.

If not satisfied For any Then not satisfied
 If any child activity is not satisfied, then the cluster is not satisfied.

If satisfied For at least 3 Then satisfied
 If at least three child activities are satisfied, then the cluster is satisfied.

If satisfied or completed For all Then completed
 If all child activities are satisfied or completed, then the cluster is completed.

If satisfied and attempted For all Then satisfied
 If all child activities are satisfied and attempted, then the cluster is satisfied.

If not attempted For at least 50% Then incomplete
 If at least 50% of the child activities are not attempted, then the cluster is incomplete.

► Rollup Condition Set

A condition set of a rollup rule is defined in the following format:

condition_combination ((condition_operator, condition_element),)

This is in the same format as for precondition sequencing rules (Section 3.4.3.) The two rule condition combination elements (All and Any) and the two rule condition operators (Not and NO-OP) are the same as those in precondition rules.

The rule condition elements of a rollup rule differ from those of a precondition or post-condition sequencing rule. Table 3.10 shows a list of condition elements used in rollup rules. The differences from those of a precondition or post-condition sequencing rule are that, first, there is no condition element for comparing the values of objective measures, and second, as the target objective is limited only to the primary objective value, no objective is specified.

► Rollup Child Activity Set

The *Rollup Child Activity Set* element is used in a rollup rule to define how to determine the final evaluation result of *True* or *False* from the application of the specified condition set to the child activities. For example, a rollup rule can be defined specifying that if 80% of the child activities satisfy the specified condition set, then the final result is set to *True*. Table 3.11 shows a list of rollup child activity elements.

Table 3.10 List of Rollup Rule Condition Elements

Condition	Tracking Information	Description
<i>Satisfied</i>	<i>Objective Satisfied Status</i>	The condition is evaluated as <i>True</i> if the <i>Objective Satisfied Status</i> for the primary objective is <i>True</i>
<i>Objective Status Known</i>	<i>Objective Satisfied Status</i>	The condition is evaluated as <i>True</i> unless the <i>Objective Satisfied Status</i> for the primary objective is <i>Unknown</i>
<i>Objective Measure Known</i>	<i>Objective Normalized Measure</i>	The condition is evaluated as <i>True</i> unless the <i>Objective Normalized Measure</i> for the primary objective is <i>Unknown</i>
<i>Completed</i>	<i>Attempt Completion Status</i>	The condition is evaluated as <i>True</i> if the <i>Attempt Completion Status</i> for the activity is <i>True</i>
<i>Activity Progress Known</i>	<i>Attempt Completion Status</i>	The condition is evaluated as <i>True</i> unless the <i>Attempt Completion Status</i> for the activity is <i>Unknown</i>
<i>Attempted</i>	<i>Activity Attempt Count</i>	The condition is evaluated as <i>True</i> if the <i>Activity Attempt Count</i> for the activity is 1 or more

Condition	Tracking Information	Description
<i>Attempt Limit Exceeded</i>	<i>Activity Attempt Count</i>	The condition is evaluated as <i>True</i> if the <i>Activity Attempt Count</i> for the activity is equal to or greater than the <i>Limit Condition Attempt Limit</i> for the activity
<i>Never</i>	<i>None</i>	The condition is always evaluated as <i>False</i>

Table 3.11 List of Child Activity Sets

Name	Description
<i>All</i>	If the condition combination of all child activities is evaluated as <i>True</i> , then the specified rollup action is applied
<i>Any</i>	If the condition combination of any child activity is evaluated as <i>True</i> , then the specified rollup action is applied
<i>None</i>	If none of the child activities contains a condition combination that is evaluated as <i>True</i> , then the specified rollup action is applied
<i>At Least Count</i>	If the number of activities that contain a condition combination evaluated as <i>True</i> is at least equal to the number specified by the <i>Rollup Minimum Count</i> element, then the specified rollup action is applied
<i>At Least Percent</i>	If the percentage of activities containing a condition combination evaluated as <i>True</i> is at least equal to the number specified by the <i>Rollup Minimum Percent</i> element, then the specified rollup action is applied

How to specify a child activity set which is to be subject to a rollup rule is discussed here. In general, the tracking status data for all child activities is used in the rollup evaluations of the parent activity. However, a content developer can selectively include or exclude certain activities from the rollup child activity set for rollup evaluations. For example, when the *At Least Count* element is used for a rollup activity child set definition in a rollup rule, the number of child activities containing a condition combination evaluated as *True* can be calculated by excluding child activities from the target rollup child activity set under the following circumstances:

- A child activity whose *Tracked* element is defined as *False*, which means that no tracking status data is maintained for the activity, is never included during rollup.
- A child activity whose *Rollup Objective Satisfied* element is defined as *False* is not included in the evaluation of rollup rules having a *Satisfied* or *Not Satisfied* rollup action.
- A child activity whose *Rollup Progress Completion* element is defined as *False* is not included in the evaluation of rollup rules having a *Completed* or *Incomplete* rollup action.
- A child activity that has various *Required for* rollup elements defined, which indicate, conditionally, when an activity is included in the evaluation of rollup

rules having specified rollup actions, might not be included. *Required for* rollup elements are shown in Table 3.12.

Table 3.12 List of Required for Rollup Elements

Name	Description	Vocabulary (common)
<i>Required for Satisfied</i>	Indicates when the action is considered in the <i>Satisfied</i> rollup rule	<ul style="list-style-type: none"> • <i>always</i> – To be always considered. • <i>ifNotSuspended</i> – To be considered when the child activity has been attempted but is not suspended at the time of evaluation • <i>ifAttempted</i> – To be considered when the child activity has been attempted at the time of evaluation • <i>ifNotSkipped</i> – To be considered if the child activity has not been skipped at the time of evaluation
<i>Required for Not Satisfied</i>	Indicates when the action is considered in the <i>Not Satisfied</i> rollup rule	
<i>Required for Completed</i>	Indicates when the action is considered in the <i>Completed</i> rollup rule	
<i>Required for Incomplete</i>	Indicates when the action is considered in the <i>Incomplete</i> rollup rule	

► Rollup Rule Actions

The *Rollup Rule Action* element specifies one of four actions, *Satisfied*, *Not Satisfied*, *Completed*, or *Incomplete*, that will be applied to the parent activity to which rollup rule is associated. Rollup Rule Actions are shown in Table 3.13.

Table 3.13 List of Rollup Actions

Rollup action	Description
<i>Satisfied</i>	The <i>Objective Satisfied Status</i> of the rollup objective for the associated parent activity is set to <i>True</i>
<i>Not Satisfied</i>	The <i>Objective Satisfied Status</i> of the rollup objective for the associated parent activity is set to <i>False</i>
<i>Completed</i>	The <i>Attempt Completion Status</i> of the rollup objective for the associated parent activity is set to <i>True</i>
<i>Incomplete</i>	The <i>Attempt Completion Status</i> of the rollup objective for the associated parent activity is set to <i>False</i>

3.4.6 Local Objectives and Shared Global Objectives

Each activity is associated with one local objective by default. Moreover, a course developer can specify any number of local objectives for a given activity as required.

These local objectives can be associated with shared global objectives. The shared global objectives enable activities to share tracking information for sequencing. The introduction of global objectives makes it possible, for example, for the content developer to define a situation where a learner is led through a tutorial activity or can skip it depending on the result of a pretest activity. Figure 3.4 shows the relationships between activities, local objectives and shared global objectives.

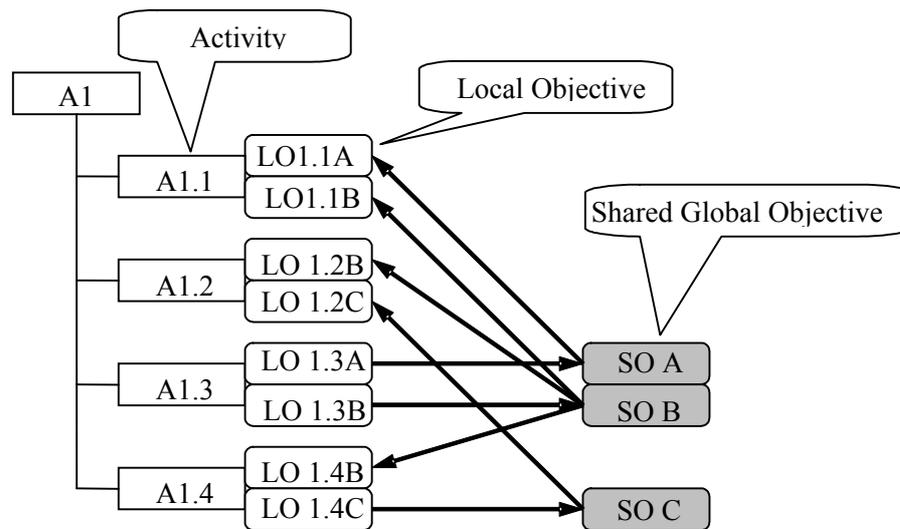


Figure 3.4 Relationships between Activities, Local Objectives and Shared Global Objectives

There are limitations, though, on how activities are related to local and shared global objectives:

- An activity may have more than one local objective. For example, refer to the relationship of A1.1 with LO1.1A and LO1.1B in Figure 3.4.
- One local objective can be related to only one shared global objective. For example, refer to the relationship between LO1.1A and SO A.
- One shared global objective can be related to more than one local objective. For example, refer to the relationship of SO B with LO1.1B, LO1.2B, and LO1.4B.
- The above means that one activity can be related to shared global objectives through the activity's associated local objectives. For example, refer to the relationship of A1.1 with SO A and SO B.
- From the perspective of a shared global objective, one shared global objective can be related to multiple activities through its connection with the local objectives of those activities. For example, refer to the relationship of SO B with A1.1, A1.2, and A1.4.

When a local objective is connected to a shared global objective, the direction for transmission of the objective measure and satisfaction status data should be defined. That is, whether the local objective data is to be written to the shared global objective data or the shared global objective data is to be read to a connected local objective should be defined. The following restriction is applied in this situation:

- For a given activity and a shared global objective, the information from only one local objective can be written to that shared global objective, and the shared global objective data cannot reflect information from any other local objective. For example, refer to the relationship of SO B with LO1.1B, LO1.2B, and LO1.4B in Figure 3.4.

3.4.7 Shared Global Objectives and Rule Evaluations

3.4.7.1 Shared Global Objectives with Precondition, Post-Condition and Exit Rules

As discussed in 3.4.3 Precondition Sequencing Rules and 3.4.4. Post-Condition Rules and Exit Action Rules, the condition part of these rules can refer to the local objectives associated with the activity. If the connection between a local objective of the activity and a shared global objective is defined in such a way that the information from the shared global objective is to be read into the local objective, the information from the shared global objective is used in the rule evaluation.

3.4.7.2 Shared Global Objectives and Rollup Rules

As described in Section 3.4.5 Rollup Rules, only the primary objective of an activity is used for rollup. If the primary objective of an activity is connected to a shared global activity, the shared global objective information is affected by the rollup rules.

Figure 3.5 shows the relationship between rollup and shared global objectives. Rollup rules are evaluated on the basis of each leaf activity whose tracking information has changed because of a state transition on the associated SCO and the activities which refer to the shared global objective whose value is written from the leaf activity. The collection of activities that is the base for rollup is called a rollup set. If there is a change to the tracking information of Activity A1.1 in Figure 3.5, for example, the rollup set includes three activities: A1.1.2 and A.1.2, which read the value reflecting the state change from the shared global objective SO B, and A.1.1.1.

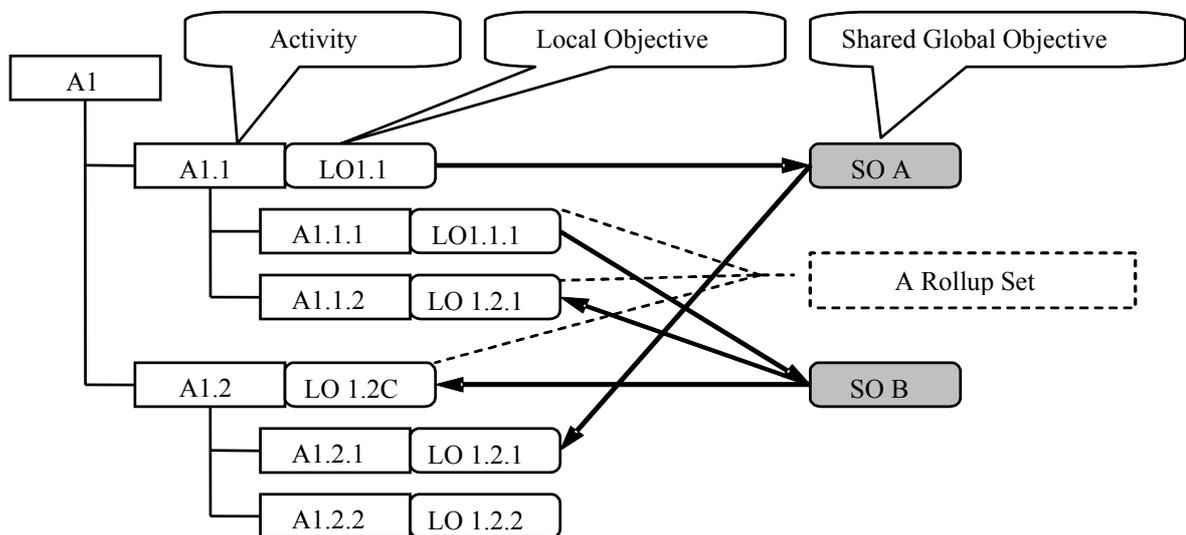


Figure 3.5 Relationships between Shared Global Objectives and Rollup

If a rollup process originating from an activity in the rollup set reaches another activity in a rollup set, the second activity is removed from the rollup set. Furthermore, if the information regarding a shared global objective is updated during a rollup process to reflect a change in the state of an associated local objective of an activity,

no separate rollup behavior occurs among the other activities that refer to that shared global objective. In Figure 3.5, for example, although the tracking information of A1.1 is affected by the rollup behavior from A.1.1.1 and the shared global objective SO A reflects this change, the change is not propagated to A1.2.1.

3.5 Attempts

An attempt refers to a learning effort from the point at which an activity is delivered to the learner and the learner starts the activity to the point at which the learner completes the activity and another activity is selected for delivery. If an activity has been completed during an attempt and that activity is selected and delivered again, this is regarded as a separate attempt.

Attempts are managed in terms of parent and child relationships in an activity tree. Therefore, if A1.1.2 in Figure 3.6 is being currently attempted, for example, A1.1.2, A1.1, and A1 are all regarded as being attempted. If a learner has selected A1.1.1 after having completed A1.1.2 in this case, A1.1.2 is regarded as completed but A1.1 and A1 are still regarded as being attempted.

An attempt can be suspended by a Suspend All navigation request. An attempt on an activity can be resumed from the suspended state through a Resume All navigation request. A resumed attempt is not regarded as a new attempt, but as a continuation of the suspended attempt.

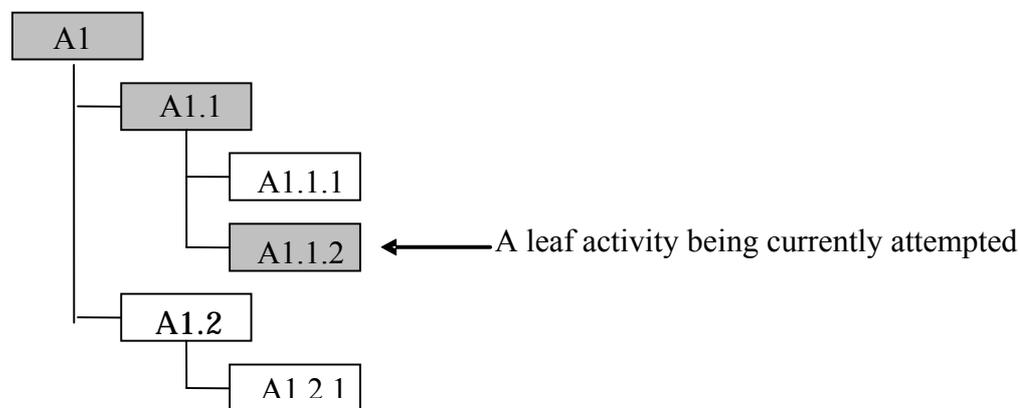


Figure 3.6 Attempts

4 Navigation

This section explains the navigation feature that has been added to SCORM 2004. It covers the basic navigation concepts from a broad perspective, implications of the navigation feature for user interface implementation, and how navigation behavior can be specified.

4.1 Navigation Control Overview

4.1.1 SCO Navigation in SCORM 1.2

The specifications of SCORM 1.2 stipulate that all navigation controls between SCOs should be provided by the LMS. For example, the user interface controls required for the presentation of an SCO and the transition from that SCO to another SCO must be provided by the LMS under SCORM 1.2. In other words, content developers had no say on the navigation behavior between SCOs, and thus could not provide navigation buttons from one SCO to another.

In addition, SCORM 1.2 did not define how an LMS should manage the navigation behavior of SCOs. This made it difficult to provide consistent user interfaces since the user interfaces differed, depending on each LMS, with respect to the presence or absence of buttons and menus and their display positions on the screen, as well as in their captioning and navigation methods. That is, these limitations on navigation control design hindered content developers creating content for multiple LMS environments under SCORM 1.2 who wanted to design and provide user interfaces with a consistent look and feel.

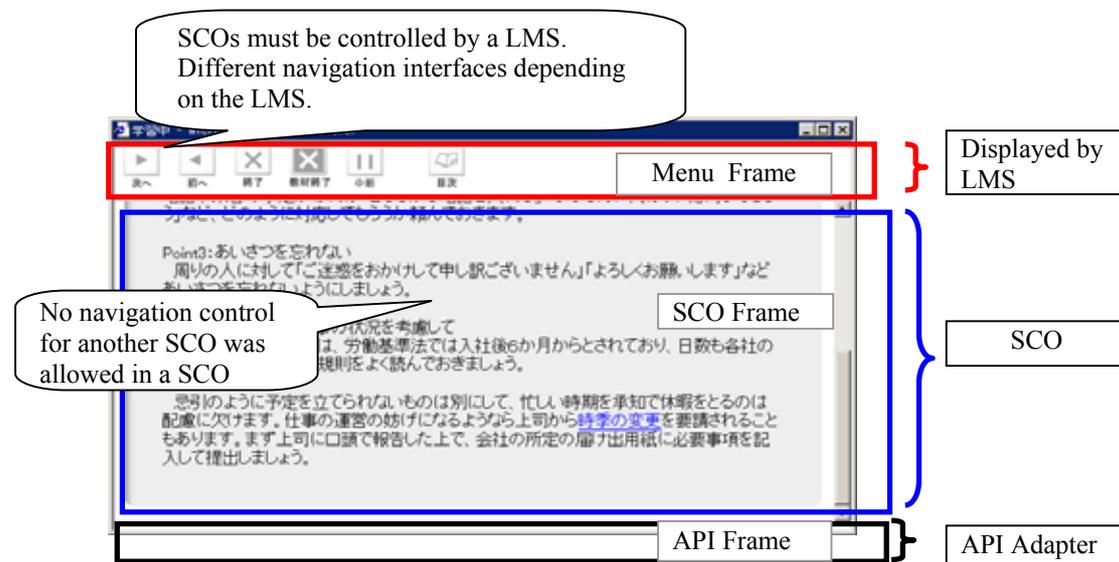


Figure 4.1 An Example of SCOM 1.2 Navigation Implementation

4.1.2 SCO Navigation in SCORM 2004

In SCORM 2004, a set of new specifications has been added for managing the navigation methods of SCOs and content developers can now control the navigation of SCOs. More specifically, SCOs can now issue SCO navigation requests and request that the LMS display or hide navigation buttons.

This new functionality enables content developers to standardize a consistent navigation design, considered an important goal in content development, without taking into account the type of LMS environment in which the content will be used.

Note that the LMS does not interfere with the navigation control within an SCO (or an asset) under SCORM 2004 or SCORM 1.2, and the content developer must consider all aspects of this.

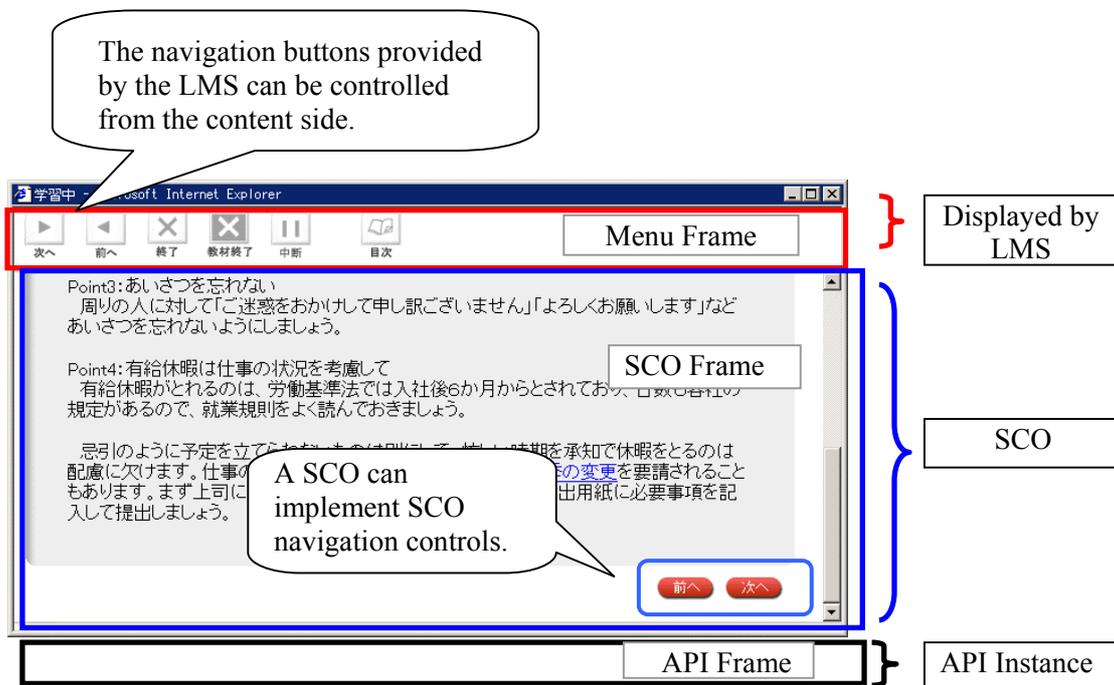


Figure 4.2 An Example of SCORM 2004 Navigation Implementation

4.2 Triggering a Navigation Event and SCO Termination

4.2.1 SCO Navigation Event Triggered by an SCO

In SCORM 2004, a set of navigation events like Continue and Previous can be triggered from an SCO. These events are now available in addition to the SCO navigation requests issued by an LMS that could be used in previous SCORM versions. The navigation events triggered by an SCO are processed in the same way as those triggered by an LMS.

The navigation events that can be triggered from an SCO are listed in Table 4.1.

Table 4.1 Navigation Events that can be Triggered from an SCO

Navigation event	Behavior description
<i>continue</i>	This event leads to the termination of the current SCO and the issuing of a <i>continue</i> navigation request
<i>previous</i>	This event leads to the termination of the current SCO and the issuing of a <i>previous</i> navigation request
<i>choice</i>	This event leads to the termination of the current SCO and the issuing of a <i>choice</i> navigation request
<i>exit</i>	This event leads to the ending of the current attempt on the current activity, and to the issuing of an <i>exit</i> navigation request
<i>exitAll</i>	This event leads to the ending of the current attempt on the current activity tree and all the associated activities, and to the issuing of an <i>exit all</i> navigation request
<i>abandon</i>	This event leads to the ending of the current attempt on the current activity, and to the issuing of an <i>abandon</i> navigation request
<i>abandonAll</i>	This event leads to the ending of the current attempt on the current activity tree and all the associated activities, and to the issuing of an <i>abandon all</i> navigation request
<i>_none_</i>	This event leads to the clearing of all navigation requests that are not yet processed

The navigation requests that are generated by navigation events triggered by an SCO are considered valid for processing in conjunction with sequencing control modes in the same way as SCO navigation requests issued by an LMS. A sequencing control mode specified on a cluster defines the navigation requests that can be applied to the cluster's child activities.

If the *Sequencing Control Choice* is valid, for example, a *choice* navigation request is applied to the child activities of the cluster. In the same way, if the *Sequencing Control Flow* is valid, a *continue* or *previous* navigation request can be applied to the child activities of the cluster.

4.2.2 Navigation Request Event and SCO Termination

To allow an SCO to trigger a navigation event, an element called *adl.nav.request* has been introduced in the navigation data model of SCORM 2004. An SCO triggers a navigation event in the runtime environment or in the LMS by calling a *SetValue* function with a value like *continue*, *previous*, *choice* or *exit* for the *adl.nav.request* navigation data model element in the following format:

```
SetValue("adl.nav.request", <REQUEST>)
    where <REQUEST> is one of the following: continue, previous,
    choice, exit, exitAll, abandon, or abandonAll.
```

Note that to trigger a **choice** navigation event, it is necessary to specify the identifier of an activity to be delivered. The format is as follows:

```
SetValue("adl.nav.request", "{target =<STRING>}choice")
    where <STRING> indicates the item identifier of the target activity.
```

When an SCO has communicated a navigation request using an API method call to the LMS with a navigation request event, the LMS performs a sequencing process based on the navigation request after having accepted the termination processing of the SCO. Even if a navigation request has been communicated to the LMS, the LMS does not immediately respond to it, but starts processing the request only when it has accepted a termination request from the SCO.

An SCO may repeatedly invoke the function for a navigation request event many times, but each time it invokes that function the value set by the SCO is replaced with the new value provided in the latest function call. This means that the only event to be responded to is the last navigation request event that is recorded after the LMS has completed processing a terminate request from the SCO.



Figure 4.3 An Example Use of a “continue” Navigation Event

4.2.3 Validity of Navigation Request Events

To enable SCOs to check with the LMS as to whether a navigation request *continue*, *previous*, or *choice* is valid, a new element (*adl.nav.request_valid.REQUEST*) has been added as a navigation data model element. When this query is made by an SCO, the LMS returns a value indicating whether the request is valid. The query result is *True* if the request is valid, *False* if it is not valid, and *Unknown* if the validity is not known.

```
GetValue("adl.nav.request_valid.<REQUEST>")
  where <REQUEST> is continue, previous or choice.
Return value: true, false or unknown
```

To check whether a choice navigation request is valid, the identifier of the target activity must be specified as shown below.

```
GetValue("adl.nav.request_valid.choice {target=<STRING>}")
  where <STRING> indicates the item identifier of the target activity.
```

In addition, an SCO can confirm the value that is currently set to the `adl.nav.request` element through the `GetValue` method. The value returned from a `GetValue` method call is the value currently stored at the data element on the LMS side. If there is no value set to the element, `_none_`, which is the initial value of `adl.nav.request`, is returned.

```
GetValue("adl.nav.request", <REQUEST>)
    where <REQUEST> is one of the following: continue, previous,
    choice, exit, exitAll, abandon, or abandonAll.
```

4.3 Controlling LMS-Provided Navigation Devices

SCORM 2004 allows a content developer to specify whether the user interface devices of the LMS are to be hidden or shown. By including navigation request events in SCOs and specifying the user interface device control on the LMS, content developers can introduce their own design policies to their content user interface and content organization.

It is possible to specify whether to show or hide the LMS-provided user interface devices corresponding to the *continue*, *previous*, *exit* and *abandon* navigation requests by defining an appropriate token as the value of the `hideLMSUI` element for each activity in the manifest file (`imsmanifest.xml`). Table 4.2 shows a list of vocabulary tokens that can be used with the `hideLMSUI` element.

Table 4.2 Tokens for Controlling LMS-provided Navigation Devices

Token	Description
<i>previous</i>	Hide the <i>Previous</i> navigation device
<i>continue</i>	Hide the <i>Continue</i> navigation device
<i>exit</i>	Hide the <i>Exit</i> navigation device
<i>abandon</i>	Hide the <i>Abandon</i> navigation device

The example below specifies a situation where the *Continue* and *Previous* navigation user interface devices provided by the LMS will not be displayed at the run-time for Activity “item1”.

```
<organization>
  <item identifier="item1" identifierref="Resource1" isvisible="true">
    <adlnav:presentation>
      <adlnav:navigationInterface>
        <adlnav:hideLMSUI>continue</adlnav:hideLMSUI>
        <adlnav:hideLMSUI>previous</adlnav:hideLMSUI>
      </adlnav:navigationInterface>
    </adlnav:presentation>
  </item>
</organization>
```

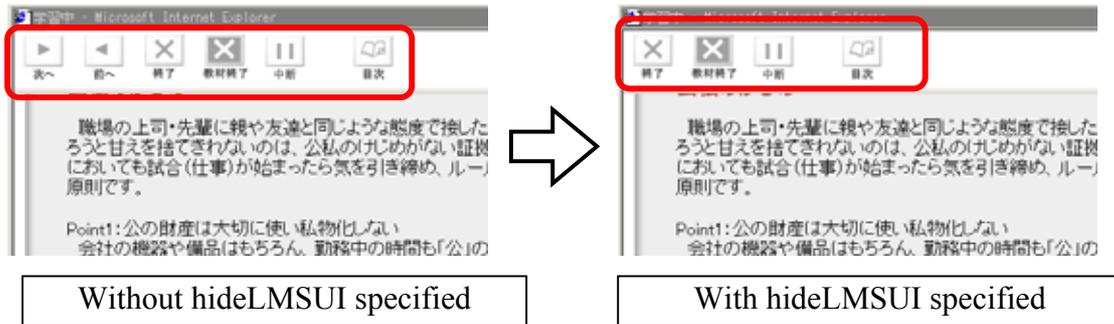


Figure 4.4 Displaying and Hiding LMS-provided Navigation Devices

5 Run-Time Environment (RTE)

The section describes the SCORM Run-Time Environment that lies between learning resources like SCOs and the LMS. It covers the changes made to the Run-Time Environment of SCORM 2004 from that of SCORM 1.2.

5.1 SCORM 2004 Run-Time Environment Overview

The SCORM Run-Time Environment (RTE) book describes a common launching content object mechanism of learning resources, a common communication mechanism between learning resources and the LMS, and a common data model for handling the tracking information for managing each learner's performance and progress with the learning resources. In a Run-Time Environment, an SCO that has been delivered with the use of an Application Programming Interface (API) instance communicates with the LMS (Figure 5.1).

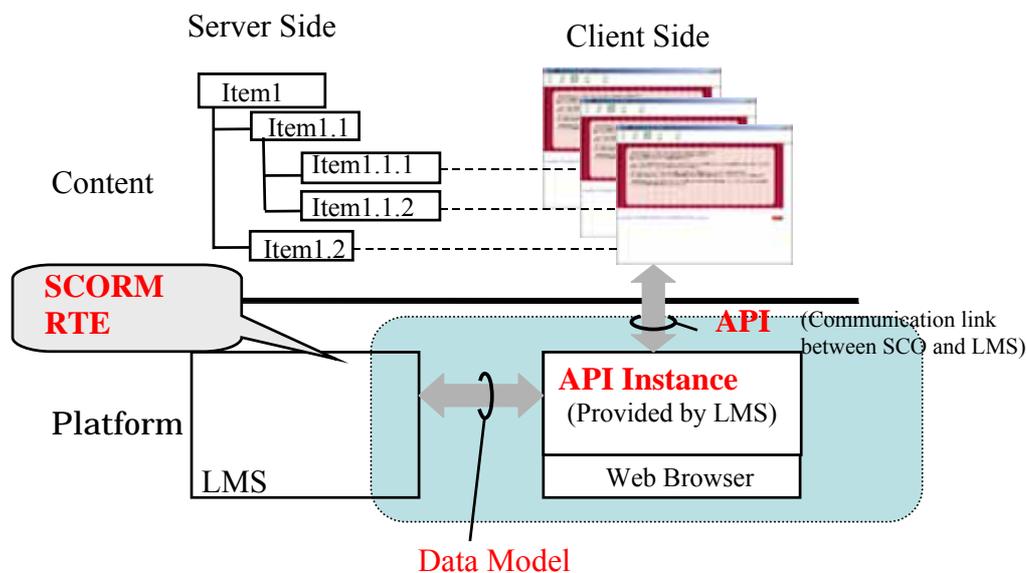


Figure 5.1 SCORM 2004 Run-Time Environment

The launching process defines common methods used by the LMS to launch Web-based learning resources. This mechanism specifies the methods and responsibilities for establishing a communication session between the LMS and a Content Object that has been delivered. The communication mechanism is standardized with a common API.

The API is a communication mechanism for sending status information regarding the Content Object (for example, regarding the initialization, termination, and error status) and is used for setting or getting data between the launched SCO and the LMS. The API Instance in Figure 5.1 indicates a software component that is provided by the LMS for a launched SCO to locate and use for communication with the LMS in an ECMAScript compatible language (e.g., JavaScript).

The data model is a standard set of data elements defined for storing and referencing the necessary information to be tracked such as the completion status of an SCO and the score from a quiz or test assessment. The LMS and SCOs are implemented in such a way that the LMS and the SCOs communicate with each other using the data model elements under the assumption that the other party knows the meanings and uses of these data model elements.

5.2 Launching Content Objects

The LMS is responsible for determining a learning activity to be attempted based on a navigation request, and then delivering an associated learning resource for the learner. When delivering a learning resource, the LMS launches the target resource using the URL that has been defined as the launch location for the learning resource. A launch method may be implemented either on the client side or the server side, but the learning resource whose launch location is defined using the HTTP protocol is displayed on the Web browser window of the client.

There are two types of learning resource that can be launched by the LMS: SCOs and assets.

5.2.1 Assets

An asset refers to a resource consisting of digital media such as text and images that can be loaded through a Web browser. An asset does not communicate with the LMS. An asset does not need to include a function called to the API that is provided by the LMS.

5.2.2 SCOs

The standard specifies that as a collection of one or more assets, a Sharable Content Object (SCO) should communicate with the LMS using the Run-Time Environment. An SCO is the minimum resource unit whose behavior can be recorded and managed by the LMS.

It is also specified in SCORM that only one SCO can be launched by the LMS at a given time and one SCO can be activated.

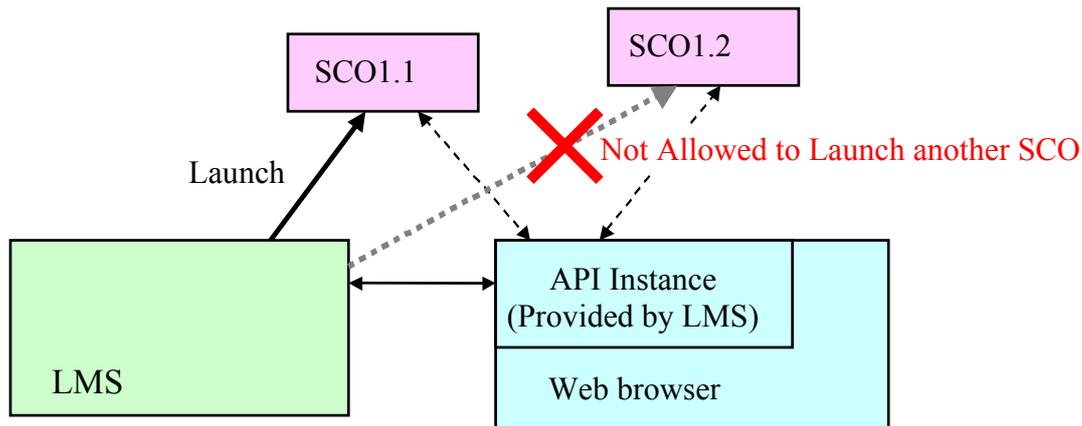


Figure 5.2 Launching a SCO

5.3 API

5.3.1 API Overview

Prior to its current version (SCORM 2004), SCORM was based on the AICC *CM1001 Guidelines for Interoperability*. The AICC has submitted its work to the IEEE Learning Technology Standards Committee (LTSC) as a candidate to become an international standard. SCORM has adopted the *IEEE 1484.11.2-2003 Standard for Learning Technology – ECMAScript Application Programming Interface for Content* as the standard for Runtime Services (RTS) communication. Adopting a common API made it possible for SCORM to meet its high-level requirements concerning interoperability and usability. The API provides a standardized way for SCOs to communicate with LMSs, and also enables LMS implementers to encapsulate their own specific implementation details from content developers. The LMS has to provide the API, API Implementation and API Instance that SCOs need to communicate.

5.3.2 API Instance Overview

An API Instance³ is a piece of software that implements and exposes the necessary API methods, and it is provided to an SCO by the LMS as an interface through which the SCO can communicate with the LMS. Content developers must develop their contents (SCOs) in such a way that they can locate this API Instance provided by the LMS.

An important aspect of the API is that it allows an SCO to communicate with the LMS. Once an SCO has been launched, the SCO can store (*SetValue*) or refer to (*GetValue*) the data that the LMS is keeping for the SCO. Communication between the API Instance and an SCO is realized by invoking the methods of the API Instance.

The name of the API Instance is “API_1484_11”⁴ in SCORM 2004.

³ The API Instance was called the API adapter in SCORM versions prior to 1.2.

⁴ The API Instance name has been changed from “API” to “API_1484_11” in SCORM 2004.

5.3.3 Using the API Instance

To establish a communication session with the LMS, a launched SCO must find the instance of the API implementation object that has been provided to it by the LMS. This means that the SCO must recursively search the parent windows and opener window for the API Instance. In this case, the LMS must make the API Instance available in the Document Object Model (DOM) context as an object called “API_1481_11”.

5.3.3.1 LMS Responsibilities

The LMS must provide an API Instance under the following conditions:

- The LMS must make it possible for SCOs to access the API Instance in the DOM context with the name of “API_1481_11”.
- The LMS must enable the SCO to access the API Instance using ECMAScript (JavaScript) code.
- The LMS must launch an SCO in a child window of the window where the API Instance has been loaded or in a child frame of the LMS window.

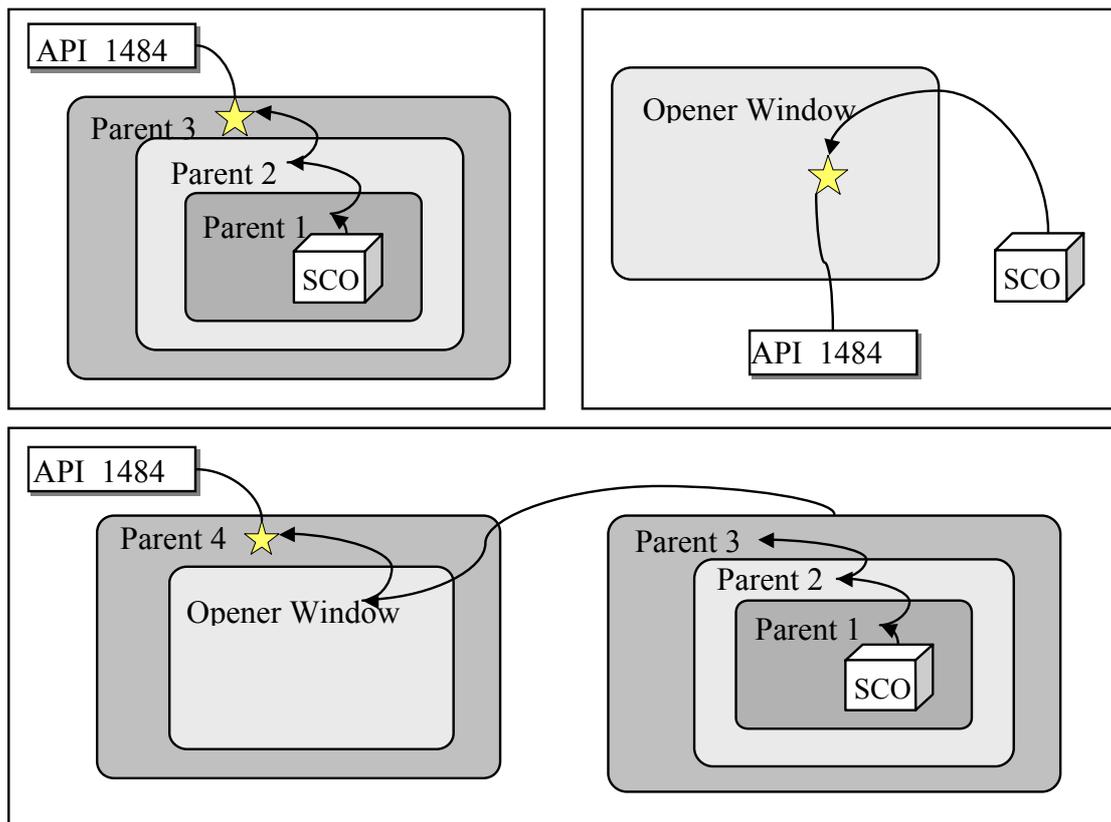


Figure 5.3 Permitted Location of an API Instance

5.3.3.2 SCO Responsibilities

SCOs must be developed in such a way that they can establish a communication session with the LMS by searching for the API Instance. For an SCO to find the API Instance located in a DOM window, it must search in the following sequence:

1. Search the chain of parents of the current window until the top window of the parent chain is reached.
2. Search the opener window (`window.opener`), which was the window opened by the SCO.
3. Search the chain of parents of the opener window, if any exist, until the top window of the parent chain is reached.

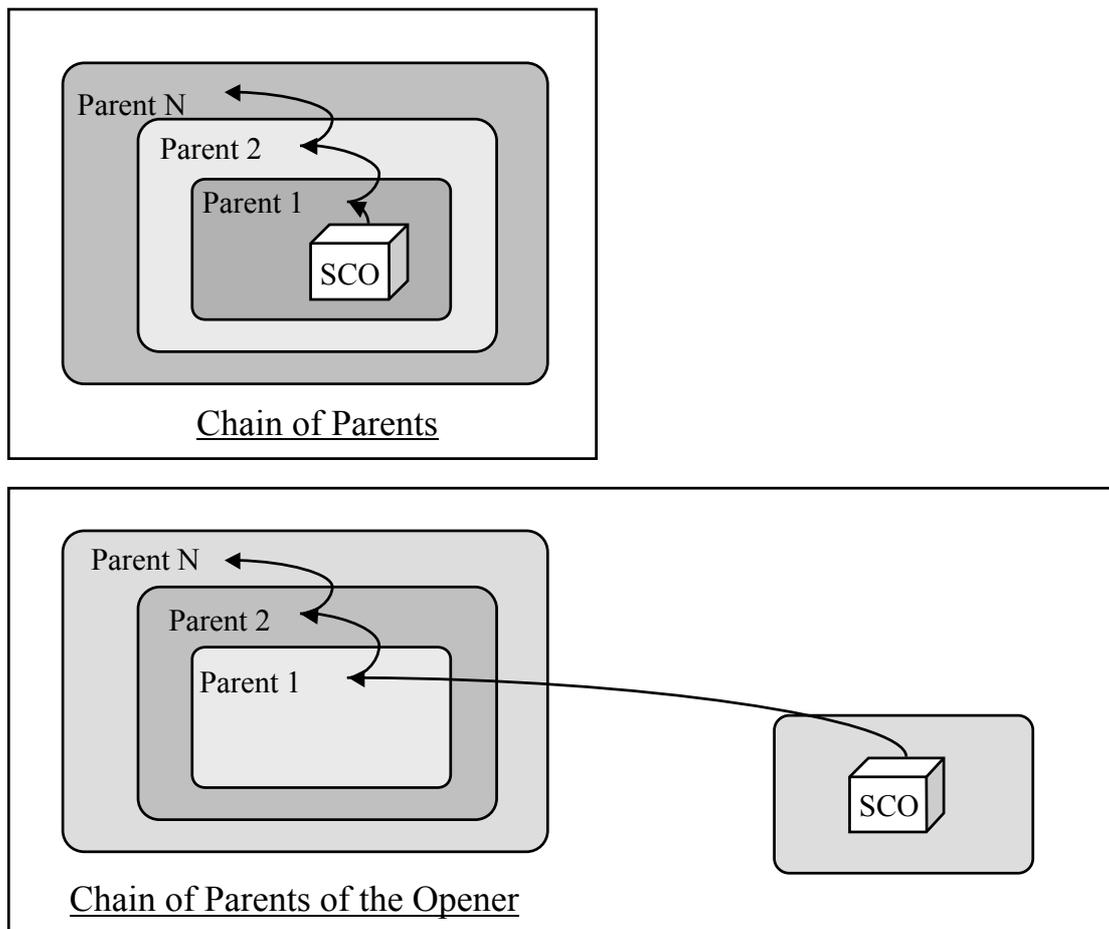


Figure 5.4 Finding the API Instance

The minimum amount of communication that an SCO must have with the LMS after locating the API Instance is the invocation of the *Initialize*("") and *Terminate*("") functions. The IEEE standard has provided a simple piece of ECMAScript that can be used to find the API Instance in a consistent manner. However, the standard does not require use of this ECMAScript code. Other means can be adopted.

5.3.4 API Method Overview

The API methods are classified into three categories as shown below.

Table 5.1 Categories of API Methods

Category	Description	API methods
Session Methods	Session methods are used to mark the beginning and end of a communication session between an SCO and an LMS through the API Instance.	<i>Initialize</i> <i>Terminate</i>
Data-transfer Methods	Data-transfer methods are used to exchange data model element values between an SCO and an LMS through the API Instance.	<i>GetValue</i> <i>SetValue</i> <i>Commit</i>
Support Methods	Support methods are used for auxiliary communications (e.g., error handling) between an SCO and an LMS through the API Instance.	<i>GetLastError</i> <i>GetErrorString</i> <i>GetDiagnostic</i>

In SCORM 2004, the names of the API methods provided by the LMS have been changed. (As the LMS prefix has been dropped from the names, it has become easier to recognize the names.)

Table 5.2 Changes to API Method Names

SCORM 1.2	SCORM 2004
LMSInitialize	Initialize
LMSFinish	Terminate
LMSGetValue	GetValue
LMSSetValue	SetValue
LMSCommit	Commit
LMSGetLastError	GetLastError
LMSGetErrorString	GetErrorString
LMSGetErrorDiagnostic	GetErrorDiagnostic

The syntactic details of the API methods are shown in Table 5.3.

Table 5.3 List of API Methods

Session Methods	
<i>Initialize</i>	<p><u>Syntax</u>: <i>Initialize</i> (parameter)</p> <p><u>Description</u>: This method is used to initialize a communication session.</p> <p><u>Parameter</u>: (“”) an empty character string</p> <p><u>Return value</u>: A character string indicating a Boolean value (true or false).</p> <p>”true”: Indicates that the initialization on the LMS side was successful.</p> <p>”false”: Indicates that the initialization on the LMS side was not successful. In this case, the API Instance sets an error code. Support methods are used to interpret the error data.</p>
<i>Terminate</i>	<p><u>Syntax</u>: <i>Terminate</i> (parameter)</p> <p><u>Description</u>: This method is used to terminate the communication session. The termination process should also transmit the data that the SCO has set with the API Instance but has not yet been stored in the LMS. Once this method has been executed, it is not possible to call any support methods.</p> <p><u>Parameter</u>: (“”) an empty character string</p> <p><u>Return value</u>: A character string indicating a Boolean value (true or false).</p> <p>”true”: Indicates that the termination on the LMS side was successful.</p> <p>”false”: Indicates that the termination on the LMS side was not successful. In this case, the API Instance sets an error code. Support methods are used to interpret the error data.</p>

Data Transfer Methods	
<i>GetValue</i>	<p><u>Syntax</u>: <i>GetValue</i>(parameter)</p> <p><u>Description</u>: This method is used to request information from the LMS. The SCO can request the following information from the LMS:</p> <ul style="list-style-type: none"> • The values of the data model elements supported by the LMS • The version of the data model supported by the LMS • Whether specific data model elements are supported <p><u>Parameter</u>: The parameter indicates the identification of the target data model element.</p> <p><u>Return value</u>: One of two types of character string:</p> <ul style="list-style-type: none"> • A character string representing the value of the data model element indicated by the parameter • An empty character string (“”) when an error occurs. In this case, the API Instance sets an error code. Support methods are used to interpret the error data.
<i>SetValue</i>	<p><u>Syntax</u>: <i>GetValue</i>(parameter_1, parameter_2)</p> <p><u>Description</u>: This method is used to set the value indicated by parameter_2 as the value of the data element indicated by parameter_1 at the LMS. The data may be instantly transmitted to the LMS or sent in a batch after being cached for a while, depending on the design as follows:</p> <ul style="list-style-type: none"> • The values of the data model elements supported by the LMS • The version of the data model supported by the LMS • Whether specific data model elements are supported <p><u>Parameter</u>: parameter_1 indicates the name of the target data element parameter_2 indicates the value to be stored (a character string)</p> <p><u>Return value</u>: A character string indicating a Boolean value (true or false).</p> <p>”true” : Indicates that the data transfer to the LMS side was successful.</p> <p>”false” : Indicates that the data transfer to the LMS side was not successful. In this case, the API Instance sets an error code. Support methods are used to interpret the error data.</p>

Data Transfer Methods	
<i>Commit</i>	<p><u>Syntax</u>: <i>Commit</i>(parameter)</p> <p><u>Description</u>: This method is used to commit stored data from the SCO to the LMS. If there is any data from the SCO that has been cached by the API Instance since the last call to <i>Initialize</i>("") or <i>Commit</i>(""), whichever occurred most recently. If the commit is successful, the LMS sets the error code to "0" (no error encountered) and returns "true". If the API Instance does not cache any data, this method is processed in the same way as above.</p> <p><u>Parameter</u>: ("") an empty character string.</p> <p><u>Return value</u>: A character string indicating a Boolean value (true or false).</p> <p>"true": Indicates that the commit process on the LMS side was successful.</p> <p>"false": Indicates that the commit process on the LMS side was not successful. In this case, the API Instance sets an error code. Support methods are used to interpret the error data.</p>

Support Methods	
<i>GetLastError</i>	<p><u>Syntax</u>: <i>GetLastError</i>()</p> <p><u>Description</u>: This method is used to request the error code for the latest error state of the API Instance. The API Instance does not alter the state of the current error upon a call of this method by the SCO, and simply returns the error code.</p> <p><u>Parameter</u>: No parameter is specified.</p> <p><u>Return value</u>: A character string indicating the error code for the current error state.</p>

<i>GetErrorString</i>	<p><u>Syntax:</u> <i>GetErrorString</i>(parameter)</p> <p><u>Description:</u> This method is used to request a textual description of the current error state. The API Instance should guarantee the support for the error code implemented at the API. The API Instance does not alter the state of the current error upon a call of this method by the SCO, and simply returns a character string which is the error description.</p> <p><u>Parameter:</u> A character string indicating a target error code.</p> <p><u>Return value:</u> A character string representing the error message corresponding to the error code indicated by the parameter.</p> <ul style="list-style-type: none"> • The maximum length of a return value character string is 255 characters. • While a set of error codes is explicitly specified by SCORM, the description of each error code is specific to the LMS. • When the LMS cannot identify the error code, an empty character string (“”) is returned.
<i>GetDiagnostic</i>	<p><u>Syntax:</u> <i>GetDiagnostic</i>(parameter)</p> <p><u>Description:</u> This method is provided for specific use of the LMS. It allows the LMS to define additional diagnostic information through the API Instance.</p> <p><u>Parameter:</u> An implementer-specific value for diagnostics. The maximum length of the parameter value is 255 characters. An error code may be used as the parameter, but the parameter can also be used for other purposes.</p> <p><u>Return value:</u> A character string representing the diagnostic information that is implemented by the LMS. The maximum length of a return value character string is 255 characters.</p> <p>Note: If the <i>GetDiagnostic</i>() function is called with an empty character string (“”) as the parameter, it is recommended that the function should return a character string representing diagnostic information about the last error encountered.</p>

5.3.5 API Instance State Transitions

A conceptual model has been defined to specify the transitions of the API Instance during execution. The states of the API Instance indicate the transitions of the API Instance for specific events. These states are defined as

- Not Initialized
- Running
- Terminated

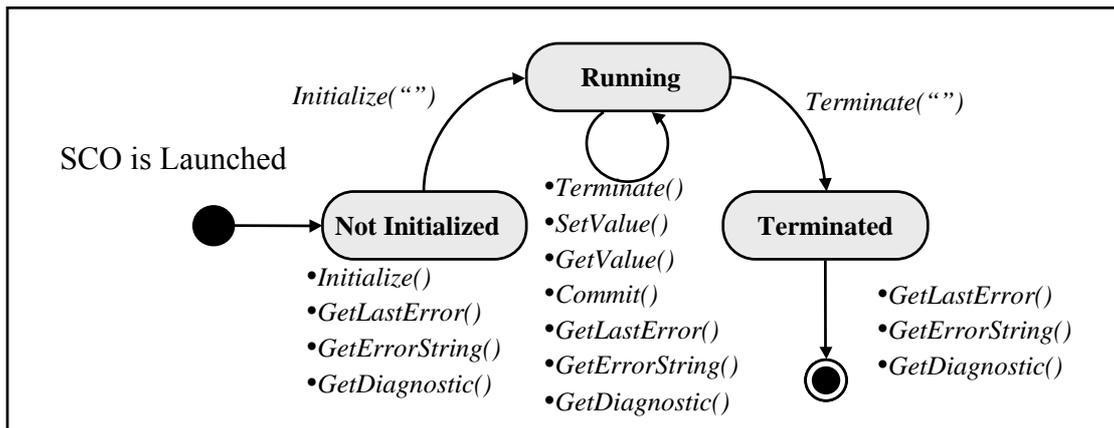


Figure 5.5 API Instance State Transitions and SCORM API

(1) Not Initialized

This is a state where no communication session has been established. It is the state before the SCO has successfully invoked the *Initialize* API method. The SCO is permitted to call the following set of API functions:

- *Initialize()*
- *GetLastError()*
- *GetErrorString()*
- *GetDiagnostic()*

(2) Running

This is a state in which the SCO exchanges data with the LMS after establishing a communication session. The SCO is permitted to call the following set of API methods:

- *Terminate()*
- *SetValue()*
- *GetValue()*
- *Commit()*
- *GetLastError()*
- *GetErrorString()*
- *GetDiagnostic()*

(3) Terminated

This is a state in which the SCO has successfully invoked the *Terminate* function. The SCO is permitted to call the following set of API methods:

- *GetLastError()*
- *GetErrorString()*
- *GetDiagnostic()*

5.3.6 API Error Code Overview

All error codes are character strings representing integers (0 – 65535). The IEEE standard has reserved codes between 0 and 999. Additional error codes may be

defined by implementers using the remaining numbers from 1000 to 65535. The IEEE has defined the error code categories shown in Table 5.4.

Table 5.4 Categories of Error Codes

Error Code Category	Error Code Range	Description
No Error	0	The LMS returns this value when there is no error
General Errors	100 – 199	Errors that occur during the handling of API methods
Syntax Errors	200 – 299	Syntax errors found in the invoked API methods
RTS Errors	300 – 399	Errors associated with the implementation of the run-time system
Data Model Errors	400 – 499	Errors in the data sent to the LMS or the data received from the LMS
Implementation-defined Errors	1000 - 65535	For LMS implementers to use for their specific purposes

Table 5.5 shows the details of the API implementation errors.

Table 5.5 API Error Code Details

Code	Message	Description	API Methods
0	No Error	This code is returned when there is no error state.	All methods
101	General Exception	General errors occurred during the processing of API method requests.	<i>Initialize()</i>
102	General Initialization Failure	An error occurred during the initialization process of a communication session. The communication state remains “ <i>Not Initialized</i> ”.	<i>Initialize()</i>
103	Already Initialized	After the communication was successfully established, the SCO attempted to initialize it again.	<i>Initialize()</i>
104	Content Instance Terminated	The SCO attempted to invoke the <i>Initialize</i> method after the communication session was terminated.	<i>Initialize()</i>
111	General Termination Failure	A general failure occurred when an attempt was made to terminate the session.	<i>Terminate()</i>
112	Termination Before Initialization	The SCO attempted to terminate a communication session before initializing the session.	<i>Terminate()</i>

Code	Message	Description	API Methods
113	Termination After Termination	The SCO attempted to terminate the communication session after the communication session was successfully terminated.	<i>Terminate()</i>
122	Retrieve Data Before Initialization	The SCO attempted to retrieve data before it successfully initialized a communication session.	<i>GetValue()</i>
123	Retrieve Data After Termination	The SCO attempted to retrieve data after termination of the corresponding communication session.	<i>GetValue()</i>
132	Store Data Before Initialization	The SCO invoked a <i>SetValue</i> method of the API Instance to store data before it successfully initialized a communication session.	<i>SetValue()</i>
133	Store Data After Termination	The SCO invoked a <i>SetValue</i> method of the API Instance to store data after termination of the corresponding communication session.	<i>SetValue()</i>
142	Commit Before Initialization	The SCO invoked a <i>Commit</i> method to save its data to a persistent store in the LMS before it successfully initialized a communication session.	<i>Commit()</i>
143	Commit After Termination	The SCO invoked a <i>Commit</i> method to save its data to a persistent store of the LMS after termination of its communication session.	<i>Commit()</i>
201	General Argument Error	An attempt was made to pass an invalid argument to the invoked API method.	<i>Initialize()</i> <i>Terminate()</i> <i>Commit()</i>
301	General Get Failure	A general failure occurred during the attempt to retrieve the requested data and there is no other error information available. The communication state remains “ <i>Running</i> ” in this situation.	<i>GetValue()</i>
351	General Set Failure	A general failure occurred during the attempt to set the requested data and there is no other error information available. The communication state remains “ <i>Running</i> ” in this situation.	<i>SetValue()</i>

Code	Message	Description	API Methods
391	General Commit Failure	A general failure occurred during the attempt to commit the data and there is no other error information available. The communication state remains “ <i>Running</i> ” in this situation.	<i>Commit()</i>
401	Undefined Data Model Element	The invoked API method contains a parameter that the API Instance cannot recognize. The communication state remains “ <i>Running</i> ” in this situation.	<i>GetValue()</i> <i>SetValue()</i>
402	Unimplemented Data Model Element	The invoked API method contains a parameter that has not been implemented by the LMS. The communication state remains “ <i>Running</i> ” in this situation. This should not occur when accessing SCORM data model elements, but may occur when accessing extension data model elements.	<i>GetValue()</i> <i>SetValue()</i>
403	Data Model Element Value Not Initialized	The SCO attempted to retrieve the value of a data model element that has not been initialized. Note that some data model elements are initialized by an SCO while others are initialized by the LMS.	<i>GetValue()</i>
404	Data Model Element Is Read Only	The SCO attempted to set a new value to a read-only data model element.	<i>SetValue()</i>
405	Data Model Element Is Write Only	The SCO attempted to retrieve the value of a write-only data model element.	<i>GetValue()</i>
406	Data Model Element Type Mismatch	The SCO attempted to store a value that was of an incorrect data type for the target data model element.	<i>SetValue()</i>
407	Data Model Element Value Out Of Range	The SCO attempted to store an invalid value that was outside the specified range for the target data model element.	<i>SetValue()</i>

Code	Message	Description	API Methods
408	Data Model Dependency Not Established	This error code is designed for handling a situation where an SCO attempts to read the value from, or write a value to, a data model element that has a dependence relationship defined with one or more other elements when the dependence has not yet been established.	<i>GetValue()</i> <i>SetValue()</i>

Table 5.6 compares the SCORM 1.2 and SCORM 2004 error codes in conjunction with the API Instance state transitions.

Table 5.6 Comparison of Error Codes between SCORM 1.2 and SCORM 2004

SCORM 1.2 Error Code	SCORM 2004 Error Code
0 - No error	0 - No error
101 - General Exception	101 - General Exception
	102 - General Initialization Failure
	103 - Already Initialized
	104 - Content Instance Terminated
	111 - General Termination Failure
	112 - Termination Before Initialization
	113 - Termination After Termination
	122 - Retrieve Data Before Initialization
	123 - Retrieve Data After Termination
	132 - Store Data Before Initialization
	133 - Store Data After Termination
	142 - Commit Before Initialization
	143 - Commit After Termination
201 - Invalid argument error	201 - General Argument Error
202 - Element cannot have children	
203 - Element not an array. Cannot have count	301 - General Get Failure
	351 - General Set Failure
	391 - General Commit Failure
401 - Not implemented error	401 - Undefined Data Model Element
401 - Not implemented error	402 - Unimplemented Data Model Element
301 - Not initialized	403 - Data Model Element Value Not Initialized
403 - Element is read only	404 - Data Model Element Is Read Only
404 - Element is write only	
402 - Invalid set value, element is a keyword	405 - Data Model Element Is Write Only
405 - Incorrect Data Type	406 - Data Model Element Type Mismatch
	407 - Data Model Element Value Out Of Range
	408 - Data Model Dependency Not Established

5.4 Data Model

5.4.1 Data Model Overview

The SCORM Run-Time Environment Data Model specification is based on the *IEEE P1484.11.1 Draft Standard for Learning Technology – Data Model for Content Object Communication*. This standard defines a set of data model elements that can be used for communication from SCOs to the LMS. The data model includes, among other things,

- Information about the learner
- Interactions between an SCO and the LMS
- Objectives
- Success status and completion status.

The data model elements are defined in such a way that they can be used for a variety of content purposes.

The main uses of the Run-Time Environment data are

- To track the learner’s progress and status
- To support sequencing decisions
- To report on the learner’s overall interactions with the SCO.

The SCORM Version 1.2 Run-Time Environment Data Model was based on the *AICC CMI001 Guideline for Interoperability*. Since the release of SCORM Version 1.2, AICC has submitted CMI001 to the IEEE for standardization. SCORM 2004 has introduced changes to the data model in accordance with *the IEEE 1484.11.1 Draft Standard*. The major changes can be summarized as follows:

- All data model elements have become mandatory for the LMS to implement.
- Changes to the data model
 - Removal of *cmi.core* and *cmi.student_data*
 - Addition of *score.scaled*
 - Addition of sequencing data model elements corresponding to *objectives*
- Detailed specification of interactions
- Adoption of Unicode (ISO-10646-1) for multi-language implementation support, including multi-byte code.

5.4.2 Data Model Basics

5.4.2.1 Data Model Elements

To differentiate them from the other data model elements, the names of all Run-Time Environment Data Model elements start with “*cmi*”. This indicates to the LMS that these data model elements are part of the IEEE P148411.1 standard. It is anticipated that when a different data model is to be developed, its data elements will be named with a different prefix (for example, *adl.elementName* instead of *cmi.elementName*).

It is mandatory for the LMS to implement all these data model elements and to guarantee their behaviors.

Content developers can choose to use any or all of the data model elements in SCOs.

The names of all the data model elements must be bound to ECMAScript character strings with dot notation (for example, *cmi.success_status*).

5.4.2.2 Data Model Effects on Sequencing

Using the Run-Time Data Model elements, each SCO reports the results of interactions between the learner and the SCO to the LMS during a session. The LMS uses the information reported from an SCO when it makes a sequencing decision as to which activity is to be delivered next. For example, when an SCO reports its attempt status as completed (by the learner) to the LMS using the data element “*cmi.completion_status*” (as tracking information), the LMS regards the activity associated with the SCO as completed and selects another activity for delivery. Some data model elements of the RTE are related to the tracking information for each activity, and thus affect the sequencing process.

5.4.2.3 Handling Collections

Some data model elements are collections of related elements defined for specific requirements. Such collection data is referred to as a record of data in the SCORM RTE book. Each record of data is collected as an entity in an array. The record of data is accessed using an index value representing the record of data’s position in the array. All arrays are implemented with a starting index of 0 (zero-based arrays).

The following data model elements are defined as collections of data records:

- Comments from learner (*cmi.comments_from_learner*)
- Comments from LMS (*cmi.comments_from_lms*)
- Objectives (*cmi.objectives*)
- Interactions (*cmi.interactions*)

These data model elements are to ensure that SCOs can track multiple comments, objectives and/or interactions. The Objectives and Interactions data model elements contain an identifier data model element that indicates a unique identifier for each of the SCO’s Objectives and Interactions.

The data model elements in a collection are referred to using a dot-number notation (represented as *n* in the following.)

cmi.objective.n.completion_status

For example, the value of the data model element representing the completion status of the first objective in an SCO is described as “*cmi.objective.0.completion_status*”, and that of the fourth objective is described as “*cmi.objective.3.completion_status*”. The *_count* keyword data model element is used to determine the current number of data model elements in the collection. For example, to determine the number of objectives currently stored for the SCO, the following API method call would be used:

```
var numOfObjectives = GetValue("cmi.objectives._count");
```

5.4.2.4 Smallest Permitted Maximum (SPM)

In SCORM 2004, the smallest permitted maximums (SPMs) are defined for the data model elements in two cases. The SPM is defined as the length of a character string value or the number of entries (data model elements) in collections; that is, the SPM is defined as the smallest permitted length of a character string that any implementation must accept and process or the smallest permitted number of entries in a collection. An implementation may elect to support the storage of more than the SPM. If an implementation only supports the SPM and truncates a character string, it is necessary for content developers to be aware of the SPM and what may happen if it is exceeded.

5.4.2.5 Keyword Data Model Elements

SCORM defines a set of data model elements for getting the data managed by the LMS and the status data of some data model elements. These data model elements are called keyword data model elements. The keyword data model elements can only be applied to certain data model elements and are implemented as read-only data model elements.

- *_version*: The *_version* keyword data model element is used to retrieve the version of the data model supported by the LMS.
- *_count*: The *_count* keyword data model element is used to retrieve the number of data model elements contained in a collection.
- *_children*: The *_children* keyword data model element is used to retrieve the entire set of child data model elements included in a parent data model element supported by the LMS. The LMS must implement the return value for this *_children* request as a list of character strings delimited by a comma, with each string representing a child data model element. This data model element can only be applied to a data model element that has its child data model elements.

5.4.2.6 Reserved Delimiters

A special reserved delimiter must be used to represent

- The language type for a particular character string (Data type: *localized_string_type*)
- The indication as to whether the order matters in the learner's responses to an interaction
- The indication as to whether the case matters in the learner's response to an interaction
- A set of values in a list or pairs of values.

For each of the above cases, a default value is provided where applicable. This default value is used if the special reserved delimiter is not specified. In any case, the reserved delimiters should not be counted toward the value of the SPM.

Table 5.7 Reserved Delimiters

Reserved Delimiter Syntax	Default Value	Example
{lang=<language_type>}	{lang=en}	{lang=en}
{case_matters=<boolean>}	{case_matters=false}	{case_matters=true} {case_matters=false}
{order_matters=<boolean>}	{order_matters=true}	{order_matters=true} {order_matters=false}
[.]	Not applicable. A value must be provided.	Used to separate a pair of values that are related for an interaction: 1[.]a
[,]	Not applicable. A value must be provided.	Used to separate a set of values for an interaction's collection: 1[.]a[,]2[.]c[,]3[.]b
[:]	Not applicable. A value must be provided.	Used to represent a separator between a range of numeric values: 1[:]100 – a range where the numeric value is between 1 and 100 (inclusive)

5.4.2.7 Data Types

Each of the data type elements has a designated data type. The values of a data model element must satisfy the data type requirements of the element. Below is a description of the specific requirements for each data model element.

(1) *characterstring*

A string of characters defined in ISO 10646, which is equivalent to the Unicode standard.

(2) *localized string type*

A character string that contains a character string indicating the language of the *characterstring*. SCORM uses a reserved delimiter to represent the language for a character string: {lang=<language_type>}. Whether to specify this localized string type is optional. If it is not specified, the default language is en, which means English as in {lang=en}. The syntax for specifying this data type element is as follows:

“{lang=<language_type>}<actual_character_string>”

Example: {lang=ja}鈴木 一郎

(3) language type

The data type used to represent the language. The format of this data type element is a character string consisting of a language code (langcode) followed by zero or more hyphen-prefixed subcodes (subcode).

language_type ::= langcode ["-" subcode]*

Example: ja en-GB

(4) long identifier type

This data type element represents a label or identifier. This label or identifier must be unique within the context of an SCO. It must conform to the syntax defined for the universal resource identifier (URI). SCORM recommends that the URI be a globally unique identifier in the form of a uniform resource name (URN). The values of this long identifier type element should be implemented with an SPM of 4000 characters.

<URN> ::= "urn." <NID> "." <NSS>

where <NID> is a name space identifier and <NSS> is a name space string.

Example: urn:ADL:interaction-id-0001

(5) short identifier type

This label or identifier must be unique within the context of an SCO. This data type element must conform to the syntax defined for the URI. It is not assumed that the values of this data type element are globally unique identifiers. The values should be implemented with an SPM of 250 characters.

(6) integer

This data type element is a member of a set of positive whole numbers (e.g., 1, 2, 3), negative whole numbers (e.g., -1, -2, -3) and zero (0).

(7) state

Some of the data model elements values have a defined set of states. This is defined by a statement like the following:

Example: state (browse,normal,review)

(8) real(10,7)

This data type element refers to a real number with seven significant digit precision.

(9) time(second, 10, 0)

This time data type has a required precision of 1 second and an optional precision of 0.01 second.

Example: 2003-07-25T03:00:00

(10) timeinterval(second, 10,2)

The value for this data type element represents a period of elapsed time with a precision of 0.01 second.

Example: P1Y3M2DT3H

which means 1 year, 3 months, 2 days and 3 hours.

5.4.2.8 SCORM Run-Time Environment Data Model Extension

The SCORM Run-Time Environment Data Model is expected to be implemented without extension. If an LMS receives an API request with an undefined data model element, the LMS should handle it as an error.

5.4.3 SCORM Run-Time Environment Data Model

5.4.3.1 Data Model Overview

The SCORM Run-Time Environment Data Model contains a set of data that can be tracked by an SCO on the LMS during the run-time of the SCO. These data model elements are used to track items such as status, scores, interactions, and objectives. Some data model elements are used to exchange data between the SCO and the LMS, while others may be used to affect the sequencing process for other SCOs that are associated within the activity tree. Table 5.8 summarizes the data model elements.

Table 5.8 List of SCORM Run-Time Environment Data Model Elements

No	Data Model Element	Description
1	<i>cmi.comments_from_learner</i>	Contains text from the learner
2	<i>cmi.comments_from_lms</i>	Contains comments and annotations to be provided for the learner
3	<i>cmi.completion_status</i>	Indicates whether the learner has completed the SCO
4	<i>cmi.completion_threshold</i>	Indicates a value against which the learner's measure of progress is to be compared to determine whether the learner has completed the SCO
5	<i>cmi.credit</i>	Indicates whether the learner's performance with the SCO is to be credited
6	<i>cmi.entry</i>	Contains information indicating whether the learner has accessed the SCO before
7	<i>cmi.exit</i>	Contains information as to why and how the learner exited from the SCO
8	<i>cmi.interactions</i>	Defines information concerning an interaction for the purpose of measurement or assessment
9	<i>cmi.launch_data</i>	Provides data specific to an SCO that the SCO can use for initialization
10	<i>cmi.learner_id</i>	Identifies the learner for whom the SCO instance was launched
11	<i>cmi.learner_name</i>	Represents the name of the learner
12	<i>cmi.learner_preference</i>	Specifies learner preferences associated with the learner's use of the SCO
13	<i>cmi.location</i>	Represents a location in the SCO. Its value and meaning are determined by the SCO.

No	Data Model Element	Description
14	<i>cmi.max_time_allowed</i>	Indicates the accumulated amount of time that the learner is allowed to use for an SCO in the learner attempt
15	<i>cmi.mode</i>	Identifies the modes in which the SCO may be presented to the learner
16	<i>cmi.objectives</i>	Specifies learning objectives associated with an SCO
17	<i>cmi.progress_measure</i>	Identifies a measure of the progress the learner has made toward completing the SCO
18	<i>cmi.scaled_passing_score</i>	Identifies the scaled passing score for an SCO
19	<i>cmi.score</i>	Identifies the learner's score for the SCO
20	<i>cmi.session_time</i>	Identifies the amount of time that the learner has spent in the current session for the SCO
21	<i>cmi.success_status</i>	Indicates whether the learner has mastered the SCO
22	<i>cmi.suspend_data</i>	Provides additional space for storing and retrieving information relating to the suspension of an SCO
23	<i>cmi.time_limit_action</i>	Indicates what the SCO should do when the maximum time allowed is exceeded
24	<i>cmi.total_time</i>	Provides the sum of all of the learner's session times accumulated in the learner's current attempt

5.4.3.2 Details of the Data Model Elements

Table 5.9 SCORM RTE Data Model Elements – Detail

No	Data Model Element	Data Type	Value Space	SCO	Remarks
0.1	<i>cmi._version</i>	characterstring	ISO-10646-1	R	Delimited by a period "1.0"
1.	<i>cmi.comments_from_learner</i>	collection SPM: 250		-	
1.0.1	<i>cmi.comments_from_learner._children</i>	characterstring	ISO-10646-1	R	
1.0.2	<i>cmi.comments_from_learner._count</i>	integer	Non-negative integer	R	
1.1	<i>cmi.comments_from_learner.n.comment</i>	localized_string_type SPM: 4000	Localized string type (ISO-10646-1)	R/W	Not initialized
1.2	<i>cmi.comments_from_learner.n.location</i>	characterstring SPM: 250	ISO-10646-1	R/W	
1.3	<i>cmi.comments_from_learner.n.timestamp</i>	time (second,10,0)		R/W	
2.	<i>cmi.comments_from_lms</i>	collection SPM: 100		-	
2.0.1	<i>cmi.comments_from_lms._children</i>	characterstring	ISO-10646-1	R	
2.0.2	<i>cmi.comments_from_lms._count</i>	integer	Non-negative integer	R	
2.1	<i>cmi.comments_from_lms.n.comment</i>	localized_string_type SPM: 4000	Localized string type (ISO-10646-1)	R	
2.2	<i>cmi.comments_from_lms.n.location</i>	characterstring SPM: 250	ISO-10646-1	R	
2.3	<i>cmi.comments_from_lms.n.timestamp</i>	time (second,10,0)		R	
3.	<i>cmi.completion_status</i>	state	"complete" "incomplete" "not_attempted" "unknown"	R/W	Default: "unknown" It is assumed that an SCO is to set the value of this element, and that it affects sequencing .
4.	<i>cmi.completion_threshold</i>	real(10,7) range (0..1)		R	LMS determines the completion status by comparing the 「 17. cmi.progress_measure 」 against the value. The result of this has a higher priority than 「 3. cmi.completion_status 」 which is set by an SCO. This is initialized with a value defined for <adlcp:completionThreshold> in the manifest file.

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

No	Data Model Element	Data Type	Value Space	SCO	Remarks
5.	<i>cmi.credit</i>	state	"credit" "no_credit"	R	Default: "credit"
6.	<i>cmi.entry</i>	state	"ab_initio" "resume" ""(Empty string)	R	
7.	<i>cmi.exit</i>	state	"time-out" "suspend" "logout" "normal" ""(Empty string)	W	
8.	<i>cmi.interactions</i>	collection SPM: 250		-	
8.0.1	<i>cmi.interactions._children</i>	characterstring	ISO-10646-1	R	
8.0.2	<i>cmi.interactions._count</i>	integer	Non-negative integer	R	
8.1	<i>cmi.interactions.n.id</i>	long_identifier_type SPM: 4000	URI (RFC 2396)-compliant character string URN (RFC 2141) Recommended	R/W	Must be unique within the context of an SCO.
8.2	<i>cmi.interactions.n.type</i>	state	"true-false" "choice" "fill-in" "long-fill-in" "likert" "matching" "performance" "sequencing" "numeric" "other"	R/W	The correct_response and learner_response elements are dependent on this data model element, and this must be set before these two dependent elements are used.
8.3	<i>cmi.interactions.n.objectives</i>	collection SPM: 10		-	
8.3.0.1	<i>cmi.interactions.n.objectives._count</i>	integer	Non-negative integer 0	R	
8.3.1	<i>cmi.interactions.n.objectives.n.id</i>	long_identifier_type SPM: 4000	URI (RFC 2396)-compliant character string URN (RFC 2141) Recommended	R/W	
8.4	<i>cmi.interactions.n.timestamp</i>	Time (second,10,0)		R/W	
8.5	<i>cmi.interactions.n.correct_responses</i>	collection SPM: 10		-	
8.5.0.1	<i>cmi.interactions.n.correct_responses._count</i>	integer	Non-negative integer	R	

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

No	Data Model Element	Data Type	Value Space	SCO	Remarks
8.5.1	<i>cmi.interactions.n.correct_responses.n.pattern</i>	Dependent on 8.2 cmi.interactions.n.type		R/W	
8.6	<i>cmi.interactions.n.weighting</i>	real (10,7)	Real with 7 significant decimal digits	R/W	
8.7	<i>cmi.interactions.n.learner_response</i>	Dependent on 8.2 cmi.interactions.n.type		R/W	
8.8	<i>cmi.interactions.n.result</i>	state	"correct" "incorrect" "unanticipated" "neutral" real(10,7)	R/W	
8.9	<i>cmi.interactions.n.latency</i>	timeinterval (second,10,2)		R/W	
8.10	<i>cmi.interactions.n.description</i>	Localized_string_type SPM: 250	Localized string type	R/W	
9.	<i>cmi.launch_data</i>	characterstring SPM: 4000	ISO-10646-1	R	Initialized by the with a value defined for <adlcp:dataFrom LMS> in the manifest file.
10.	<i>cmi.learner_id</i>	long_identifier_type SPM: 4000	URI (RFC 2396)-compliant character string URN (RFC 2141) Recommended	R	Provided by the LMS
11.	<i>cmi.learner_name</i>	localized_string_type SPM: 250	Localized string type	R	Provided by the LMS
12..	<i>cmi.learner_preference</i>			-	
12.0.1	<i>cmi.learner_preference._children</i>	characterstring	ISO-10646-1	R	
12.1	<i>cmi.learner_preference.audio_level</i>	real(10,7) , range (0..*)	A real number with 7 significant decimal digits	R/W	
12.2	<i>cmi.learner_preference.language</i>	language_type SPM: 250	ISO-646	R/W	
12.3	<i>cmi.learner_preference.delivery_speed</i>	real(10,7) , range (0..*)	A real number with 7 significant decimal digits	R/W	
12.4	<i>cmi.learner_preference.audio_captions</i>	state	"-1" "0" "1"	R/W	Corresponding tokens: "off" "no_change" "on"

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

No	Data Model Element	Data Type	Value Space	SCO	Remarks
13.	<i>cmi.location</i>	characterstring SPM: 1000	ISO-10646-1	R/W	The initial value is "" (Empty string) An LMS should not update or interpret this value. This can be used to save the SCO's exit point.
14.	<i>cmi.max_time_allowed</i>	timeinterval (second,10,2) With resolution to 0.01 second.		R	Initialized with a value defined for <imsss:attemptAbsoluteDurationLimit> in the manifest file.
15.	<i>cmi.mode</i>	state	"browse" "normal" "review"	R	Default: "normal" Related with : 「 5. cmi.credit 」
16.	<i>cmi.objectives</i>	collection SPM: 100		-	
16.0.1	<i>cmi.objectives._children</i>	characterstring	ISO-10646-1	R	
16.0.2	<i>cmi.objectives._count</i>	integer	Non-negative integer	R	
16.1	<i>cmi.objectives.n.id</i>	long_identifier_type SPM: 4000	URI (RFC 2396)-compliant character string URN (RFC 2141) Recommended	R/W	Must be unique at least within the context of an SCO. Initialized with the value defined as the identifier for <imsss:objectives> in the manifest file.
16.2	<i>cmi.objectives.n.score</i>			-	
16.2.0.1	<i>cmi.objectives.n.score._children</i>	characterstring	ISO-10646-1	R	
16.2.1	<i>cmi.objectives.n.score.scaled</i>	real (10,7) range (-1..1)	Real number with 7 significant decimal digits within the range between -1.0 and 1.0	R/W	The data model element affects the objective measure for the activity associated with the SCO.
16.2.2	<i>cmi.objectives.n.score.raw</i>	real (10,7)	Real number with 7 significant decimal digits	R/W	

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

No	Data Model Element	Data Type	Value Space	SCO	Remarks
16.2.3	<i>cmi.objectives.n.score.min</i>	real (10,7)	Real number with 7 significant decimal digits	R/W	
16.2.4	<i>cmi.objectives.n.score.max</i>	real (10,7)	Real number with 7 significant decimal digits	R/W	
16.2.5	<i>cmi.objectives.n.success_status</i>	state	"passed" "failed" "unknown"	R/W	The data model element affects the Objective Progress Status for the activity associated with the SCO.
16.2.6	<i>cmi.objectives.n.completion_status</i>	state	"completed" "incomplete" "not_attempted" "unknown"	R/W	
16.2.7	<i>cmi.objectives.n.progress_measure</i>	real (10,7) range (0..1)	Real number with 7 significant decimal digits within the range between 0 and 1.0.	R/W	
16.2.8	<i>cmi.objectives.n.description</i>	Localized _string_type SPM: 250	Localized string type (ISO-10646-1)	R/W	
17.	<i>cmi.progress_measure</i>	real (10,7) range (0..1)	Real number with 7 significant decimal digits within the range between 0 and 1.0	R/W	Mapped with the value of 「 3. cmi.completion_status 」 0 "not attempted" 1 " completed" 0 > value < 1 " incomplete" Note) This applies when there is no defined threshold value.
18.	<i>cmi.scaled_passing_score</i>	real (10,7) range (-1..1)	Real number with 7 significant decimal digits within the range between -1.0 and 1.0	R	Initialized with the value defined for <imsss:minNormalizedMeasure> in the manifest file.
19.	<i>cmi.score</i>			-	Mainly used by SCOs
19.0.1	<i>cmi.score._children</i>	characterstring	ISO-10646-1	R	

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

No	Data Model Element	Data Type	Value Space	SCO	Remarks
19.1	<i>cmi.score.scaled</i>	real (10,7) range (-1..1)	Real number with 7 significant decimal digits within the range between -1.0 and 1.0	R/W	The value of this data model element must be synchronized with the initial value of the Objective Measure Status for the SCO .
19.2	<i>cmi.score.raw</i>	real (10,7)	Real number with 7 significant decimal digits	R/W	
19.3	<i>cmi.score.max</i>	real (10,7)	Real number with 7 significant decimal digits	R/W	
19.4	<i>cmi.score.min</i>	real (10,7)	Real number with 7 significant decimal digits	R/W	
20.	<i>cmi.session_time</i>	timeinterval (second,10,2) With resolution to 0.01 second		W	
21.	<i>cmi.success_status</i>	state	"passed" "failed" "unknown"	R/W	<p>Initialized by the SCO</p> <p>The LMS cannot directly change this data model element, but it can indirectly change the value by setting a value to 「 18. cmi.scaled_passing_score 」</p> <p>(In this case, the LMS can override the value of the data model element reported by the SCO.)</p> <p>This data model element must be synchronized with the initial Objective Measure Status value for the SCO .</p>

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

No	Data Model Element	Data Type	Value Space	SCO	Remarks
22.	<i>cmi.suspend_data</i>	characterstring SPM: 4000	ISO-10646-1	R/W	The LMS should not attempt to interpret or change this data, which is meant to be used by the SCO for current or subsequent sessions. Related to : ┌ 13. cmi.location └
23.	<i>cmi.time_limit_action</i>	state	“exit,message” “continue,message” “exit,no message” “continue,no message”	R	Initialized with the value defined for <adlcp:timeLimitAction> in the manifest file. Default: “continue,no message”
24.	<i>cmi.total_time</i>	timeinterval (second,10,2) With resolution to 0.01 second		R	The LMS cannot determine the latest value for this data element until the SCO sets session time values to the cmi.session_time element.

* Notes on the SCO column: R: Read-only, W: Write-only, R/W: Read and Write

6 Features of SCORM 2004 Content

This section explains how to represent the characteristic behaviors of SCORM 2004 content by using actual examples to make it easy to understand the correspondence between behavior and code. It also explains, using examples, how educational concepts like completion and objectives are represented in the SCORM 2004 standard.

6.1 Sequencing Features

This subsection describes the features of sequencing, added as a main function in SCORM 2004, and explains how to encode sequencing strategies.

6.1.1 Specifying Sequencing Strategies

The content developer controls the behavior of content by describing content structures and their associated behavior rules (sequencing rules) in the manifest file (imsmanifest.xml). The majority of sequencing rules are specified in cluster units, and the rules specified with the root of a cluster are applied to the entire cluster.

In addition, tracking information used in the application of sequencing rules is associated with each individual activity. The tracking information for a cluster, which is the tracking information for a parent activity, is rolled up from the data for its child activities. How the parent activity's data should be rolled up is specified by the content developer using a set of rollup rules.

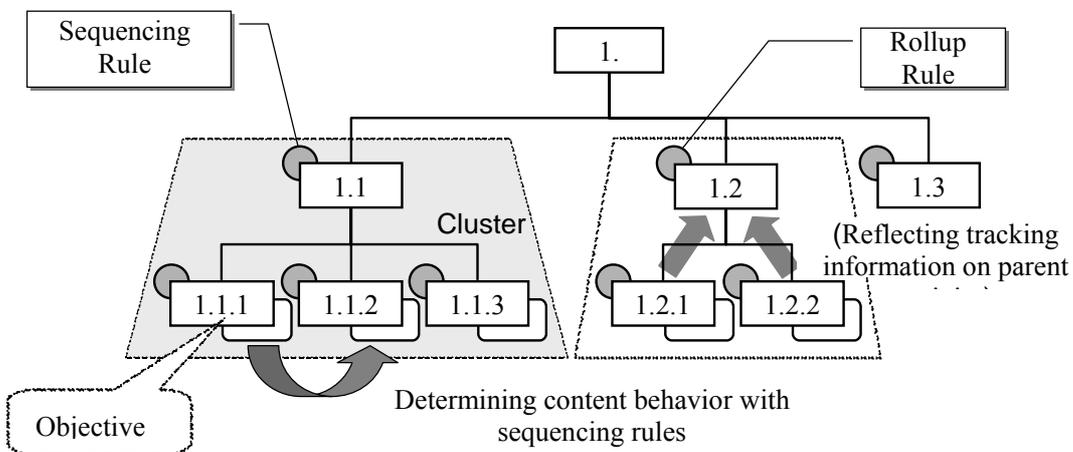


Figure 6.1 Content Behavior Control through Sequencing Rules

6.1.2 Specifying Sequencing Control Modes

(1) Overview

Sequencing control modes control the sequencing behavior for a cluster, and these control modes are used to specify the degree of freedom allowed for the learner to select SCOs and decide upon a navigation path.

The sequencing control modes can be specified for each cluster, and the specified control modes are applied to the cluster's child activities. For example, a control

mode specification may be [*The activities in this cluster should be presented in the forward direction only and backward movement is not allowed.*]

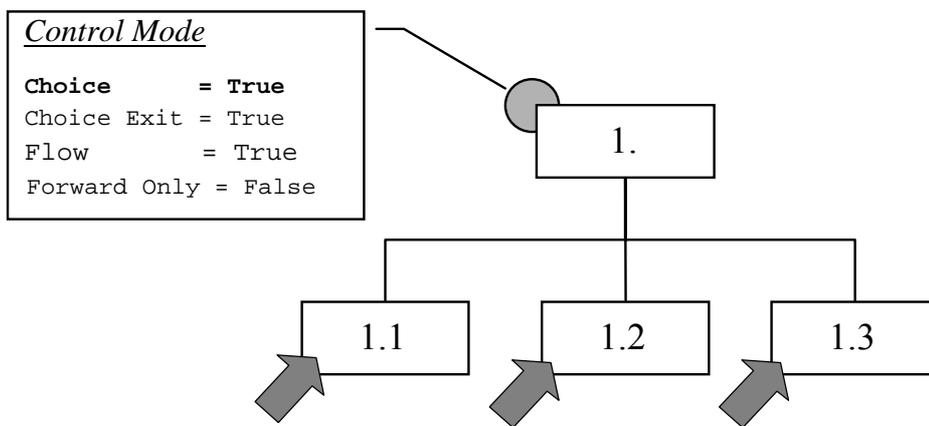
(2) Specification Method

Specify the `<imsss:controlMode>` element in the manifest file for each cluster.

(3) Description of Each Attribute

a. Sequencing Control Choice

This sequencing control mode is used to specify whether the learner will be allowed to choose a candidate activity to attempt from the menu. If the value is *True*, then the learner is allowed to choose any activity within the cluster. The default value is *True*.



The learner is free to choose.

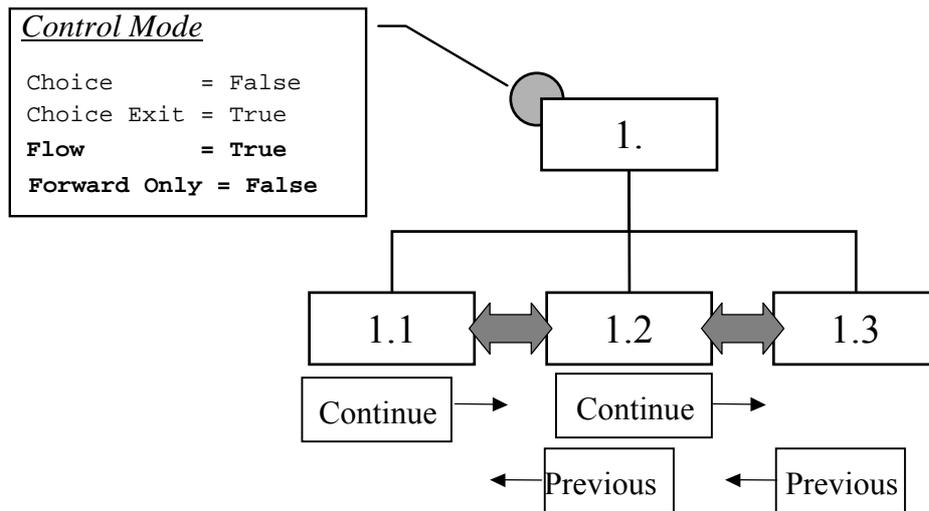
Figure 6.2 Sequencing Control Choice

b. Sequencing Control Choice Exit

This sequencing control mode is used to specify whether a *Choice* sequencing request can be used to move from the current activity or any of its descendant activities to another activity outside of the current activity's control. The default value is *True*.

c. Sequencing Control Flow

This sequencing control mode is used to specify whether to allow movement from one activity to another within the cluster in a forward or backward direction using a *Continue* sequencing request or a *Previous* sequencing request. The default is *True*.

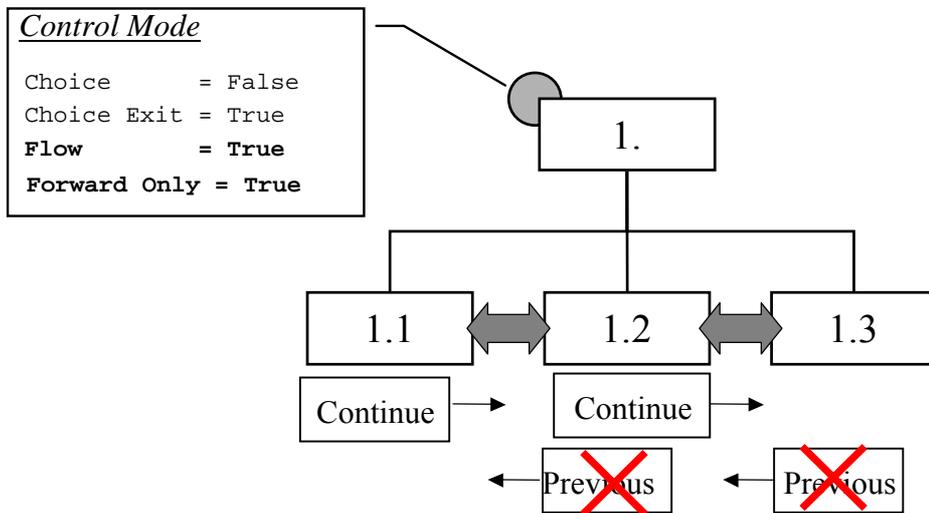


The learner is free to move.

Figure 6.3 Sequencing Control Flow

d. Sequencing Control Forward Only

This sequencing control mode is used to specify whether to allow only forward movement among the activities in a cluster without allowing backward movement. If the value is *True*, the learner is allowed to move in the forward direction only, with a *Continue* sequencing request or a *Choice* sequencing request in the forward direction.



The learner cannot move backward.

Figure 6.4 Sequencing Control Forward Only

e. Use Current Attempt Objective Information

This sequencing control mode is used to specify whether the objective information from the current attempt on a child activity only is used in the rule evaluation for the cluster as tracking information regarding its child activities. The default is *True*. If the value is *False*, the latest information, including the most recent previous attempt on a child activity, is used for the rule evaluation.

f. Use Current Attempt Progress Information

This sequencing control mode is used to specify whether the progress information from the current attempt on a child activity only is used in the rule evaluation for the cluster as the tracking information regarding its child activities. The default is *True*. If the value is *False*, the latest information, including the most recent previous attempt on a child activity, is used for the rule evaluation.

(4) Coding Example

```
<item identifier="PRETEST1">
  <title>A Pretest</title>
  <item identifier="Q1" isvisible = "false" identifierref=" RQ1">
    <title>Question1</title>
  </item>
  <item identifier="Q2" isvisible = "false" identifierref=" RQ2">
    <title>Question2</title>
  </item>
  <item identifier="Q3" isvisible = "false" identifierref=" RQ3">
    <title>Question3</title>
  </item>
  <imsss:sequencing>
    <imsss:controlMode choice="false" choiceExit="false"
      flow="true" forwardOnly = "true"/>
  </imsss:sequencing>
</item>
```

(5) Note for Implementation

- The *Sequencing Control Flow* should be set to *True* when the *Sequencing Control Choice* is *True*.
Reason: Otherwise, unless the learner specifically selects an activity, nothing will be displayed.
- It will help learner understanding if the specification for the first activity of a cluster contains an indication that its child activities (SCOs) can be freely selected.
- The *Sequencing Control Forward Only* has priority to the *Sequencing Control Flow*.

6.1.3 Specifying Sequencing Rules**6.1.3.1 Sequencing Rules****(1) Overview**

Sequencing rules are used to determine the sequence of learning experiences based on tracking information (objective status). For example, it is possible to specify the behavior of content as follows:

- If a particular learning objective is satisfied, then this activity is to be skipped.
- If a post-test score does not exceed 70 points, the post-test is repeated up to three times.

Condition rules are described on the basis of the **if – then** structure. If the tracking information satisfies a given condition, a constraint is placed on content behavior such as the issuing of a sequencing request or the learner's navigation between activities.

(2) Specification Method

Specify the `<imsss:sequencingRules>` element in the manifest file for each activity.

(3) Component Element Specification**a. Sequencing rules**

Sequencing rules are defined using the **if – then** rule structure.

If [*condition_set*] **Then** [*action*]

There are three rule conditions depending on when the rules are applied if they are satisfied:

- Precondition rules: Rules that are applied prior to the delivery of the associated activity.
- Postcondition rules: Rules that are applied when the associated activity has ended.
- Exit rules: Rules that are applied when a descendent activity has ended.

b. Condition set

The condition set describes a set of conditions that need to be satisfied for the execution of the action part. The condition set is a logical expression that is evaluated as *True* or *False* depending on the values of tracking information elements such as the objective measure and the progress status for the activity. A condition set may contain multiple conditions.

A condition combination indicates how the component conditions are to be combined in an evaluation. The *All* condition combination indicates that the result of all the component conditions must be *True* for the condition set to be evaluated as *True*, and the *Any* condition combination indicates that any one of the component conditions must be evaluated as *True* for the condition set to be evaluated as *True*. The default condition combination is *Any*.

- *all*: Set to *True* if all conditions are *True*.
- *any*: Set to *True* if any condition is *True*.

A condition set and a condition combination are specified in the `<imsss:ruleConditions>` element.

A condition operator is a unary logical operator applied after the evaluation of the rule condition.

- *noOp*: The Boolean value of the associated condition is not changed and is used as is in the rule evaluation.
- *not*: The associated condition is negated in the rule evaluation.

The default condition operator is *noOp*.

The condition elements represent elements whose values are set to *True* or *False* depending on the tracking information for the activity. Condition elements and condition operators are specified in the `<imsss:RuleCondition>` element.

c. Action

The action part determines the activity to be delivered next. It may also be used to issue a new sequencing request or termination request. The action to be performed differs depending on the sequencing condition applied. Only one action can be specified for a sequencing rule `<imsss:ruleAction>`.

- Precondition actions:
 - “Skip”, “Disabled”, “Hidden from Choice”, “Stop Forward Traversal”
- Post-condition actions:
 - “Exit Parent”, “Exit All”, “Retry”, “Retry All”, “Continue” “Previous”
- Exit actions:
 - “Exit”

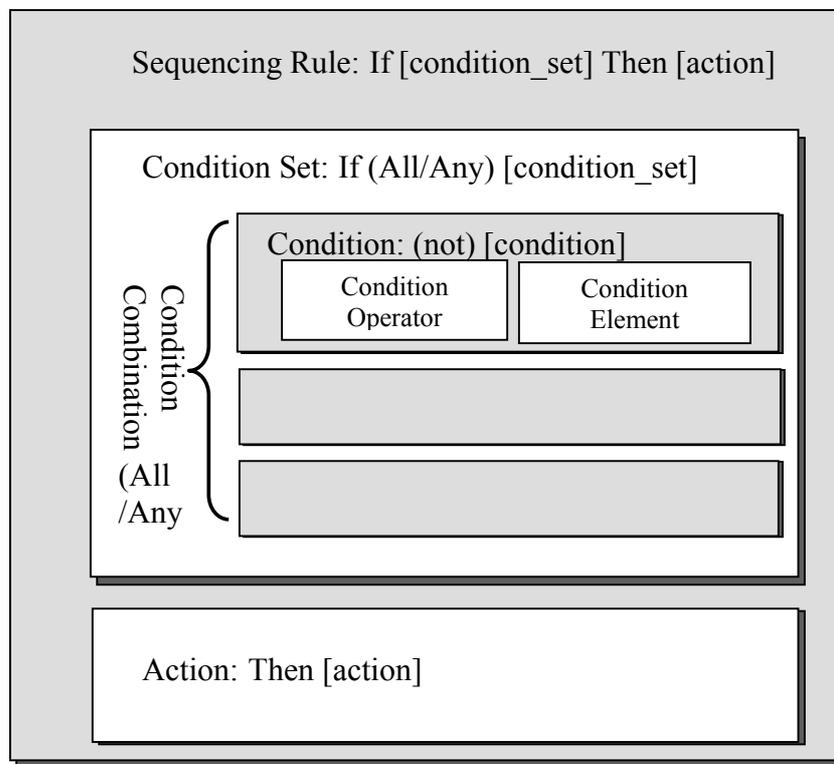


Figure 6.5 Sequencing Rule Description

(4) Coding Example

The coding example below shows how to describe a sequencing rule, *If “not satisfied” Then “retry”*.

```

<item identifier="DRILL1">
  <title>Drill</title>
  <item identifier="Q1" isVisible = "false" identifierref="RQ1">
    <title>Question 1</title>
  </item>
  <item identifier="Q2" isVisible = "false" identifierref="RQ2">
    <title>Question 2</title>
  </item>
  <imsss:sequencing>
    <imsss:sequencingRules >
      <imsss:postConditionRule>
        <imsss:ruleConditions>
          <imsss:ruleCondition operator = "not"
            condition = "satisfied"/>
        </imsss:ruleConditions>
        <imsss:ruleAction action = "retry"/>
      </imsss:postConditionRule>
    </imsss:sequencingRules>
  </imsss:sequencing>
</item>

```

(5) Note for Implementation

If the tracking information to be captured is information regarding the *Objective Satisfied Status* and the *Objective Normalized Measure*, it is necessary to specify the target objective for the *Rule Condition Referenced Objective*, and to specify a threshold value for the *Rule Condition Measure Threshold* against which the *Objective Normalized Measure* is compared.

6.1.3.2 Specifying Precondition Rules

a. Sequencing Rule Description

Precondition rules are defined using the `<preConditionRule>` element.

b. Condition_set

The condition elements that can be used in a condition set are shown in Table 6.1.

Table 6.1 Condition Elements for Precondition Rules

Condition Element	Tracking Information	Description
<i>Satisfied</i>	<i>Objective Satisfied Status</i>	Evaluated as <i>True</i> if the <i>Objective Satisfied Status</i> is "satisfied"
<i>Objective Status Known</i>	<i>Objective Satisfied Status</i>	Evaluated as <i>True</i> unless the <i>Objective Satisfied Status</i> is "unknown"
<i>Objective Measure Greater Than</i>	<i>Objective Normalized Measure</i>	Evaluated as <i>True</i> if the <i>Objective Normalized Measure</i> is greater than the specified threshold value
<i>Objective Measure Less Than</i>	<i>Objective Normalized Measure</i>	Evaluated as <i>True</i> if the <i>Objective Normalized Measure</i> is less than the specified threshold value
<i>Completed</i>	<i>Attempt Completion Status</i>	Evaluated as <i>True</i> if the <i>Attempt Completion Status</i> is "completed"

Condition Element	Tracking Information	Description
<i>Activity Progress Known</i>	<i>Attempt Completion Status</i>	Evaluated as <i>True</i> unless the <i>Attempt Completion Status</i> is “unknown”
<i>Attempted</i>	<i>Activity Attempt Count</i>	Evaluated as <i>True</i> if the <i>Activity Attempt Count</i> is 1 or more
<i>Attempt Limit Exceeded</i>	<i>Activity Attempt Count</i>	Evaluated as <i>True</i> if the <i>Activity Attempt Count</i> is greater than the value specified in the limit condition
<i>Always</i>	None	Always evaluated as <i>True</i>

c. Action

One of the actions shown in Table 6.2 is used for the action part of a precondition rule.

Table 6.2 List of Actions Used in Precondition Rules

Action	Description
<i>Skip</i>	This action is used when the candidate activity to be delivered next is determined by traversing the activity tree with a <i>Continue</i> or a <i>Previous</i> sequencing request.
<i>Disabled</i>	This action is used to disable the activity to be delivered.
<i>Hidden From Choice</i>	This action is used to stop the activity from being presented by a <i>Choice</i> sequencing request.
<i>Stop Forward Traversal</i>	This action is used when the candidate activity to be delivered next is determined by traversing the activity tree in the forward direction.

6.1.3.3 Specifying Postcondition Rules

a. Sequencing Rule Description

Postcondition rules are defined using the `<postConditionRule>` element.

b. Condition Set

The condition elements that can be used in a condition set are the same as those shown above for the precondition rules.

c. Action

One of the actions shown in Table 6.3 is used for the action part of a post-condition rule.

Table 6.3 List of Actions Used in Postcondition Rules

Action	Description
<i>Exit Parent</i>	This action is used to end the parent activity
<i>Exit All</i>	This action is used to end the entire activity tree
<i>Retry</i>	This action is used to specify that the current activity is to be retried. If the associated activity is not a leaf activity, the cluster is retried from its first activity.

Action	Description
<i>Retry All</i>	This action is used to specify that the entire activity tree is to be terminated and then retried
<i>Continue</i>	Move forward
<i>Previous</i>	Move backward

6.1.3.4 Specifying an Exit Rule

a. Sequencing Rule Description

An exit rule is defined using the `<exitConditionRule>` element.

b. Condition Set

The condition elements that can be used in a condition set are the same as those shown above for the precondition rules.

c. Action

The action shown below is used for the action part of an exit condition rule.

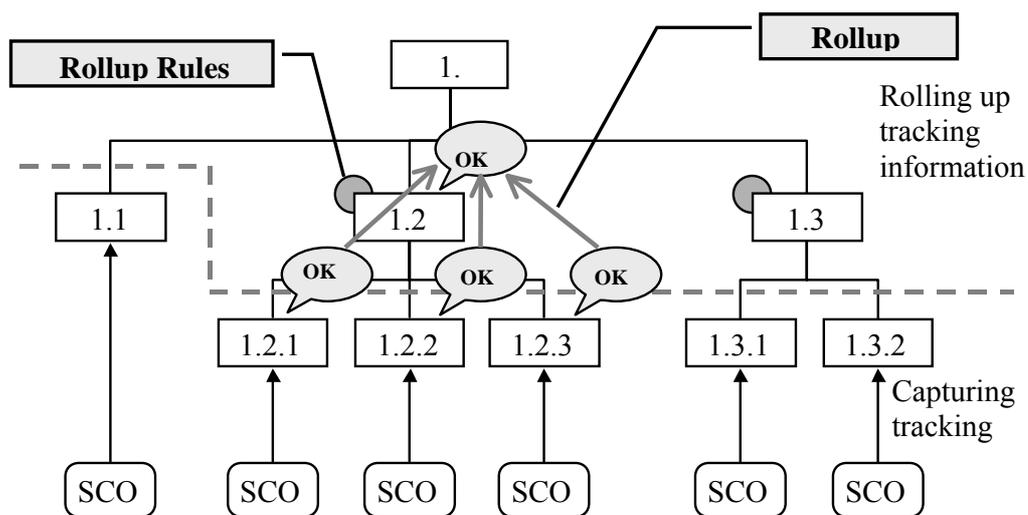
Table 6.4 Action Used in an Exit Rule

Action	Description
<i>Exit</i>	This action is used to end the activity

6.1.4 Specifying Rollup Rules

(1) Overview

The tracking information is derived from leaf activities. To derive or update the tracking information for each cluster or the entire activity tree, it is necessary to roll up the tracking information of the associated child activities to the parent activities. A mechanism for implementing this process is called rollup, and the rules applied to this process are called rollup rules.



OK: Completed

Figure 6.6 Rollup Rules

There are three types of rollup rule that can be defined: those for the objective satisfied status, those for the objective normalized measure and those for the attempt completion status.

(2) Specification Method

Specify the `<imsss:rollupRules>` element in the manifest file for each cluster.

(3) Component Element Specification

a. Rollup Rules

Rollup rules are defined using the following rule structure.

If [*condition_set*] **For** [*child activity set*] **Then** [*action*]

A rollup rule consists of a child activity set, a set of conditions (the condition set) to be evaluated for each activity included in the child activity set, and the action that sets the tracking information for the cluster if the rollup conditions are met.

b. Condition Set

A condition set describes a set of conditions that need to be satisfied for the execution of the action part. The condition set is a logical expression that is evaluated as *True* or *False* depending on the values of tracking information elements such as the objective measure and the progress status for the activity. A condition set may contain multiple conditions.

A condition combination indicates how the component conditions are to be combined in evaluation. The *All* condition combination indicates that the result of all the component conditions must be *True* for the condition set to be evaluated as *True*, and the *Any* condition combination indicates that any one of the component conditions must be evaluated as *True* for the condition set to be evaluated as *True*. The default condition combination is *Any*.

- *all*: Set to *True* if all conditions are *True*.
- *any*: Set to *True* if any condition is *True*.

A condition set and a condition combination are specified in the `<imsss:ruleConditions>` element.

A condition operator is a unary logical operator that is applied after the evaluation of the rule condition.

- *noOp*: The boolean value of the associated condition is not changed and is used as is in rule evaluation.
- *not*: The associated condition is negated in the rule evaluation.

The default condition operator is *noOp*.

The condition elements represent the elements whose values are set to *True* or *False* depending on the tracking information for the activity. Condition elements and condition operators are specified in the `<imsss:rollupCondition>` element.

The condition elements that can be used in a condition set are shown in Table 6.5.

Table 6.5 Condition Elements for Rollup Rules

Condition Element	Tracking Information	Description
<i>Satisfied</i>	<i>Objective Satisfied Status</i>	Evaluated as <i>True</i> if the <i>Rollup Objective Satisfied Status</i> is “satisfied”
<i>Objective Status Known</i>	<i>Objective Satisfied Status</i>	Evaluated as <i>True</i> unless the <i>Rollup Objective Satisfied Status</i> is “unknown”
<i>Objective Measure Known</i>	<i>Objective Normalized Measure</i>	Evaluated as <i>True</i> if the <i>Rollup Objective Normalized Measure</i> is not “unknown”
<i>Completed</i>	<i>Attempt Completion Status</i>	Evaluated as <i>True</i> if the <i>Attempt Completion Status</i> is “completed”
<i>Activity Progress Known</i>	<i>Attempt Completion Status</i>	Evaluated as <i>True</i> unless the <i>Attempt Completion Status</i> is “unknown”
<i>Attempted</i>	<i>Activity Attempt Count</i>	Evaluated as <i>True</i> if the <i>Activity Attempt Count</i> is 1 or more
<i>Attempt Limit Exceeded</i>	<i>Activity Attempt Count</i>	Evaluated as <i>True</i> if the <i>Activity Attempt Count</i> is greater than the value specified in the limit condition
<i>Never</i>	None	Always evaluated as <i>False</i>

c. Child Activity Set

A child activity set is used to define how the results of applying the specified conditions to each child activity included in the child activity set are used to determine whether the final evaluation result should be set to *True* or *False*. For example, a child activity set can be used to define a situation as follows: if 80% of the child activities satisfy the specified condition set after the condition set has been applied to each child activity, the final evaluation result becomes *True*.

Table 6.6 List of Child Activity Sets

Name	Description
<i>All</i>	If the result for all child activities is <i>True</i> , the final result is considered <i>True</i>
<i>Any</i>	If the result for any child activity is <i>True</i> , the final result is considered <i>True</i>
<i>None</i>	If none of the child activities satisfies the condition, the final result is considered <i>True</i>
<i>At Least Count</i>	If the result for at least the specified number of child activities is <i>True</i> , the final result is considered <i>True</i>
<i>At Least Percent</i>	If the result for at least the specified percent of child activities is <i>True</i> , the final result is considered <i>True</i>

When *At Least Count* is specified, a minimum number of activities must be specified using the *minimumCount* attribute.

When *At Least Percent* is specified, a number to indicate the minimum percentage value must be specified using the *minimumPercent* attribute.

The child activity set is defined in the `<imsss:rollupRule>` element.

d. Action

If the specified rollup rules are evaluated as *True*, the status information of the parent is updated.

Table 6.7 List of Actions Used in Rollup Rules

Action	Description
<i>Satisfied</i>	The <i>Objective Satisfied Status</i> value for the parent is set to “satisfied”
<i>Not Satisfied</i>	The <i>Objective Satisfied Status</i> value for the parent is set to “not satisfied”
<i>Completed</i>	The <i>Attempt Completion Status</i> value for the parent is set to “completed”
<i>Incomplete</i>	The <i>Attempt Completion Status</i> value for the parent is set to “incomplete”

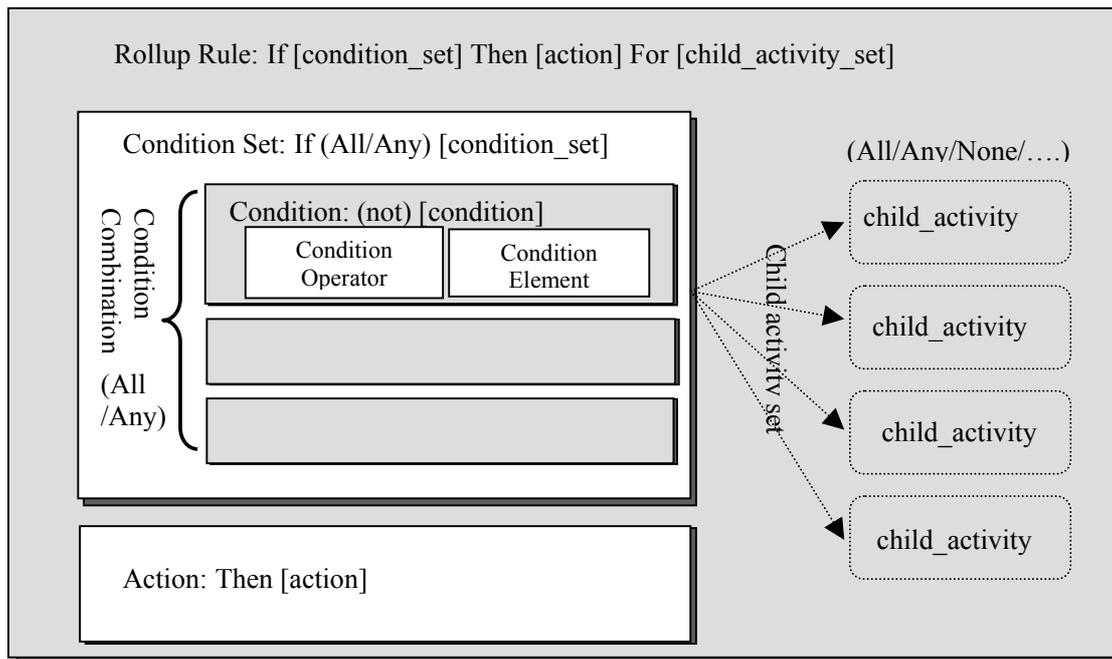


Figure 6.7 Rollup Rule Description

(4) Coding Example

The coding example below shows how to describe a rollup rule, *If “attempted” For “All” Then “completed”*, which means that if all the child activities have been attempted the parent activity is regarded as completed.

```

<item identifier="SCORM1">
  <title>SCORM Handbook</title>
  <item identifier=" CHAPTER1">
    <title>SCORM Overview</title>
    <item identifier="C1" isvisible = "true"
      identifierref="RC1">
      <title>What is SCORM?</title>
    </item>
    <item identifier="C2" isvisible = "true"
      identifierref="RC2">
      <title>History of SCORM</title>
    </item>
    <item identifier="C3" isvisible = "true"
      identifierref="RC3">
      <title>SCORM Basics</title>
    </item>
    <imsss:sequencing>
      <imsss:rollupRules >
        <imsss:rollupRule childActivitySet = "all">
          <imsss:rollupConditions>
            <imsss:rollupCondition
              condition = "attempted"/>
          </imsss:rollupConditions>
          <imsss:rollupAction action = "completed"/>
        </imsss:rollupRule>
      </imsss:rollupRules>
    </imsss:sequencing>
  </item>
</item>

```

(5) Note for Implementation

A child activity set that is subject to a rollup rule generally includes all the child activities in the cluster, but some child activities are not included in the rollup rule application as follows:

- Activities for which the *Tracked* element of the *deliveryControls* element is defined as *False*. As no tracking information is derived for these activities, they do not contribute to the rollup of their parents.
- Activities for which the *Rollup Objective Satisfied* element is defined as *False*. These activities are not subject to rollup rules which have the *Satisfied* or *Not satisfied* action.
- Activities for which the *Rollup Progress Completion* element is defined as *False*. These activities are not subject to rollup rules which have the *Completed* or *Incomplete* action.
- Activities that do not satisfy the condition specified in the *Required For* element of the parent activity definition. These activities are not considered in the rollup of their parents.

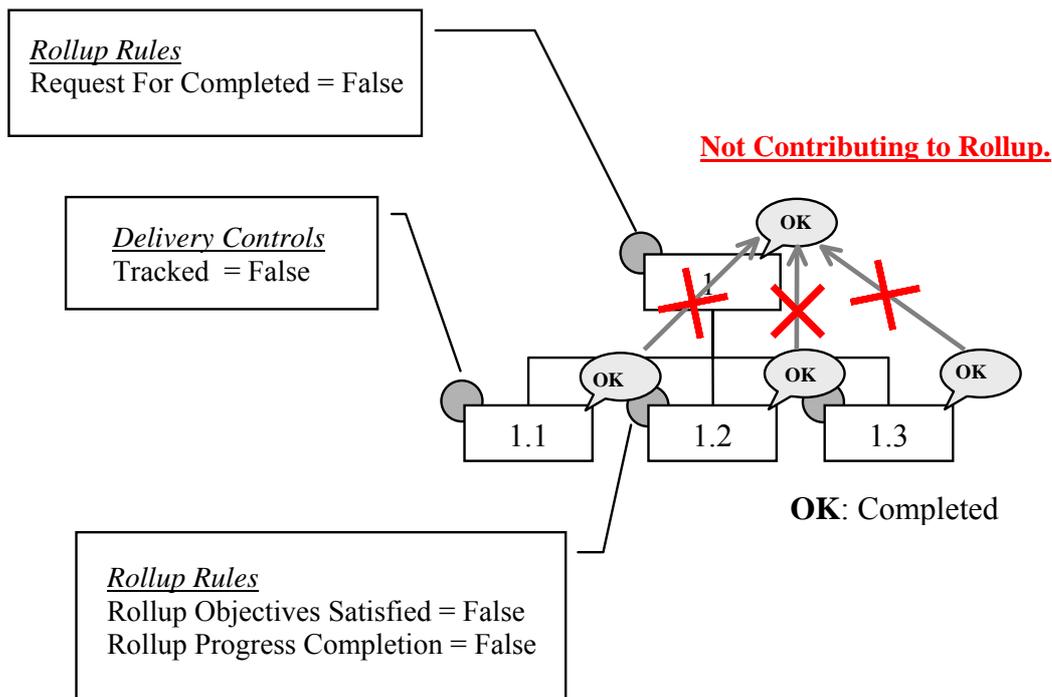


Figure 6.8 Rollup

6.1.5 Specifying Limit Conditions

(1) Overview

Limit conditions can be defined that describe conditions under which delivery of an activity is not allowed. In the current standard, for example, the number of attempts can be limited. If the *Limit Condition Attempt Limit* element is defined for an activity, the learner is not allowed to attempt that activity for more than the specified limit value.

(2) Specification Method

Specify the `<imsss:limitConditions>` element in the manifest file for each cluster.

(3) Description of Parameters

a. Attempt Limit

The *attemptLimit* attribute is used to assign a limit on the number of allowed attempts for each activity. The default value is 0. If the attempt limit value of an activity is 0, that activity is not accessible.

b. Attempt Absolute Duration Limit

This attribute is used to limit the maximum amount of time allowed for an activity. However, SCORM does not specify the behavior of the LMS and SCOs for this limit definition. The only requirement for the LMS is that it should implement the *cmi.max_time_allowed* element of the RTE data model, and initialize its value with the value specified in this attribute. (It is the responsibility of the SCO to give the actual allowed time limit.)

The limit value is specified with a numeric value in the *attemptAbsoluteDurationLimit* attribute, and the default value is 0.0.

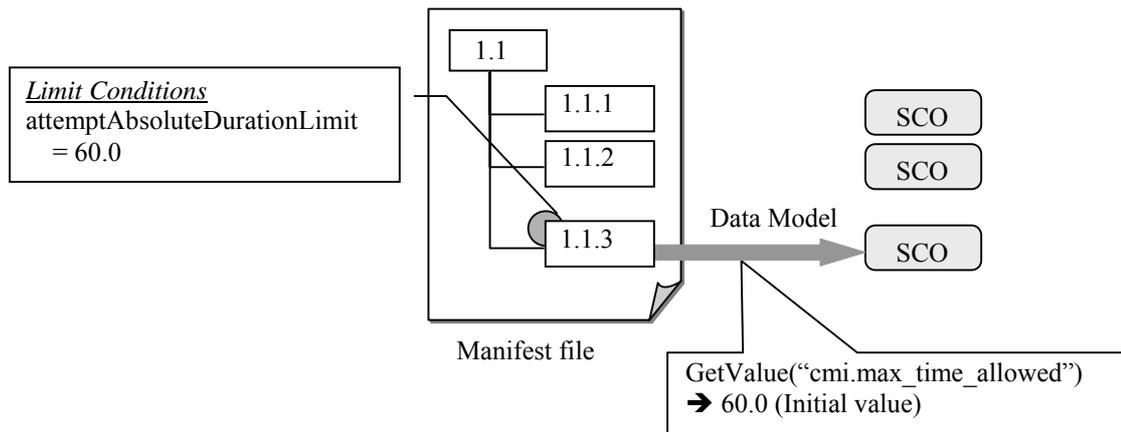


Figure 6.9 Limit Condition Description

(4) Coding Examples

Example 1: Limit the attempt count to 1.

```
<item identifier="INTRO" identifierref="RINTRO">
  <title>A Course Outline</title>
  <imsss:sequencing>
    <imsss:limitConditions attemptLimit="1"/>
  </imsss:sequencing>
</item>
```

Example2: Limit the amount of time allowed for one attempt to 2 minutes.

```
<item identifier="POST" identifierref="RPOST">
  <title>A Course on SCORM2004</title>
  <imsss:sequencing>
    <imsss:limitConditions attemptAbsoluteDurationLimit ="120.0"/>
  </imsss:sequencing>
</item>
```

6.1.6 Specifying Objectives

At least one objective should be assigned to each activity. A content developer may assign any number of additional objectives to the activity as required. The scope of these objectives is local to the activity for which they are defined and they are local objectives. However, shared global objectives can be defined in association with local objectives. These global objectives can be shared by activities for sequencing. For example, a pretest activity and a tutorial activity may have a shared global objective to make it possible to determine whether the tutorial activity is to be delivered depending on the result of the pretest activity.

The main points concerning objective descriptions can be summarized as follows:

- (1) More than one objective can be defined for one activity.

- (2) In principle, objectives are local to the activity for which they are defined.
- (3) Other activities can refer to an activity's objective information by defining a shared global objective.

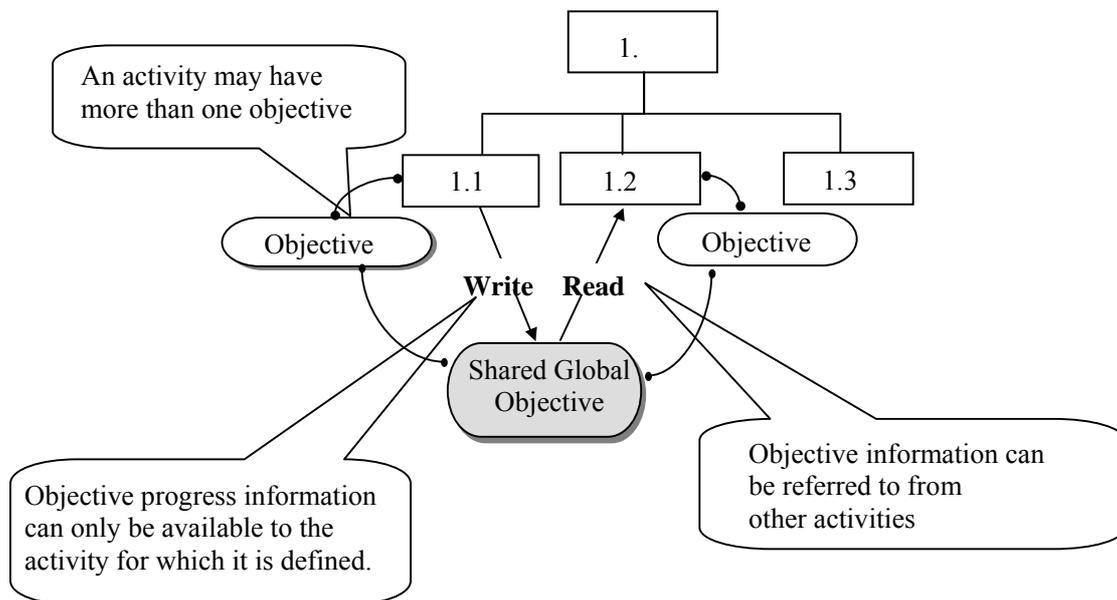


Figure 6.10 Objective Description

6.1.6.1 Objectives

(1) Overview

An activity has one local objective by default. However, a content developer may assign any number of local objectives for an activity as required.

(2) Specification Method

Objectives are defined by using the `<imsss:objectives>` element for each activity in the manifest file. The default objective is defined by using `<imsss:primaryObjective>` and additional objectives are defined by using `<imsss:objective>`.

(3) Description of Parameters

a. Objective ID

This attribute defines an identifier of an objective associated with an activity. It should be unique within the scope where the objective is referred to.

b. Objective Satisfied by Measure

This attribute indicates whether the satisfied status of the objective should be determined by the *Minimum Normalized Measure for the Objective*. The value is described with a boolean value: “true” or “false”. The default value is “false”.

c. Objective Minimum Satisfied Normalized Measure

This element indicates the minimum satisfaction measure for the objective. This value is used as the threshold value against which the objective's normalized measure is evaluated. The measure is defined with a normalized value between -1 and 1. The default value is 1.0.

d. Objective Contributes to Rollup⁵

This attribute indicates whether the objective contributes to the rollup of its parent. The value is described with a boolean value: “true” or “false”. The default value is “false”

(4) Coding Example

This coding example shows a pretest with 80 points set as the passing score.

```
<item identifier="PRETEST1">
  <title>Pre-test</title>
  <imsss:sequencing>
    <imsss:objectives>
      <imsss:primaryObjective objectiveID = "PRIMARYOBJ"
        satisfiedByMeasure = "true">
        <imsss:minNormalizedMeasure>0.8</imsss:minNormalizedMeasure>
      </imsss:primaryObjective>
    </imsss:objectives>
  </imsss:sequencing>
</item>
```

(5) Note for Implementation

If the *Objective Satisfied by Measure* element is set to “True” the value of the SCORM Run-Time Environment Data Model element (*cmi.scaled_passing_score*) is initialized with the value set for the *Objective Minimum Satisfied Normalized Measure*. This is the threshold value against which the objective’s satisfaction is evaluated.

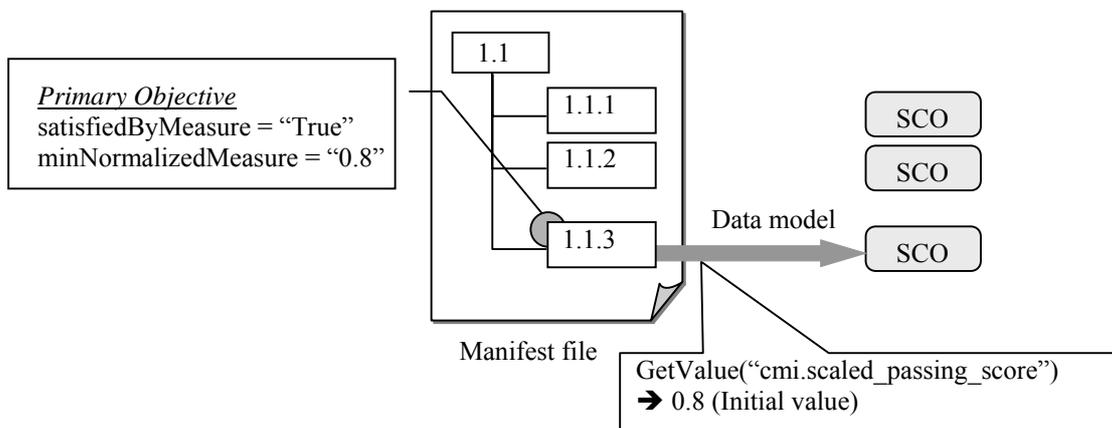


Figure 6.11 Defining Threshold Point for Satisfaction Evaluation

An SCO sets the values for its objective information using the RTE Data Model elements (*cmi.objectives*).

⁵ The SCORM CAM book specifies that only the objective defined in the <primaryObjective> element contributes to the rollup of the parent activity. This means that the primary objective defined using the primaryObjective element is the only objective that contributes to the rollup and the value of this attribute is regarded as *True* for this objective.

- *cmi.objectives.n.success_status* → Objective Satisfied Status
- *cmi.objectives.n.completion_status* → Attempt Completion Status

To allow an SCO to set values to multiple local objectives, a collection of data model elements is defined, and values are set to the data model elements corresponding to each objective.

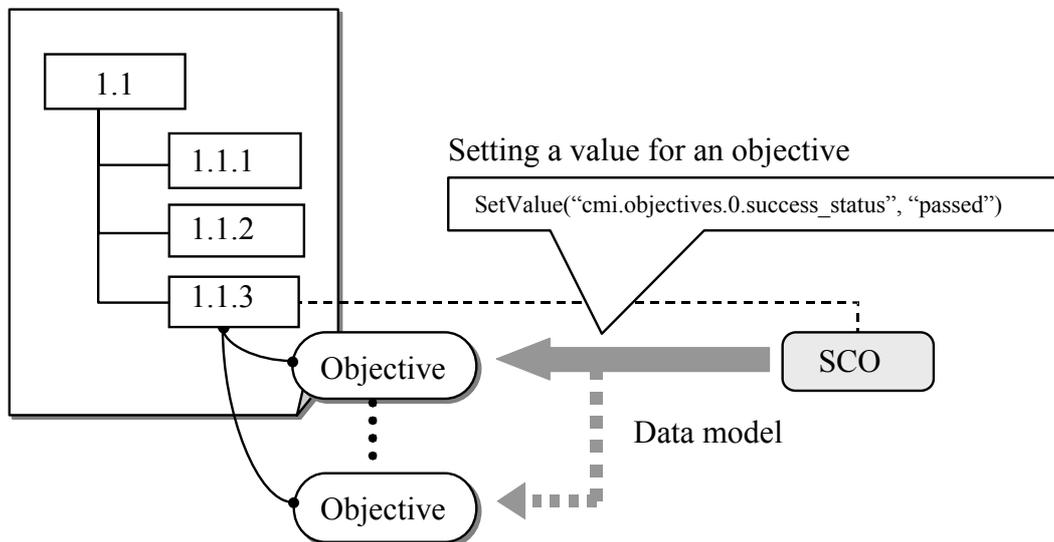


Figure 6.12 Setting Objective Data

6.1.6.2 Specifying Shared Global Objectives

(1) Overview

Each local objective may be connected to a shared global objective. In this way, the tracking information may be shared between activities for sequencing. A shared global objective is associated with the local objectives of some activities so that the associated activities can refer to the shared global objective under the following conditions:

- One local objective can be associated with one and only one shared global objective
- One shared global objective can be associated with multiple local objectives
- One activity can refer to multiple shared global objectives via multiple local objectives
- A shared global objective can be updated through only one local objective.

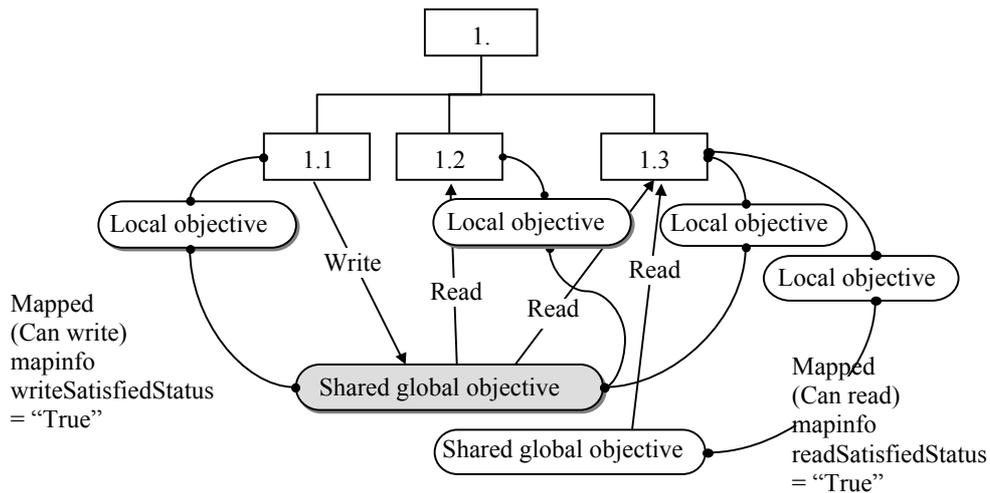


Figure 6.13 Shared Global Objectives

(2) Specification Method

The `<imsss:mapInfo>` element is used to specify the relationships between objectives within each objective description (`<imsss:primaryObjective>` and `<imsss:objective>`).

[Explanation of Parameters]

a. Activity Objective ID

This indicates an identifier of a local objective associated with the activity. As there is no default defined, this must be specified with a unique value.

b. Target Objective ID

This indicates an identifier of a shared global objective that is the target for mapping. As there is no default defined, this must be specified with a unique value.

c. Read Objective Satisfied Status

This indicates whether the *Objective Satisfied Status* should be referred to and should be specified with a boolean value. The default value is "True".

d. Write Objective Satisfied Status

This indicates whether the *Objective Satisfied Status* should be set and should be specified with a boolean value. The default value is "False".

e. Read Objective Normalized Measure

This indicates whether the *Objective Normalized Measure* should be referred to and should be specified with a boolean value. The default value is "True".

f. Write Objective Normalized Measure

This indicates whether the *Objective Normalized Measure* should be set and should be specified with a boolean value. The default value is "False".

(3) Coding Example

This coding example shows how to define a shared global object that can be written to by the associated activity.

```

<item identifier="PRETEST1">
  <title>Pre-test</title>
  <item identifier="Q1" isVisible = "false" identifierref="RQ1">
    <title>Question 1</title>
  </item>
  <imsss:sequencing>
    <imsss:objectives>
      <imsss:primaryObjective
        objectiveID = "PRIMARYOBJ"
        satisfiedByMeasure = "true">
        <imsss:minNormalizedMeasure>0.6</imsss:minNormalizedMeasure>
        <imsss:mapInfo targetObjectiveID = "obj1"
          readNormalizedMeasure = "false"
          writeSatisfiedStatus = "true" />
      </imsss:primaryObjective>
    </imsss:objectives>
  </imsss:sequencing>
</item>

```

(4) Note for Implementation

Activity objective IDs and target objective IDs are uniquely defined within the scope where they are applied. However, it is desirable to make them unique also within the associated content package to identify tracking information.

6.1.7 Specifying Other Controls

6.1.7.1 Selection and Randomization Controls

(1) Overview

Selection and randomization controls are used to define how child activities in a cluster are selected and how they are re-ordered before they are delivered to the learner. These controls are classified into two behaviors.

- **Selection controls:** These controls are used to define how child activities are selected. It is possible to define the timing for selection and the number of activities to be selected.
- **Randomization controls:** These controls are used to control how activities are re-ordered. It is possible to define whether the target child activities are to be re-ordered and the timing for that action.

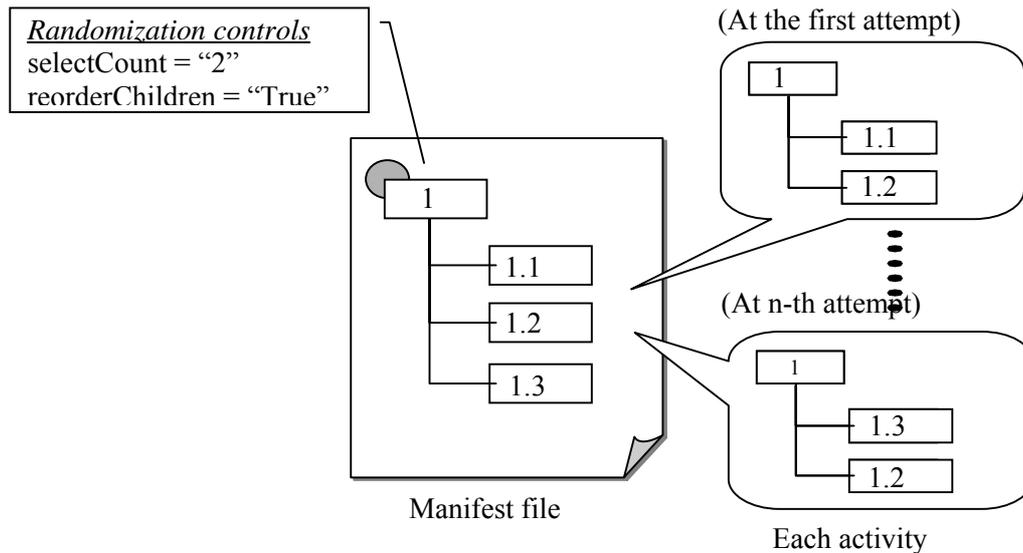


Figure 6.14 Selection and Randomization Controls

(2) Specification Method

The selection and randomization controls are defined using the `<imsss:randomizationControls>` element in the manifest file.

(3) Explanation of Parameters

a. Select Count

This attribute indicates the number of child activities that must be selected from the associated cluster for the selection and randomization. The value is specified with an integer value. The default value is 0. When the *selectCount* value is 0, no activity is selected. If the value exceeds the number of all associated child activities, all the child activities are selected.

b. Selection Timing

This attribute is used to define when the selection should occur. The value should be one of the following tokens:

Table 6.9 List of Tokens for Selection Timing

Token	Description
<i>never</i>	Selection does not occur. All child activities are regarded as valid ones.
<i>once</i>	Selection occurs at the first attempt on the activity
<i>onEachNewAttempt</i> ⁶	Selection occurs every time a new attempt is made on the activity

Default: “*never*”

c. Reorder Children

This attribute is used to indicate whether the child activities should be reordered and should be specified with a boolean value. The default value is “False”.

d. Randomization Timing

⁶ The current SCORM standard does not specify the *onEachNewAttempt* attribute and its behavior.

This attribute is used to define when the randomization should occur. The value should be one of the following tokens:

Table 6.9 List of Tokens for Randomization Timing

Token	Description
<i>never</i>	Randomization does not occur
<i>once</i>	Randomization occurs at the first attempt on the activity
<i>onEachNewAttempt</i>	Randomization occurs ever time a new attempt is made on the activity

Default: “*never*”

(4) Coding Example

This coding example shows how to define a case where two child activities are to be selected from among three child activities.

```

<item identifier="POSTTEST1">
  <title>PostTest</title>
  <item identifier="Q1" isVisible = "false" identifierref="RQ1">
    <title>Question 1</title>
  </item>
  <item identifier="Q2" isVisible = "false" identifierref="RQ2">
    <title>Question 2</title>
  </item>
  <item identifier="Q3" isVisible = "false" identifierref="RQ3">
    <title>Question 3</title>
  </item>
  <imsss:sequencing>
    <imsss:randomizationControls selectCount="2"
                                selectionTiming="once" />
  </imsss:sequencing>
</item>

```

6.1.7.2 Delivery Controls

(1) Overview

Delivery controls indicate actions that should be taken prior to an attempt on an activity. These controls are used for the management of tracking information on the content side that is expected to be used by LMSs.

(2) Specification Method

Delivery controls are defined using the `<imsss:deliveryControls>` element in the Manifest file for each activity.

(3) Explanation of Parameters

a. Tracked

This element indicates whether the tracking information that includes the information about the activity’s objective progress, attempts and completion should be tracked. The element contains a boolean value. The default value is “True”.

b. Completion Set By Content

This element indicates whether the *Attempt Completion Status* is to be set by the SCO or not. The element contains a boolean value. If the value is set to “True”, the attempt completion status of the activity cannot be changed by the LMS, and can only be updated by the content (the SCO). The default value is “False”.

c. Objective Set By Content

This element indicates whether the *Objective Satisfied Status* is to be set by the SCO or not. The element contains a boolean value. If the value is set to “True”, the objective’s satisfaction status for the activity cannot be changed by the LMS, and can only be updated by the content (the SCO). The default value is “False”.

(4) Coding Example

In this example, tracking information is not tracked for the activity.

```
<item identifier="L1" identifierref="RL1">
  <title>What is a Learning Objective?</title>
  <imsss:sequencing>
    <imsss:deliveryControls tracked = "false"/>
  </imsss:sequencing>
</item>
```

6.2 Features of SCORM 2004 SCOs

This section discusses the major points to note when implementing SCORM 2004 conformant SCOs.

6.2.1 RTE Usage Examples

6.2.1.1 Implementing an API Instance

When launched, an SCO must be able to find the LMS-provided API Instance to establish a communication session with the LMS. To find the API Instance, the SCO must search the following locations for the API Instance:

- The chain of parents of the current window until the top parent is reached
- The opener window, which is the window that has opened the SCO’s window
- The chain of parents of the opener window until the top parent is reached.

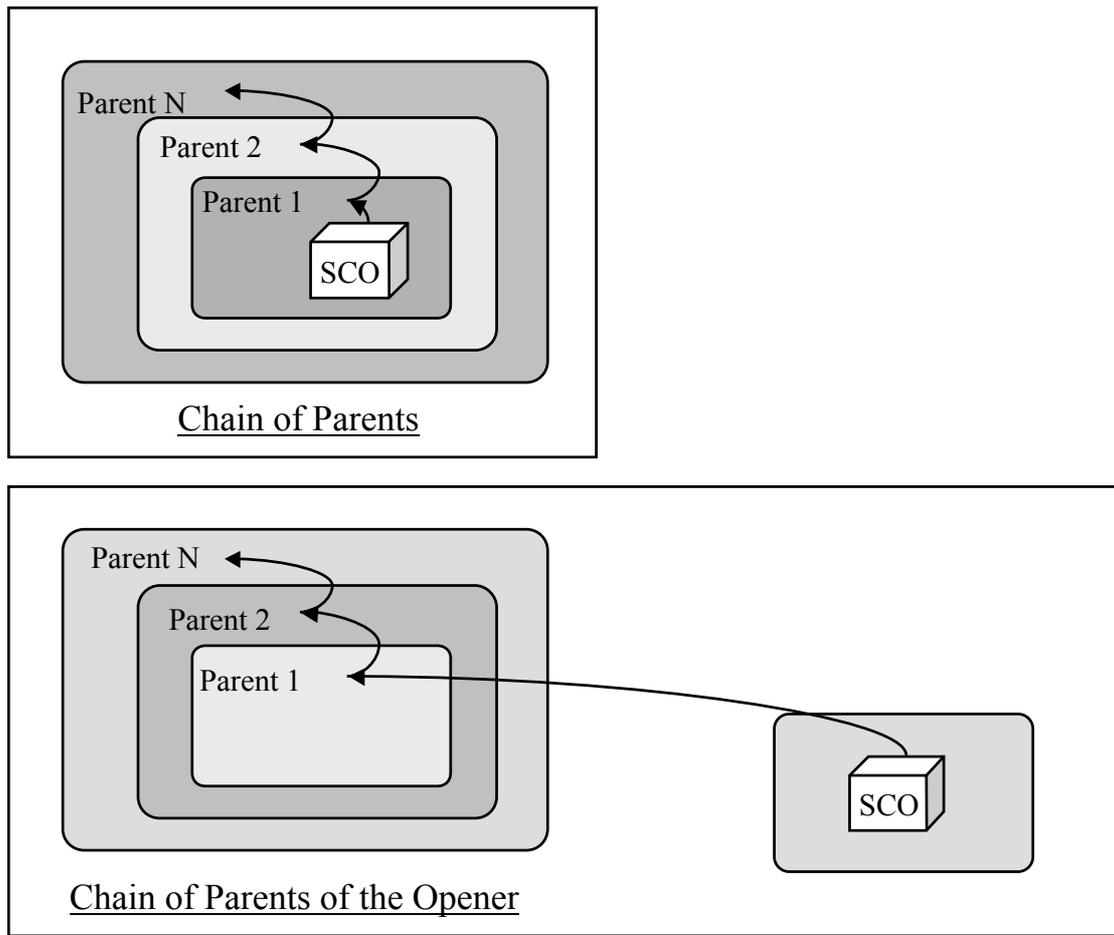


Figure 6.15 Finding the API Instance

The SCO must also be able to call the *Initialize*("") and *Terminate*("") API methods once it has found the API Instance. The IEEE standard provides a simple ECMAScript (JavaScript) coding example that shows how to find the API Instance. As the standard does not require use of ECMAScript, however, other coding methods can be used.

The code below shows an example of how to apply recursive searching for the API Instance.

```

<html>
<head>
<script type="text/javascript">
<!--
  //----- Find the API Instance -----
  var API = null;
  function FindAPI(win) {
    if ((typeof(win.API_1484_11) != "undefined") &&
        (win.API_1484_11 != null)) {
      return win.API_1484_11;
    } else if (win.location == top.location) {
      return null;
    } else {
      return FindAPI(win.parent);
    }
  }
  function MyInit() {
    // Locate the API frame
    if ((window.parent != null) && (window.parent != window)) {
      API = FindAPI(window.parent);
    }
    // To find the API Instance on the window opener
    if ((API == null) && (window.opener != null)) {
      API = FindAPI(window.opener);
    }
    if (API != null) {
      // Call the Initialize API method
      API.Initialize("");
    } else {
      alert("Cannot find the API Instance.");
    }
  }
  function MyFin() {
    if (API != null) {
      // Call the Terminate API method
      API.Terminate("");
    }
  }
  //-->
</script>
</head>
<body onload="MyInit();" onunload="MyFin();">
<h1>SCORM Sample Code</h1>
</body>
</html>

```

6.2.1.2 Calling API Methods

This section discusses how to call API methods that are required when developing SCORM compliant content.

(1) Using API Methods

a. Initialize

This method is used for an SCO to establish a communication session with the LMS after having found the API Instance located in the API frame. The SCORM standard specifies that the SCO must always make one *Initialize* method call.

b. Terminate

This method is used for an SCO to terminate the current communication session with the LMS. The SCORM standard specifies that the SCO must always use this API method with the *Initialize* method as a pair. The SCO should send the relevant data at the same time as it terminates its communication session with the LMS.

c. GetValue

The *GetValue* method is used when the SCO retrieves data from the LMS. It is possible for the SCO to use the obtained values after assigning them to variables within the SCO's scope.

d. SetValue and Commit

The *SetValue* method is used to set the data that will be sent to the LMS (after being cached by the API Instance). The *Commit* method is used to forward to the persistent data store of the LMS any data that has been cached by the API Instance. To ensure that the data set with the *SetValue* method calls is saved by the LMS in a persistent store, the *Commit* or *Terminate* method must be called.

Example 1: This example shows how the SCO retrieves the learner's name from the LMS and displays it on the screen.

```

<html>
<head>
<script type="text/javascript">
<!--
  //----- Find the API Instance. -----
  var API = null;
  function FindAPI(win) {
    // (Omitted)
  }
  function MyInit() {
    // Locate the API frame.
    if ((window.parent != null) && (window.parent != window)) {
      API = FindAPI(window.parent);
    }
    // Locate the opener window or API frame.
    if ((API == null) && (window.opener != null)) {
      API = FindAPI(window.opener);
    }
    if (API != null) {
      // Call the Initialize API method.
      API.Initialize("");
    } else {
      alert("Cannot find the API Instance.");
    }
  }
  function MyFin() {
    if (API != null) {
      // Call the Terminate API method.
      API.Terminate("");
    }
  }
  function GetLearnerName() {
    var name = "";
    if (API != null) {
      // Get the learner's name.
      name = API.GetValue("cmi.learner_name");
    }
    return name;
  }
  //-->
</script>
</head>
<body onload="MyInit();" onunload="MyFin();">
  <h1>SCORM EXAMPLE 1</h1>
  <script type="text/javascript">
  <!--
  document.write("<p>Welcome! Dear " + GetLearnerName() + ".</p>");
  //--></script>
</body>
</html>

```

Example 2: This example shows how the SCO keeps track of the learner's answers and stores the result at the LMS.

```

<html>
<head>
<script type="text/javascript">
<!--
  //----- Find the API Instance.-----
  var API = null;
  function FindAPI(win) {
    // (Omitted)
  }
  function MyInit() {
    // Locate the API frame.
    if ((window.parent != null) && (window.parent != window)) {
      API = FindAPI(window.parent);
    }
    // Locate the opener window or the API Instance frame.
    if ((API == null) && (window.opener != null)) {
      API = FindAPI(window.opener);
    }
    if (API != null) {
      // Call the Initialize API method.
      API.Initialize("");
    } else {
      alert("Cannot find the API Instance.");
    }
  }
  function MyFin() {
    if (API != null) {
      // Call the Terminate API method.
      API.Terminate("");
    }
  }
  function SetAnswer() {
    var ans = document.form1.text1.value;
    if (API != null) {
      //Save the answer data.
      API.SetValue("cmi.interactions.0.learner_response",
        ans);
      //Commit the answer data to the LMS.
      API.Commit("");
    }
  }
  //-->
</script>
</head>
<body onload="MyInit();" onunload="MyFin();">
  <h1>SCORM Example 2</h1>
  <br>
  <form name="form1">
  <p><input name="answer1" type="text" size="20">
  <input type="button" value="Answer" onclick="SetAnswer();"></p>
  </form>
</body>
</html>

```

(2) Handling API Errors

When an error occurs during a state transition of the API Instance, an error code is set at the API Instance. SCOs must be implemented in such a way that they should be able to handle the possible error conditions for all API method calls.

The API Instance provides a set of support methods that the SCOs can call to determine the errors. The support methods include *GetLastError*, *GetErrorString* and *GetErrorDiagnostic*.

The following example shows how to handle errors.

```

<html>
<head>
<script type="text/javascript">
<!--
    var API = null;
    function FindAPI(win) {
        (Omitted)
    }
    function MyInit() {
        (Omitted)
    }
    function MyFin() {
        (Omitted)
    }
    //-----Check the error.-----
    function CheckError() {
        var errMsg = "";
        if (API != null) {
            if (parseInt(API.GetLastError()) > 0) {
                errMsg = API.GetErrorString() + ":" +
API.GetDiagnostic();
                alert(errMsg);
            }
        }
    }
    //-->
</script>
</head>
<body onload="MyInit();" onunload="MyFin()">
    <h1>SCORM Example</h1>
</body>
</html>

```

6.2.1.3 Using the RTE Data Model

The RTE Data Model is a collection of data model elements used for communication between the LMS and SCOs. When the LMS and the SCOs communicate with each other, both assume that the other party knows about each of these data model elements. The major uses of these data model elements in communication are

- Initialization
- Reading data
- Writing data
- Saving and storing

The usage patterns of data model elements are read-only, write-only, and read and write.

(1) Read-Only Data Model Elements

a. Examples

Example 1: Learner's ID

```
id = GetValue("cmi.learner_id");
```

Example 2: The launch data of the SCO

```
Lprm = GetValue("cmi.launch_data");
```

Example 3: Maximum attempt duration on the SCO

```
TimeAllowed = GetValue("cmi.max_time_allowed");
```

b. Behaviors

- The LMS performs initialization using the session data and the information concerning the content structure.
- SCOs utilize this data after retrieving it from the LMS.
- The SCOs cannot change or update this data.

c. Explanation

The learner's ID information in Example 1 is assumed to be initialized by the LMS using the session data.

The *cmi.launch_data* element in Example 2 and the *cmi.max_time_allowed* element in Example 3 are initialized by the LMS using the values defined in the manifest file (*imsmanifest.xml*).

Example 2: SCO launch data is initialized with the value defined in `<adlcp:dataFromLMS>`.

Example 3: The maximum attempt duration on the SCO is initialized with the value defined in `<imsss:attemptAbsoluteDurationLimit>`.

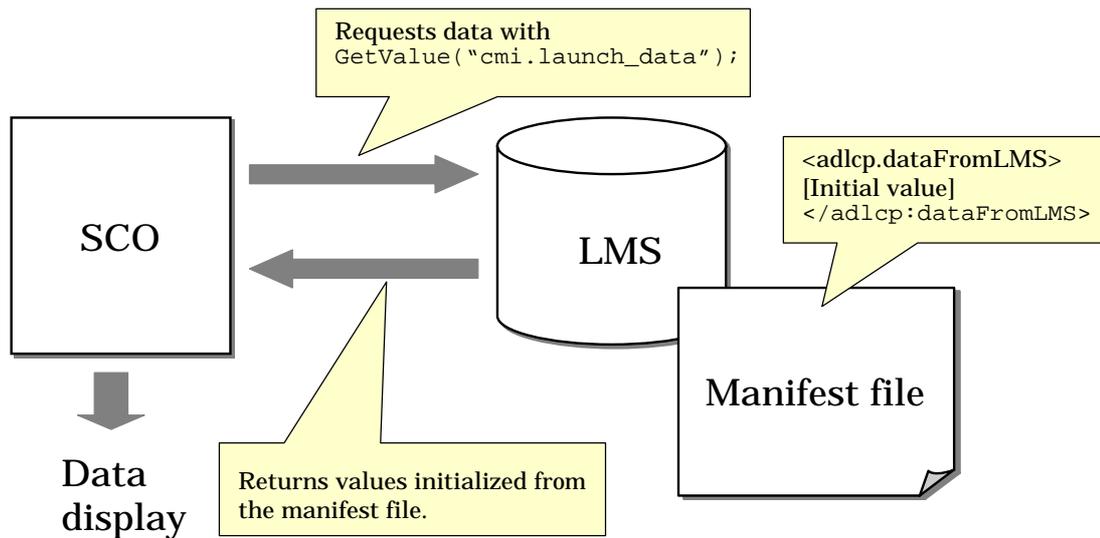


Figure 6.16 Using Data Model Elements (Read-only)

For these read-only data model elements, there are corresponding elements defined in the manifest file for their initial values.

Table 6.10 List of Data Model Elements Initialized from the Manifest File

Data Model Element Name	Description
Element Name in Manifest File	
<i>cmi.completion_threshold</i>	Indicates a value against which the learner's completion of the SCO is evaluated
<adlcp:completionThreshold>	
<i>cmi.launch_data</i>	Parameters that the SCO can use for initialization
<adlcp:dataFromLMS>	
<i>cmi.max_time_allowed</i>	Indicates the maximum amount of time allowed for an attempt on the SCO
<imsss:attemptAbsoluteDurationLimit>	
<i>cmi.scaled_passing_score</i>	Indicates a value against which the objective satisfaction measure is to be evaluated
<imsss:minNormalizedMeasure>	
<i>cmi.time_limit_action</i>	Indicates an action to be taken when the maximum allowed time has been reached
<adlcp:timeLimitAction>	

(2) Write-Only Data Model Elements

a. Example

Example 4: Session time

```
SetValue("cmi.session_time", "05:15:00");
```

b. Behaviors

- The LMS does not perform initialization.
- The SCO sets a value to the specified data model element.

- The LMS processes, saves and stores the information.

c. Explanation

- The data model elements are used mainly for storing the learner’s performance and progress data with an SCO.

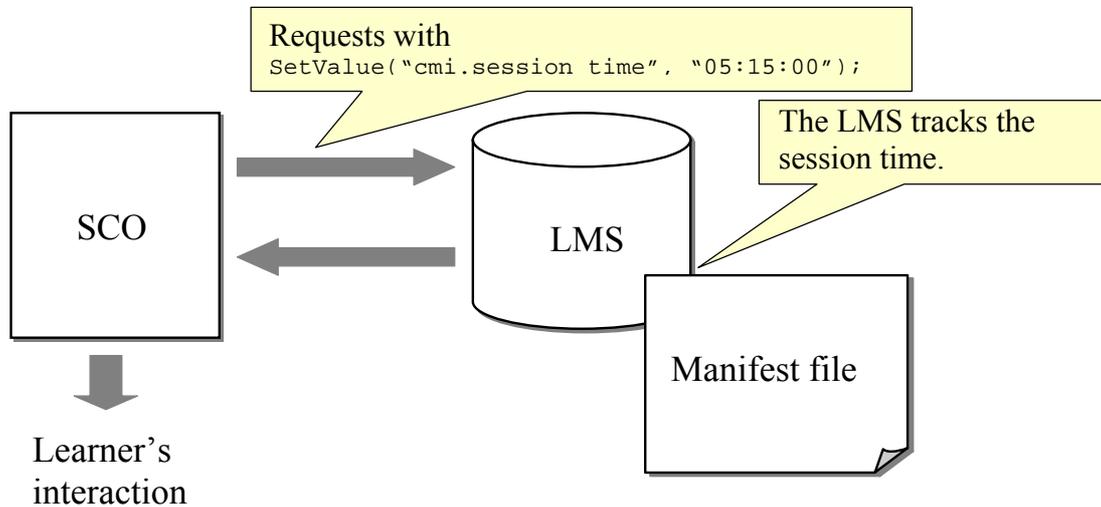


Figure 6.17 Using Data Model Elements (Write)

(3) Read and Write Data Model Elements

a. Examples

Example 5: Reading the learner’s success status.

```
Loc = GetValue("cmi.success_status");
```

Example 6: Writing the learner’s success status data.

```
SetValue("cmi.success_status", "passed");
```

b. Behaviors

- The LMS initializes the associated data model elements as “unknown”.
- The SCO uses the data after retrieving it and then updates it.
- The LMS processes, saves and stores the information.
- The SCO may use the data again after retrieving the updated data.

c. Explanation

- The data model elements for reading and writing are used to capture state changes during a session such as progress status, success status and performance measures.
- The SCO is supposed to initialize these data model elements.
- As state values and numerical values are used, errors are caused if the values are not within the defined set of keywords or within the allowed range.

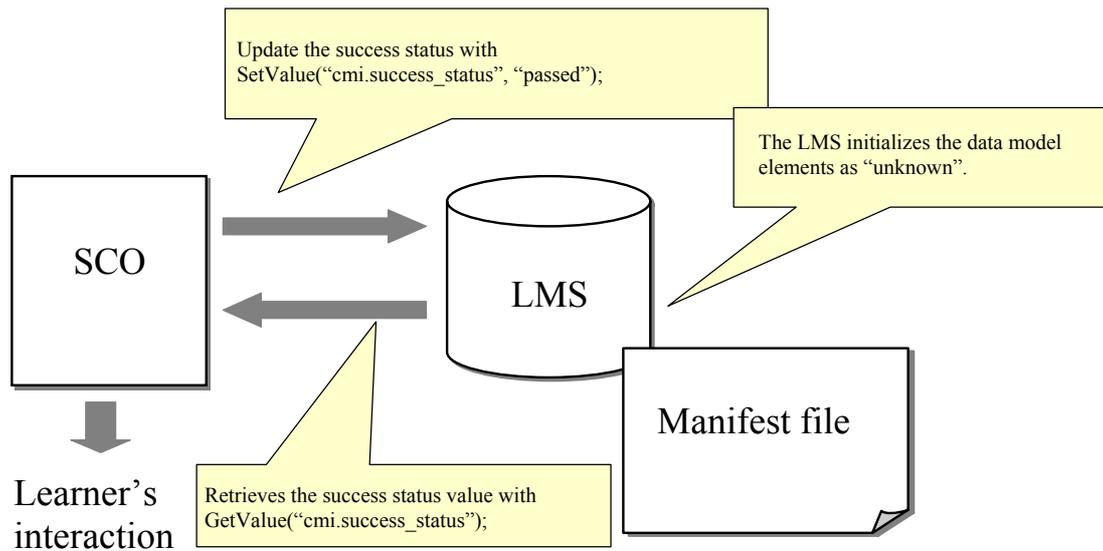


Figure 6.18 Using Data Model Elements (Read/Write)

6.2.2 Example Uses of Navigation

SCORM 2004 allows the navigation of SCOs to be controlled from within an SCO.

- A navigation request event can be triggered within an SCO for another SCO.
- It is now possible for an SCO to request that the LMS hide or show the LMS-provided user interface devices.

This section explains how to use this feature.

6.2.2.1 Navigation Requests from an SCO

(1) Overview

In addition to the LMS-provided user interface devices for navigation, SCORM 2004 allows an SCO to trigger such navigation request events as *Continue* and *Previous*.

(2) Specification Method

A navigation request event can be triggered from content (SCOs) using the *adl.nav.request* RTE data model element in a *SetValue* API method call.

(3) Explanation of Parameters

Table 6.11 shows a list of navigation control requests that can be used in the API method call.

Table 6.11 List of Navigation Requests from SCOs

Parameter	Description
<i>continue</i>	Terminate the current SCO and issue a <i>continue</i> navigation request
<i>previous</i>	Terminate the current SCO and issue a <i>previous</i> navigation request
<i>choice</i>	Terminate the current SCO, and issue a navigation request to move to a specified SCO

Parameter	Description
<i>exit</i>	Terminate the current SCO
<i>exit all</i>	Terminate the current activity tree
<i>abandon</i>	Abandon the current activity
<i>abandon all</i>	Abandon the current activity tree
<i>_none</i>	Clears unprocessed navigation requests

(4) Coding Example

Example 1: To trigger a *Continue* navigation event

```
SetValue("adl.nav.request", "continue");
```

Example 2: To trigger a *Choice* navigation event by selecting activity A1 as the target

```
SetValue("adl.nav.request", "{target='A1'}choice");
```

To trigger a *Choice* navigation event, the target activity to move to must be specified.

(5) Points to Note

- Navigation request events are not processed by the LMS until the *Terminate* API method call is successfully processed. This means that the *Terminate* method must be invoked after a *SetValue* method has been issued in order to trigger a navigation request event.
- Depending on the LMS implementation, some navigation requests made by SCOs might not be honored, so it is necessary for the SCOs to confirm the validity of the navigation requests they make and an appropriate measure for handling the requests that cannot be made must be implemented. SCOs should be able to check the validity of each request with the LMS through the *adl.nav.request_valid* RTE Data Model element.

6.2.2.2 Controlling LMS-Provided UI Devices from the Manifest File

(1) Overview

In SCORM 2004, whether the LMS-provided user interface devices are to be shown or hidden can be controlled.

(2) Specification Method

Whether the LMS-provided user interface devices are to be shown or hidden can be defined for each activity using the `<adlnav:presentation>` element in the manifest file.

(3) Explanation of Parameters

The LMS-provided user interface devices can be controlled with the `<hideLMSUI>` element definitions under the `<adlnav:presentation>` element using the tokens shown in Table 6.12.

Table 6.12 List of Tokens for Controlling LMS-provided UI Devices

Parameter	Description
<i>previous</i>	Hide the UI device for backward navigation
<i>continue</i>	Hide the UI device for forward navigation
<i>exit</i>	Hide the UI device for Exit
<i>abandon</i>	Hide the UI device for Abandon

(4) Coding Example

```

<organization>
  <item identifier="item1" identifierref="Resource1" isvisible="true">
    <adlnav:presentation>
      <adlnav:navigationInterface>
        <adlnav:hideLMSUI>continue</adlnav:hideLMSUI>
        <adlnav:hideLMSUI>previous</adlnav:hideLMSUI>
      </adlnav:navigationInterface>
    </adlnav:presentation>
  </item>
</organization>

```

7 Migration from SCORM 1.2 to SCORM 2004

Although SCORM 2004 is the successor standard of SCORM 1.2, it is not completely upwardly compatible with SCORM 1.2. To run SCORM 1.2 compliant content under a SCORM 2004 compliant LMS, the content must therefore be converted from SCORM 1.2 to the SCORM 2004 standard.

This section describes the differences between the two versions and explains how to convert content to the new standard.

7.1 Manifest File and SCOs

In SCORM 2004, the Run-Time Environment data model and content aggregation model have been upgraded from SCORM Versions 1.2 to 1.3.1. The sequencing and navigation features have also been added.

A SCORM content package consists of a manifest file and SCOs; the content packaging side is related to the manifest file, while the Run-Time Environment is closely related to the SCOs. As the content aggregation model and the Run-Time Environment have been upgraded in SCORM 2004, both the manifest file and SCOs developed in SCORM 1.2 must be converted to the SCORM 2004 standard and a set of sequencing rules must be defined in the manifest file.

7.2 Converting the Manifest File

In principle, SCORM 2004 compliant manifest files are upwardly compatible with SCORM 1.2 manifest files. In a SCORM 2004 environment, a SCORM 1.2 manifest file is regarded as a SCORM 2004 manifest file which does not contain any sequencing rule definitions.

The old manifest files must be changed, though, as some elements specified in SCORM 1.2 have been removed and changes have been made to content packaging. The major changes that must be made to the manifest file are as follows:

- Updated XML Binding
- Changes to ADL content packaging extension elements (*prerequisites* and *masteryscore*)
- Sequencing control mode specification
- Metadata description

These changes to the manifest file are explained below.

7.2.1 Basic Structure

The basic structure of the manifest file (*imsmanifest.xml*) is a hierarchical tree structure in both SCORM 1.2 and SCORM 2004, and this basic structure has not been changed. Thus, the *<metadata>*, *<organizations>* and *<resources>* elements which represent the basic content structure are the same in both versions.

7.2.2 Content Packaging

Due to the changes made to the IMS namespace declarations and metadata XML binding specifications that SCORM is based on, the content packaging specification has been changed in SCORM 2004. For this reason, it is necessary to modify various declarations in the manifest files developed under the earlier version. These changes are summarized in Table 7.1.

Table 7.1 Changes to Content Packaging

	SCORM 1.2	SCORM 2004
IMS namespace declaration	<i>http://www.imsproject.org/xsd/ims_cp_rootv1p1p2</i>	<i>http://www.imsproject.org/xsd/ims_cp_v1p1</i>
ADL namespace declaration	<i>http://www.adlnet.org/xsd/adlcp_rootv1p2</i>	<i>http://www.adlnet.org/xsd/adlcp_v1p3</i>
Metadata XML binding namespace	<i>http://www.imsglobal.org/xsd/imsmd_rootv1p2p1</i>	<i>http://ltsc.ieee.org/xsd/LOM</i>

In addition, changes have been made to some ADL content packaging extension elements, as shown in Table 7.2

Table 7.2 Changes to ADL Content Packaging Extension Elements

SCORM 1.2	SCORM 2004
<i><adlcp:prerequisites></i>	Removed (Use a sequencing rule definition)
<i><adlcp:maxtimeallowed></i>	Removed (Use a sequencing rule definition)
<i><adlcp:timelimitaction></i>	<i><adlcp:timeLimitAction></i>
<i><adlcp:datafromlms></i>	<i><adlcp:dataFromLMS></i>
<i><adlcp:masteryscore></i>	Removed (Use a sequencing rule definition)
<i><adlcp:scormtype></i>	<i><adlcp:scormType></i>

7.2.3 Changes to the Prerequisites and Masteryscore Elements

A particularly important point to note is that *<adlcp:prerequisites>* for defining prerequisites, *<adlcp:masteryscore>* for setting a threshold value for passing, and *<adlcp:maxtimeallowed>* for defining the maximum allowed time have been removed from the ADL content packaging extension elements and replaced with sequencing rule definitions.

As these ADL content packaging extension elements were treated inconsistently by different LMS implementations in SCORM 1.2, they caused interoperability problems.

These conditions can be described as sequencing rules in SCORM 2004. This enhancement enables improved interoperability and clearer definition of conditions.

This section explains how to convert the definitions of the *prerequisites* and *masteryscore* extension elements.

7.2.3.1 Converting Prerequisites Definitions

In SCORM 1.2, whether a particular activity (with an SCO or an asset) is valid for a learner was defined with the *prerequisites* ADL content packaging extension element.

As it is possible to define various behaviors of an activity through a set of sequencing rules in SCORM 2004, however, the delivery limits for an activity that had been defined in this *prerequisites* element can now be described in the sequencing rules. Therefore, the *prerequisites* extension element has been removed in SCORM 2004.

(1) Precondition Rules

There are a few different ways to control the delivery of activities; this section explains how to do it using precondition rules.

A condition for the delivery of an activity can be defined using precondition rules. Precondition rules are defined for each activity on the basis of the following structure:

If [*condition_set*] **Then** [*action*]

The condition set is a logical expression that is evaluated as *True* or *False* depending on the tracking information. The action is the limit on the delivery of the associated activity. The example below shows a case where the delivery of an activity is limited using a precondition rule.

Example:

If *not completed* **Then** *stopForwardTraversal*

This precondition rule indicates that if a learner has not completed the activity then the learner cannot move forward.

The above precondition rule is expressed in the manifest file as follows:

```
<item identifier = "SAMPLE1">
  <title>You Cannot Proceed.</title>
  <imsss:sequencing>
    <imsss:sequencingRules>
      <imsss:preConditionRule>
        <imsss:ruleConditions>
          <imsss:ruleCondition condition = "completed"
                                operator = "not"/>
        </imsss:ruleConditions>
        <imsss:ruleAction action = "stopForwardTraversal"/>
      </imsss:preConditionRule>
    </imsss:sequencingRules>
  </imsss:sequencing>
</item>
```

7.2.3.2 Converting Masteryscore Definitions

In SCORM 1.2, the threshold score for deciding pass or failure was defined using the *masteryscore* ADL content packaging extension element. In SCORM 2004 this

definition is described in the manifest file (imsmanifest.xml) as a set of sequencing rules that correspond to the definition

More specifically, the passing score for an SCO is defined with the `<imsss:objectives>` element. For an activity for which this condition is set, a passing score is defined by setting the `satisfiedByMeasure` attribute to `True` and then defining a threshold value in the `<imsss:minNormalizedMeasure>` element. This value is used to initialize the “`cmi.scaled_passing_score`” SCORM RTE Data Model element.

The example below shows how a passing score of 0.7 (70 points out of 100) is specified in the manifest file as the threshold value for the objective.

```
<item identifier="item1">
<title>Practical Exercise</title>
<imsss:sequencing>
  <imsss:objectives>
    <imsss:primaryObjective objectiveID="obj1"
                          satisfiedByMeasure="True">
      <imsss:minNormalizedMeasure>0.7</imsss:minNormalizedMeasure>
    </imsss:primaryObjective>
  </imsss:objectives>
</imsss:sequencing>
</item>
```

The passing value provided from the manifest file is used by the SCO after the SCO retrieves it from the Run-Time Environment data model element `cmi.scaled_passing_score`, which has been added in SCORM 2004.

For example, the code

```
masteryscore = GetValue("cmi.scaled_passing_score") * 100;
```

causes the SCO to retrieve the passing value that was initialized from the manifest file, multiply it by 100, and then save the result of “70” in the variable `masteryscore`.

7.2.4 Defining Sequencing Control Modes

7.2.4.1 Sequencing Control Modes

In SCORM 2004, the navigation behavior of a cluster is controlled using sequencing control modes. For example, to allow the learner to use the LMS-provided user interface devices for moving forward (*continue*) or backward (*previous*), the *Sequencing Control Flow* is defined as `True`; to allow the learner to select the next activity from the menu, the *Sequencing Control Choice* is defined as `True`.

If no sequencing control mode is defined, some LMS implementations may not deliver an activity while some may issue an error message.

Sequencing control modes are defined using the `<imsss:controlMode>` element for each cluster in the manifest file.

An example of the definition of a sequencing control mode is shown below:

```
<imsss:sequencing>
  <imsss:controlMode choice="true" choiceExit="true" flow="true"
    forwardOnly="false" />
</imsss:sequencing>
```

7.2.4.2 Attempt-Related Sequencing Control Modes

In SCORM 2004, it is possible to specify for a cluster whether to use information about the objective progress (*Use Current Attempt Objective Information*) and attempt progress (*Use Current Attempt Progress Information*) of the current attempt only or the latest information including the most recent previous attempt. The attributes to be defined in the *controlMode* element for these two are *useCurrentAttemptObjectiveInfo* and *useCurrentAttemptProgressInfo*, respectively.

The default values of the *Use Current Attempt Objective Information* and the *Use Current Attempt Progress Information* are *True*. If this control mode is omitted, only the current attempt information is used.

To use the latest information, including the most recent previous attempt, these control modes must be set to *False*.

7.2.5 Metadata Description

The Content Aggregation Metadata Application Profile has been changed in SCORM 2004, and the *<metadata>*, *<schema>* and *<schemaversion>* elements have each become mandatory.⁷ It is therefore necessary to check whether these elements are defined in the manifest file and define them if necessary.

The example below shows part of a manifest file in which these elements are defined:

```
<metadata>
  <schema>ADL SCORM</schema>
  <schemaversion>CAM 1.3</schemaversion>
</metadata>
```

⁷ As from SCORM 2004 3rd edition.

7.3 SCO Migration

To perform SCO migration from SCORM 1.2 to SCORM 2004, the following actions must be taken:

- Change the API Instance name
- Change API method names
- Change Data Model elements
- Accommodate error code changes

This section describes how to make these changes to convert SCOs to SCORM 2004.

7.3.1 API Instance Name Change

In SCORM 1.2, the name of the SCORM API Instance⁸ was “API”. This has been changed to “API_1484_11” in SCORM 2004.

When an SCO invokes the *FindAPI* function to find the API Instance, the parameter indicating the name of the API Instance must be changed to “API_1484_11”.

7.3.2 API Method Name Changes

The names of the API methods have been changed in SCORM 2004. Although there is no change to the syntax, it is necessary to change the names of these API methods.

The API methods have been renamed as follows:

Table 7.3 API Method Name Changes

SCORM 1.2	SCORM 2004
<i>LMSInitialize</i>	<i>Initialize</i>
<i>LMSFinish</i>	<i>Terminate</i>
<i>LMSGetValue</i>	<i>GetValue</i>
<i>LMSSetValue</i>	<i>SetValue</i>
<i>LMSCommit</i>	<i>Commit</i>
<i>LMSGetLastError</i>	<i>GetLastError</i>
<i>LMSGetErrorString</i>	<i>GetErrorString</i>
<i>LMSGetDiagnostic</i>	<i>GetDiagnostic</i>

7.3.3 Changes to Data Model Elements

In SCORM 2004, some RTE Data Model elements have been changed or added. To convert SCOs to SCORM 2004, it is therefore necessary to modify the SCOs to suit the SCORM 2004 RTE Data Model specification.

This section covers the major data model elements that require special attention when migrating from SCORM 1.2 to SCORM 2004.

⁸ It was called the API Adapter until SCORM 1.2.

7.3.3.1 Changed or Removed Data Model Elements

Some SCORM 1.2 data model elements have undergone a name change while others have been removed from the data model in SCORM 2004. To perform SCO migration, it is therefore necessary to modify the names of data model elements as required and replace deleted data model elements with new ones.

Table 7.4 shows the correspondence of data model elements between SCORM 1.2 and SCORM 2004.

Table 7.4 Correspondence of Data Model Elements between SCORM 1.2 and SCORM 2004

SCORM 1.2 Data Model	SCORM 2004 Data Model	Remarks
-	<i>cmi._version</i>	New
<i>cmi.comments</i>	<i>cmi.comments_from_learner._children</i> <i>cmi.comments_from_learner._count</i> <i>cmi.comments_from_learner.n.comment</i> <i>cmi.comments_from_learner.n.location</i> <i>cmi.comments_from_learner.n.timestamp</i>	Replaced
<i>cmi.comments_from_lms</i>	<i>cmi.comments_from_lms._children</i> <i>cmi.comments_from_lms._count</i> <i>cmi.comments_from_lms.n.comment</i> <i>cmi.comments_from_lms.n.location</i> <i>cmi.comments_from_lms.n.timestamp</i>	Replaced
<i>cmi.core._children</i>	-	Removed
<i>cmi.core.credit</i>	<i>cmi.credit</i>	Replaced
<i>cmi.core.entry</i>	<i>cmi.entry</i>	Replaced
<i>cmi.core.exit</i>	<i>cmi.exit</i>	Replaced
<i>cmi.core.lesson_location</i>	<i>cmi.location</i>	Replaced
<i>cmi.core.lesson_mode</i>	<i>cmi.mode</i>	Replaced
<i>cmi.core.lesson_status</i>	<i>cmi.success_status</i> <i>cmi.completion_status</i>	Replaced
<i>cmi.core.score._children</i>	<i>cmi.score._children</i>	Replaced
<i>cmi.core.score.max</i>	<i>cmi.score.max</i>	Replaced
<i>cmi.core.score.min</i>	<i>cmi.score.min</i>	Replaced
<i>cmi.core.score.raw</i>	<i>cmi.score.raw</i>	Replaced
-	<i>cmi.score.scaled</i>	New
<i>cmi.core.session_time</i>	<i>cmi.session_time</i>	Replaced
<i>cmi.core.student_id</i>	<i>cmi.learner_id</i>	Replaced
<i>cmi.core.student_name</i>	<i>cmi.learner_name</i>	Replaced
<i>cmi.core.total_time</i>	<i>cmi.total_time</i>	Replaced
<i>cmi.interactions._children</i>	<i>cmi.interactions._children</i>	
<i>cmi.interactions._count</i>	<i>cmi.interactions._count</i>	
<i>cmi.interactions.n.correct_responses._count</i>	<i>cmi.interactions.n.correct_responses._count</i>	
<i>cmi.interactions.n.correct_responses.n.pattern</i>	<i>cmi.interactions.n.correct_responses.n.pattern</i>	
-	<i>cmi.interactions.n.description</i>	New
<i>cmi.interactions.n.id</i>	<i>cmi.interactions.n.id</i>	
<i>cmi.interactions.n.latency</i>	<i>cmi.interactions.n.latency</i>	
<i>cmi.interactions.n.objectives._count</i>	<i>cmi.interactions.n.objectives._count</i>	
<i>cmi.interactions.n.objectives.n.id</i>	<i>cmi.interactions.n.objectives.n.id</i>	

SCORM 1.2 Data Model	SCORM 2004 Data Model	Remarks
<i>cmi.interactions.n.result</i>	<i>cmi.interactions.n.result</i>	
<i>cmi.interactions.n.student_response</i>	<i>cmi.interactions.n.learner_response</i>	Replaced
<i>cmi.interactions.n.time</i>	<i>cmi.interactions.n.timestamp</i>	Replaced
<i>cmi.interactions.n.type</i>	<i>cmi.interactions.n.type</i>	
<i>cmi.interactions.n.weighting</i>	<i>cmi.interactions.n.weighting</i>	
<i>cmi.launch_data</i>	<i>cmi.launch_data</i>	
<i>cmi.objectives._children</i>	<i>cmi.objectives._children</i>	
<i>cmi.objectives._count</i>	<i>cmi.objectives._count</i>	
-	<i>cmi.objectives.n.description</i>	New
<i>cmi.objectives.n.id</i>	<i>cmi.objectives.n.id</i>	
-	<i>cmi.objectives.n.progress_measure</i>	New
<i>cmi.objectives.n.score._children</i>	<i>cmi.objectives.n.score._children</i>	
<i>cmi.objectives.n.score.max</i>	<i>cmi.objectives.n.score.max</i>	
<i>cmi.objectives.n.score.min</i>	<i>cmi.objectives.n.score.min</i>	
<i>cmi.objectives.n.score.raw</i>	<i>cmi.objectives.n.score.raw</i>	
-	<i>cmi.objectives.n.score.scaled</i>	New
<i>cmi.objectives.n.status</i>	<i>cmi.objectives.n.success_status</i> <i>cmi.objectives.n.completion_status</i>	Replaced
<i>cmi.student_data._children</i>	-	Removed
<i>cmi.student_data.mastery_score</i>	<i>cmi.scaled_passing_score</i>	Replaced
<i>cmi.student_data.max_time_allowed</i>	<i>cmi.max_time_allowed</i>	Replaced
<i>cmi.student_data.time_limit_action</i>	<i>cmi.time_limit_action</i>	Replaced
<i>cmi.student_preference._children</i>	<i>cmi.learner_preference._children</i>	Replaced
<i>cmi.student_preference.audio</i>	<i>cmi.learner_preference.audio_level</i>	Replaced
<i>cmi.student_preference.language</i>	<i>cmi.learner_preference.language</i>	Replaced
<i>cmi.student_preference.speed</i>	<i>cmi.learner_preference.delivery_speed</i>	Replaced
<i>cmi.student_preference.text</i>	<i>cmi.learner_preference.audio_captioning</i>	Replaced
<i>cmi.suspend_data</i>	<i>cmi.suspend_data</i>	
-	<i>cmi.completion_threshold</i>	New
-	<i>cmi.progress_measure</i>	New

In addition, data model elements have been introduced in SCORM 2004 that enable SCOs to communicate navigation requests or check the validity of such requests.

Table 7.5 Data Model Elements for Navigation Requests and Validity Checking

SCORM 2004 Data Model Elements
<i>adl.nav.request</i>
<i>adl.nav.request_valid.previous</i>
<i>adl.nav.request_valid.continue</i>
<i>adl.nav.request_valid.choice</i>

7.3.3.2 Changes to Lesson Status

In SCORM 1.2, the objective satisfaction status and the attempt completion status information was managed with one data model element called “*cmi.core.lesson_status*”. This caused problems in the management of learning data.

For example, it was not possible to handle situations where the learner was not to be regarded as successful if he or she could not understand the content after completing it from start to end, or conversely the learner was to be regarded as successful if he or she understood the content without completing it. This was because it was not possible to represent the states of “*completed*” but “*not satisfied*”, or “*incomplete*” but “*satisfied*”, with the single data model element “*cmi.core.lesson_status*”.

In SCORM 2004, the data model elements have been changed in such a way that the completion status and success status are separately managed. In place of “*cmi.core.lesson_status*”, which has been removed, the two data model elements “*cmi.completion_status*” and “*cmi.success_status*” have been added to handle, respectively, the completion status and the success status of a lesson.

“*cmi.completion_status*” is the data model element for managing information concerning whether an attempt is completed. This data model element contains one of the following tokens:

completed
incomplete
not attempted
unknown

Note that the state token “*browsed*” used in SCORM 1.2 is no longer used.

“*cmi.success_status*” is the data model element for managing the learner’s success status for an activity to indicate whether the learner has mastered the content. This data model element contains one of the following tokens:

passed
failed
unknown

When translating “*cmi.core.lesson_status*” into the “*cmi.completion_status*” or “*cmi.success_status*” pair, it is necessary to make sure that the content developer’s intended information is correctly reflected by identifying whether the existing information indicates a completion status or success status.

7.3.3.3 Data Type Clarification

SCORM 2004 specifies the minimum and maximum values and sizes for character strings that are used as values for the data model elements. It also explicitly specifies the smallest permitted minimum (SPM) value for each data model element that a LMS must guarantee, and the new SCORM standard imposes a stricter data type description method.

When migrating from SCORM 1.2 to SCORM 2004, it is necessary to check content packages to make sure the values of the data model elements are correctly typed according to the standard and do not exceed the allowed range.

The data types specified in SCORM 2004 are as follows:

- *characterstring*
- *localized_string_type*

- *language_type*
- *long_identifier_type*
- *short_identifier_type*
- *integer*
- *state*
- *real(10,7)*
- *time(second,10,0)*
- *timeinterval(second,10,2)*

7.3.3.4 Detailed Specifications on Interactions

To improve the interoperability of the learner's log data format for practical exercise questions, the description format of replies and answers has been enhanced in SCORM 2004 in the areas of data types, the value space and expressions.

When migrating from SCORM 1.2 to SCORM 2004, it is necessary to check the content packages to ensure correct adherence to the enhanced specification.

When enabling SCOs to migrate to SCORM 2004, it is also necessary to watch for minor changes like the change of the question type name from "*choice*" to "*multiple_choice*".

7.3.3.5 Suspend Data

In SCORM 1.2, an SCO could save some values to the "*cmi.suspend_data*" data model element and the LMS could refer to these at any time during the run-time. For this reason, this data model element was used for various purposes. It was possible to save old scores and answers in the "*cmi.suspend_data*", and then refer to the values at the subsequent launch time. It was also possible to save the learner's progress information within an SCO in this data model element so that the lesson could be resumed from the interrupted point by retrieving the saved information from the data model element at the next start time.

In SCORM 2004, however, the "*cmi.suspend_data*" data model element can be saved on the LMS side only when the associated SCO is suspended (*Suspend All*). Therefore, when an SCO is terminated with the "*Continue*", "*Exit All*" or "*Abandon All*" sequencing request, the "*cmi.suspend_data*" data model element cannot be used. The "*cmi.suspend_data*" data model element becomes valid in SCORM 2004 only when an SCO is resumed after it was suspended.

In SCORM 2004, for this reason, it is necessary to use such data model elements as "*cmi.location*" rather than "*cmi.suspend_data*" for the above purposes. The *cmi.location* data model element is used to store a bookmark or checkpoint of an SCO. An SCO can save some values to the *cmi.location* and the LMS can refer to them as well.

The example below shows how to use the *cmi.location* data model element:

```
GetValue("cmi.location");
SetValue("cmi.location", "chkPt1.p3.f5");
```

Note that as the smallest permitted minimum (SPM) for the “*cmi.location*” data model element is 1000 characters while that for the *cmi.suspend_data* is 4000, caution is required when using these data model elements.

In addition, as well as being able to write values to the *interactions* data model element on the LMS, an SCO can retrieve the values from the LMS. For example, to utilize the *interactions* data model element to save a learner’s practical exercise data, answers and scores, the “*cmi.interactions.n.learner_response*” and “*cmi.interactions.n.learner_result*” data model elements can be used.

7.3.3.6 The LMS must Implement All Data Model Elements

A SCORM 2004 conformant LMS is required to implement all the data model elements. Under SCORM 1.2, SCOs could use only a limited number of data model elements depending on the LMS implementation, but this limitation has been lifted as it is mandatory that the LMS implement all data model elements.

When enabling SCOs to migrate to SCORM 2004, it is helpful to understand the uses of the RTE data model elements and then utilize the data model elements as required.

7.4 Changes to Error Codes

A more detailed error code structure has been introduced in SCORM 2004 to make it possible to check the state transitions of the API Instance and the validity of data elements. When migrating to SCORM 2004, it is therefore necessary to make changes to the SCOs so that they can correctly respond to the new range of error codes.

Table 7.6 compares the error codes between SCORM 1.2 and SCORM 2004.

Table 7.6 Error Code Comparison between SCORM 1.2 and SCORM 2004

SCORM 1.2 Error Code	SCORM 2004 Error Code
0 - No error	0 - No error
101 - General Exception	101 - General Exception
	102 - General Initialization Failure
	103 - Already Initialized
	104 - Content Instance Terminated
	111 - General Termination Failure
	112 - Termination Before Initialization
	113 - Termination After Termination
	122 - Retrieve Data Before Initialization
	123 - Retrieve Data After Termination
	132 - Store Data Before Initialization
	133 - Store Data After Termination
	142 - Commit Before Initialization
	143 - Commit After Termination
201 - Invalid argument error	201 - General Argument Error
202 - Element cannot have children	301 - General Get Failure
203 - Element not an array. Cannot have count	
	351 - General Set Failure

SCORM 1.2 Error Code	SCORM 2004 Error Code
	391 - General Commit Failure
401 - Not implemented error	401 - Undefined Data Model Element
401 - Not implemented error	402 - Unimplemented Data Model Element
301 - Not initialized	403 - Data Model Element Value Not Initialized
403 - Element is read only	404 - Data Model Element Is Read Only
404 - Element is write only	405 - Data Model Element Is Write Only
402 - Invalid set value, element is a keyword	
405 - Incorrect Data Type	406 - Data Model Element Type Mismatch
	407 - Data Model Element Value Out Of Range
	408 - Data Model Dependence Not Established

For the changes to the error codes, it is necessary to check the SCOs in the areas concerning the learner interactions and the state transitions of the API Instance where most of the specification changes have been made.

7.5 The Potential of SCORM 2004

This section has so far described how to realize the functions of SCORM 1.2 content under SCORM 2004, and explained the major points to note when converting SCORM 1.2 content to SCORM 2004. The first critical step of the content migration effort is to enable the content to run correctly in a SCORM 2004 Run-Time Environment.

However, content developers should also recognize the huge potential that SCORM 2004 offers. For example, they can take advantage of sequencing rules in developing content that can display a dynamic behavior in response to the sequence of learning experiences and the status of learning progress. They can also provide the optimal user interface for their intended learners through the new ability to trigger navigation request events from within the content. The main enabling factor is that a SCORM 2004 conformant LMS is now required to implement all the data model elements. Furthermore, as the concept of learning objectives is refined, applying a more realistic approach to the development of content that fulfills educational policies as intended will become increasingly practical.

Compared with SCORM 1.2, SCORM 2004 has enhanced the degree of freedom with which content developers can apply a variety of design strategies to their content development process. It is up to the content developers whether they will take advantage of this new potential. It is hoped that they will not be satisfied by merely ensuring their content will run in a SCORM 2004 environment – instead they should be engaged in developing more effective, easy-to-use content by taking advantage of the potential of SCORM 2004.

Index

- Activity, 12
- Activity tree, 12
- API, 44
- API adapter, 44
- API error codes, 52, 117
- API instance, 44, 51, 112
- API instance state transition, 51
- API methods, 47, 96, 112
- Asset, 43
- Attempt, 35, 111
- CAM, 6
- Changes from SCORM 1.2, 8
- characterstring, 61
- Cluster, 12, 72
- Content Aggregation Model, 6
- Content packaging, 108
- Content structure, 12
- Controlling LMS-provided UI devices, 105
- Data model, 101
- Data model elements, 9, 58, 65, 101, 113
- Data model extension, 63
- Data types, 61
- Delivery controls, 93
- DOM, 46
- ECMAScript, 46
- Exit action rules, 24
- Exit rule, 80
- integer, 62
- interactions, 116
- language type, 62
- Launching learning resources, 43
- Leaf activity, 12
- lesson_status, 114
- Limit condition, 21
- Limit conditions, 85
- LMS, 45
- LMS model, 5
- LMS-provided navigation devices, 40
- LMS-provided UI devices, 105
- Local objective, 89
- Local objectives, 86
- localized string type, 61
- long identifier type, 62
- Manifest file, 6, 8, 9, 12, 40, 65, 67, 68, 69, 71, 72, 107
- Migrating, 107
- masteryscore, 109
- Migration to SCORM 2004, 107
- Navigation, 36, 104
- Navigation control, 36
- Navigation request, 16, 104
- Navigation request event, 38
- Objective, 12, 86
- Objective measure rollup, 26
- Objective rollup, 26
- Post condition rules, 24
- Postcondition rules, 79
- Precondition rules, 78
- Precondition sequencing rules, 21
- prerequisites, 109
- Primary objective, 12, 13, 25, 26, 27, 29, 34
- Progress rollup, 27
- real, 62
- Rollup, 25
- Rollup rules, 25, 28, 72, 80, 81, 83, 84
- RTE, 6, 42, 94, 112
- RTE data model, 58, 63, 101
- RTE data model elements, 58, 112
- SCO, 43, 45, 94, 107, 112
- SCO migration, 112
- SCORM, 4
- SCORM 1.0, 7
- SCORM 1.1, 7
- SCORM 1.2, 8, 107
- SCORM 2004, 5
 - Migration, 107
- SCORM 2004 Overview, 6
- SCORM run-time environment, 42
- SCORM Run-Time Environment, 6
- SCORM run-time environment data model, 58
- SCORM Sequencing and Navigation, 6
- SCO's navigation event, 37
- Sequencing, 12, 72
- Sequencing control Choice Exit, 20
- Sequencing control Choice, 19
- Sequencing control Flow, 19
- Sequencing control Forward Only, 20
- Sequencing control modes, 19, 72, 110
- Sequencing request, 16
- Sequencing rules, 18, 75

Sharable Content Object (SCO), 43
Sharable Content Object Reference
 Model, 4, 7
Shared global objective, 86, 87, 89, 90
Shared global objectives, 86
short identifier type, 62
state, 62
Suspend data, 116
Termination request, 16
time, 62
timeinterval, 62
Tracking data, 14
Tree structure, 12
Use Current Attempt Objective
 Information, 20
Use Current Attempt Progress
 Information, 20

About This Document

Acknowledgements

This document was a translation of a Japanese document authored by the following people:

Sections 1, 2, and 4 – 7:	Miyauchi, Hiroshi (The Sanno Institute of Management) Ota, Mamoru (Enegate Co., Limited)
Sections 2, 4, and 5	Nakabayashi, Kiyoshi (NTT-Resonant Inc.)

Copyright

The Copyright of this document is owned by the e-Learning Consortium Japan (eLC), a non-profit organization.

The publication of this document was sponsored by the Japanese Ministry of Economy and Industry for the purpose of promoting the SCORM 2004 standard.

The eLC (the Licensor) permits the other party (the Licensee) to copy, distribute, display or hyper-link this document under the following conditions:

- The Licensee must acknowledge the Licensor explicitly by quoting the copyright notice and sponsorship statement above.
- The Licensee must not use this document for commercial purposes without written permission from the Licensor.