TECHNICAL REPORT

ISO/IEC TR 29163-3

First edition
2009-12-15

Information technology — Sharable Content Object Reference Model (SCORM®) 2004 3rd Edition —

Part 3: Run-Time Environment Version 1.1

Technologies de l'information — Modèle de référence d'objet de contenu partageable (SCORM®) 2004 3e édition —

Partie 3: Environnement du temps d'exécution Version 1.1

Reference number
ISO/IEC TR 29163-3:2009(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 29163-3, which is a Technical Report of type 3, was prepared by the Advanced Distributed Learning (ADL) Initiative (as SCORM® 2004 3rd Edition Run-Time Environment Version 1.1) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by the national bodies of ISO and IEC.

ISO/IEC TR 29163 consists of the following parts, under the general title *Information technology — Sharable Content Object Reference Model (SCORM®) 2004 3rd Edition*:

— *Part 1: Overview Version 1.1*

— *Part 2: Content Aggregation Model Version 1.1*

— *Part 3: Run-Time Environment Version 1.1*

— *Part 4: Sequencing and Navigation Version 1.1*

This page intentionally left blank.

Advanced Distributed Learning (ADL)

SCORM® 2004 3rd Edition Run-Time Environment (RTE) Version 1.1

For questions and comments visit
Ask The Experts at ADLNet.gov

SCORM® is a registered trademark of the Department of Defense, an agency of the United States government, located at The Pentagon, Washington, DC 20301.

This page intentionally left blank.

Chief Technical Architect
Philip Dodds

Technical Editor
Schawn E. Thropp

Acknowledgements

ADL would like to thank the following organizations and their members for their continued commitment to building interoperable e-learning standards and specifications:

Alliance of Remote Instructional Authoring & Distribution Networks for Europe (ARIADNE) (http://www.ariadne-eu.org/)

Aviation Industry CBT Committee (AICC) (http://www.aicc.org/)

Institute of Electrical and Electronics Engineers (IEEE) Learning Technology Standards Committee (LTSC) (http://ltsc.ieee.org/)

IMS Global Learning Consortium, Inc. (http://www.imsglobal.org/)

ADL would also like to thank the ADL Community for their commitment and contribution to the evolution of SCORM.

This page intentionally left blank.

Table of Contents

List of Figures

List of Tables

This page intentionally left blank.

SECTION 1
The SCORM® Run-Time Environment (RTE)

This page intentionally left blank.

1.1. Introduction to the SCORM Run-Time Environment (RTE) Book

The Sharable Content Object Reference Model (SCORM®) is often described as a set of books on a bookshelf. The Run-Time Environment (RTE) book is one of a set of books (refer to Figure 1.1a: *The Run-Time Environment Book as Part of the SCORM Bookshelf*). More information on the other SCORM books and their relationships to one another can be found in the SCORM 2004 Overview. The SCORM RTE book describes the Learning Management System (LMS) requirements in managing the run-time environment (i.e., content launch process, standardized communication between content and LMSs and standardized data model elements used for passing information relevant to the learner's experience with the content). The RTE book also covers the requirements of Sharable Content Objects (SCOs) and their use of a common Application Programming Interface (API) and the SCORM Run-Time Environment Data Model.

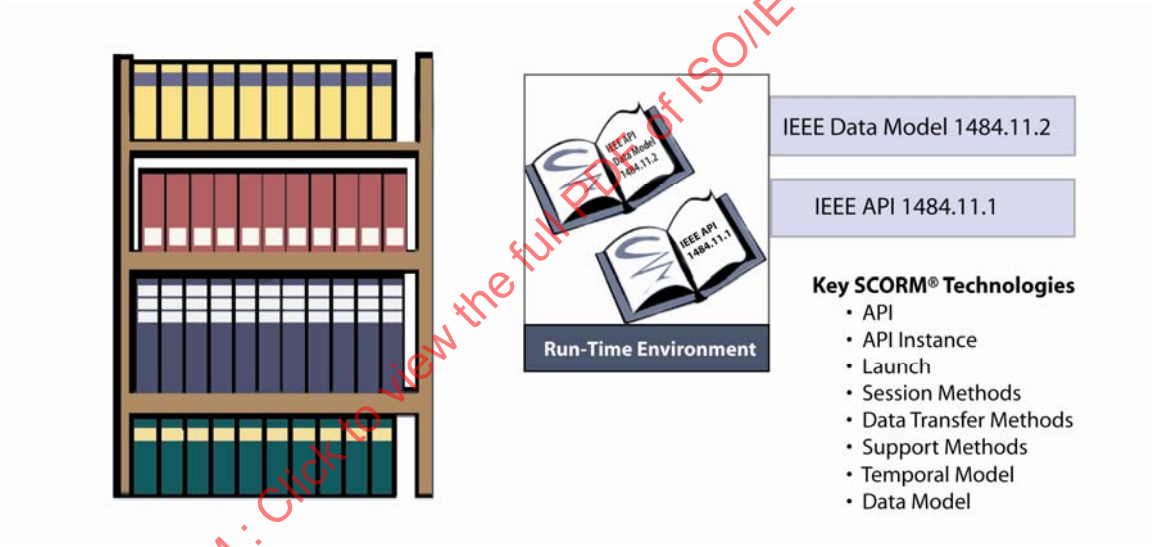

Figure 1.1a: The SCORM Run-Time Environment Book as Part of the SCORM Bookshelf

1.1.1. What is Covered in the SCORM RTE Book?

There are several key concepts that are introduced in the SCORM RTE book. The book covers the essential LMS responsibilities for sequencing content objects (SCOs or Assets) during run-time and allowing SCOs to indicate navigation requests. In addition, guidance is offered for providing navigation controls to learners. General subjects discussed include:

- RTE Management: Launching of content objects – SCOs and Assets, Management of communications with a SCO, Run-time environment data model management

- Application Programming Interface (API): LMS API requirements, SCORM communication requirements, communication error conditions
- SCORM Run-Time Environment Data Model: Data model management and behavior requirements, Data type requirements

1.1.2. Using the SCORM RTE Book

This book should prove useful to LMS, SCO and authoring tool vendors wishing to support SCORM in their products, and to anyone wishing to understand the communications relationship between content and LMSs, such as SCORM content developers.

Section 1: *The SCORM® Run-Time Environment (RTE)* and Section 2: *Managing the Run-Time Environment (RTE)* of this book cover general RTE-related concepts. These sections are recommended reading for those seeking an introduction to the concepts behind the SCORM RTE and who may not wish to delve into its technical details. Others who may find these sections useful include those wishing to learn about updates to the RTE. Section 2.1: *Run-Time Environment (RTE) Management* for instance, discusses how the new Sequencing and Navigation book impacts the SCORM RTE.

Section 3: *Application Programming Interface (API)* is the first section providing technical details about the RTE. This section explains every SCORM API method and error message available to content developers, and even provides sample code as well as API Usage Requirements and Guidelines.

Section 4: *SCORM® Run-Time Environment Data Model* covers every SCORM data model element in detail, which includes a listing of specific LMS and SCO behavior requirements in relation to a given element.

1.1.3. Relationship with other SCORM Books

While the various SCORM books are intended to stand alone, there are areas of overlap or mutual coverage. For instance, while this book focuses primarily on communication between learning content and LMSs, it frequently refers to SCOs that conduct that communication. SCOs are discussed in some detail in the SCORM Content Aggregation Model (CAM) book.

Similarly, while the SCORM Sequencing and Navigation (SN) book covers the details of SCORM sequencing and navigation processes, including detailed coverage of how an LMS evaluates navigation requests and related activities, this book deals with content delivery, and as such, lightly touches on how an LMS determines which piece of content to deliver at any given time.

1.1.3.1 The SCORM Content Aggregation Model (CAM) Book

The SCORM CAM book [18] defines responsibilities and requirements for building content aggregations (i.e., the process of assembling, labeling and packaging content). The book contains information on creating content packages, applying metadata to the components in the content package and applying sequencing and navigation details in the context of a content package. Several dependencies span from the SCORM CAM book to the SCORM RTE book.

Metadata is "data about data". Metadata can be used to describe the different components of the SCORM content model (Content Aggregations, Content Organizations, Activities, SCOs and Assets). Metadata, a form of labeling, enhances search and discovery of these components. At this time, there are no defined relationships between metadata and the SCORM Run-Time Environment Data Model. For these reasons, metadata is not discussed in detail in the SCORM RTE book. This relationship may change as SCORM evolves.

A *Content Package*, in a general sense, bundles content objects with a content structure that is described by a manifest. A SCORM Content Package may represent a SCORM course, lesson, module or may simply be a collection of related content objects that may be stored in a repository. The manifest, an essential part of all SCORM Content Packages, is contained in an Extensible Markup Language (XML)-based file named `imsmanifest.xml`. This file, similar in many ways to a "packaging slip", describes the contents of the package and may include an optional description of the content structure.

SCORM Content Packages may include additional information that describes how an LMS is intended to process the Content Package and its contents. Some of these elements are utilized by the SCORM RTE model.

- Content object launch locations and launch parameters are also described as elements in a SCORM Content Package. These elements are essential to the launch and delivery of content objects. The SCORM RTE book details these elements and their effects on launching content objects.
- Several elements in a SCORM Content Package affect initialization and management of a content object's run-time data model. The SCORM RTE book details these data model elements and the required LMS behaviors.
- Other elements in a SCORM Content Package describe initial values for specific elements of a content object's run-time data model. The SCORM RTE book details these data model elements and their initialization behavior.
- When a SCORM Content Package includes a description of content structure, sequencing information elements may be added to define an intended approach to sequencing the package's content objects. A SCORM Content Package may include User Interface (UI) elements that are intended to provide guidance to an LMS on how certain UI navigation controls are to present, enabled or hidden. When a content object is launched, as defined in this book, these elements may be used, in conjunction with sequencing information (refer to the SCORM SN book), to present the correct (at the time of rendering) UI navigation controls (e.g., "Continue" or "Previous" user interface controls).

For a better understanding of how all of the elements described above are specified in a SCORM Content Package, refer to the SCORM CAM book.

1.1.3.2 The SCORM Sequencing and Navigation (SN) Book

The SCORM SN book is based on the IMS Simple Sequencing (SS) Specification [17], which defines a method for representing the intended behavior of an authored learning experience such that any SCORM conformant LMS will sequence discrete learning activities consistently.

The SCORM SN model defines how the IMS SS specification is applied and it is extended in a SCORM environment. It defines the required behaviors and functionality that SCORM conformant LMSs must implement to process sequencing information at run-time. More specifically, it describes the branching and flow of learning activities in terms of an Activity Tree, based on the results of a learner's interactions with launched content objects and an authored sequencing strategy. An Activity Tree is a conceptual structure of learning activities managed by the LMS for each learner.

The SCORM SN book describes how learner- and system-initiated navigation events can be triggered and processed, resulting in the identification of learning activities for delivery. Each learning activity identified for delivery will have an associated content object. The SCORM RTE model describes how identified content objects are launched. The sequence of launched content objects, for a given learner and content structure, provides a learning experience (learner interaction with content objects); the SCORM RTE Model describes how the LMS manages the resulting learning experience and how that learning experience may affect the Activity Tree.

1.2. Run-Time Environment Overview

This book defines the SCORM RTE model, which details the requirements for launching content objects, establishing communication between LMSs and SCOs, and managing the tracking information that can be communicated between SCOs and LMSs. In the context of SCORM, content objects are either:

- SCOs, which communicate during run-time, or
- Assets, which do not communicate during run-time.

The SCORM RTE book describes a common content object launch mechanism, a common communication mechanism between content objects and LMSs, and a common data model for tracking a learner's experience with content objects. These aspects create an environment where several of the ADL "-ilities" are satisfied. For example, content objects that communicate through the standardized communication mechanism can be moved from LMS to LMS without modification to their communication attempts; this increases learning object portability and durability, thereby lowering the cost of development, installation and maintenance.

The SCORM RTE defines a model that picks up at the point when a specific content object has been identified for launch. The actual identification of the content object to be launched is out of scope of this book and can be found in the SCORM SN book [11].

This book only deals with the management of the run-time environment, which includes:

- The delivery of a content object to the learner's Web browser (i.e., launch);
- If necessary, how a content object communicates with the LMS; and
- What information is tracked for a content object and how the LMS manages that information.

The following sections explain the relationships between the SCORM RTE book and the remaining SCORM books. In addition, frequently used terminology will be introduced at a high level to eliminate the need for the reader to become an expert in the entire SCORM to understand this book. This, however, is not an effective method to learn and apply SCORM and its concepts as a whole. It is strongly recommended that each SCORM book be read to more fully understand the purpose, details, relationships and advantages of all of the SCORM concepts.

SCORM was developed to enable the development of content objects that are reusable and interoperable across multiple LMSs. For this to be possible, there must be a common way to launch and manage content objects, a common mechanism for content objects to communicate with an LMS and a predefined language or vocabulary forming the basis of the communication. As illustrated in Figure 1.2a, these three aspects of the RTE are Launch, API and Data Model.

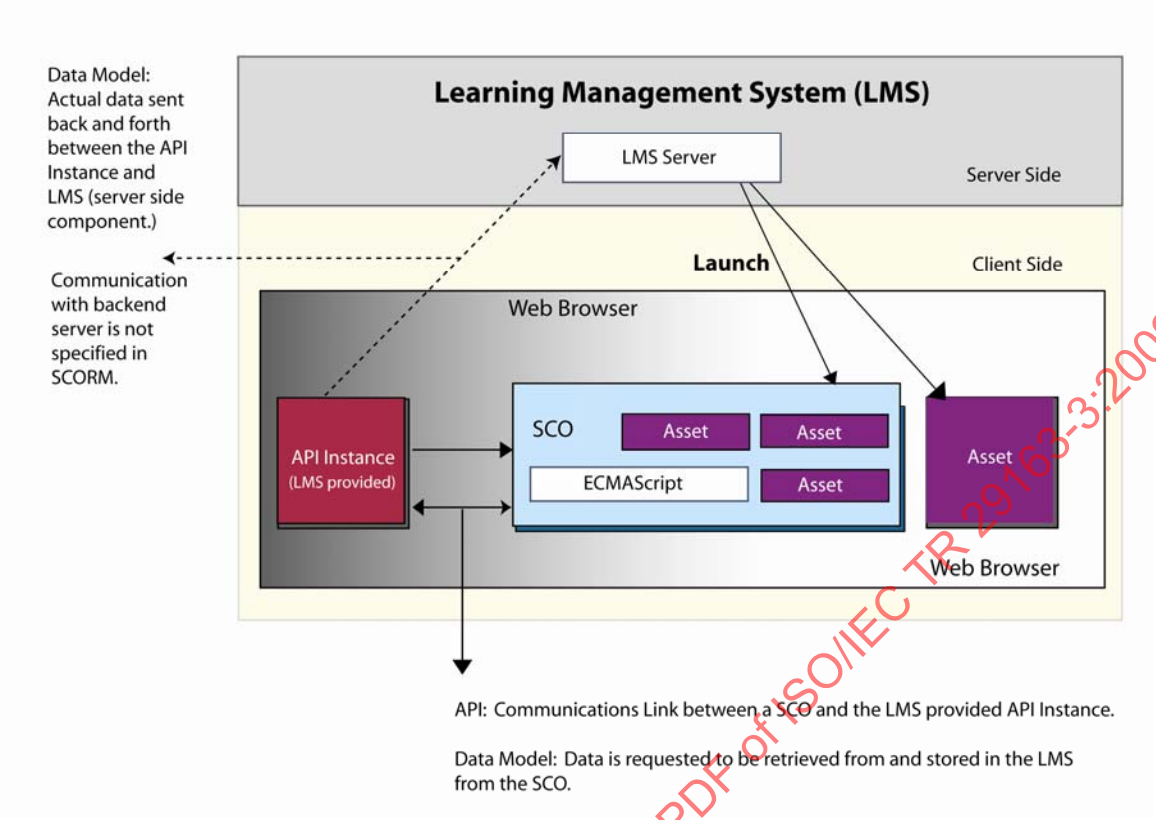

Figure 1.2a: SCORM Conceptual Run-Time Environment.

The *Launch* process defines a common way for LMSs to start Web-based content objects. The term "content object" is used generically here to describe any piece of content that can be launched for a learner. In SCORM, there are two types of content objects: SCOs and Assets. The launch process defines procedures and responsibilities for the establishment of communication between the launched content object and the LMS. The communication mechanism is standardized with a common API.

The *API* is the communication mechanism for informing the LMS of the conceptual communication state between a content object and an LMS (e.g., initialized, terminated and/or in an error condition), and is used for retrieving and storing data (e.g., score, time limits, etc.) between the LMS and the SCO.

A *Data Model* is a standard set of data model elements used to define the information being tracked for a SCO, such as the SCO's completion status or a score from an assessment such as a quiz or a test. In its simplest form, the data model defines data model elements that both the LMS and SCO are expected to "know about." The LMS must maintain the state of SCO's data model elements across learner sessions, and the SCO must utilize only these predefined data model elements to ensure reuse across multiple systems.

SECTION 2
Managing The Run-Time Environment (RTE)

This page intentionally left blank.

2.1. Run-Time Environment (RTE) Management

While the learner interacts with content objects (the learning experience), the LMS evaluates learner performance and navigation requests (refer to the SCORM SN book). When the LMS identifies an activity for delivery to the learner, the activity has a content object associated with it. The LMS will launch the content object and present it to the learner. Figure 2.1a depicts how the content structure (organization section of a manifest) can be interpreted in a tree (i.e., activity tree). The tree representation is just a different way of presenting the content structure found in the manifest (refer to the SCORM CAM book).

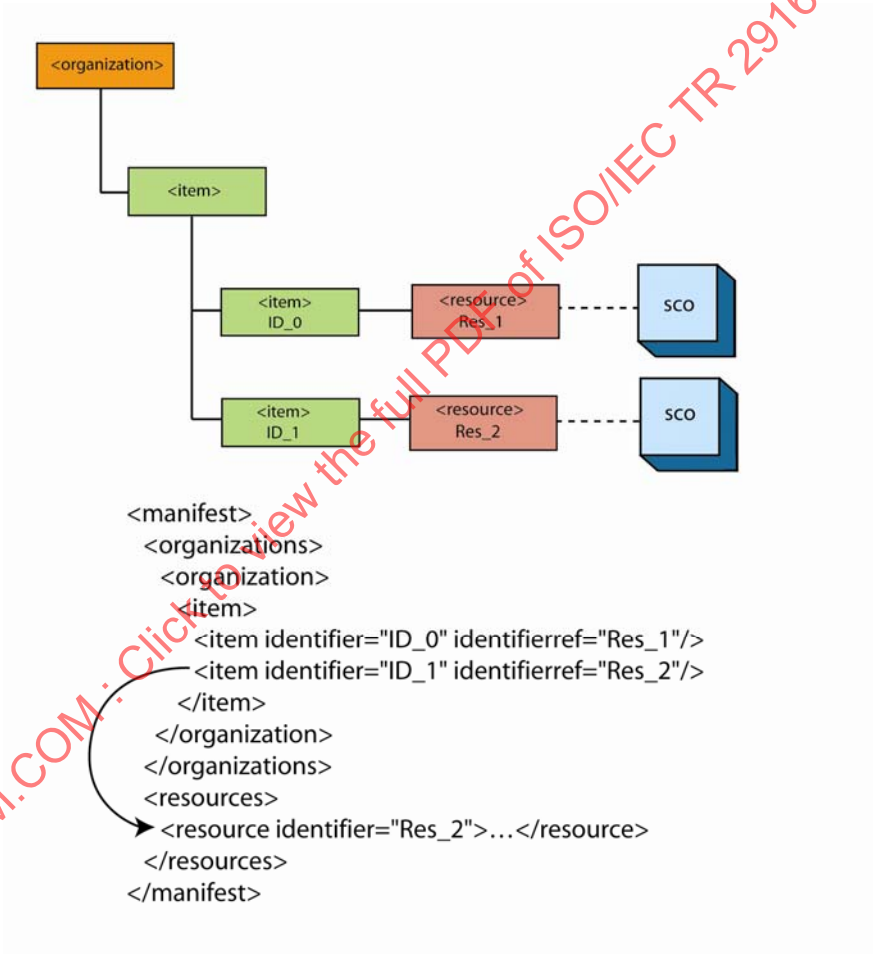

Figure 2.1a: Conceptual Content Structure Illustration

A common launch model addresses delivery of Web-enabled content objects in the form of SCOs and Assets within the context of a learning experience. This launch model enables consistency of content object launch behavior across LMSs without specifying the underlying LMS implementation. In this context, the term “LMS” is used to describe any system that provides the launch of content objects.

2.1.1. Run-Time Environment Temporal Model

A learner becomes engaged with the content object once an activity with the associated content object (i.e., SCOs or Asset) has been identified for delivery and the content object has been launched in the learner's browser-environment. Several key aspects need to be defined to aid in the tracking of the learner during the learning experience. The following terms are defined by the Institute of Electrical and Electronics Engineers (IEEE) 1484.11.1 Standard for Learning Technology – Data Model for Content Object Communication [1] and are referenced throughout this document:

Learner Attempt – A tracked effort by a learner to satisfy the requirements of a learning activity that uses a content object. An attempt may span one or more learner sessions and may be suspended between learner sessions [1].

Learner Session – An uninterrupted period of time during which a learner is accessing a content object [1].

Communication Session – An active connection between a content object (i.e., SCO) and an application programming interface [1].

Login Session – A period of time during which a learner begins a session with a system (logged on) until the time the learner terminates the session with the system (logged out).

These four terms are relevant when it comes to managing the RTE for a SCO. For an Asset, the RTE consists of only independent learner attempts and learner sessions; one learner attempt with a corresponding learner session for each launch of the Asset. The learner attempt begins once an activity has been identified to be delivered (refer to the SCORM SN book). During the attempt, the learner will be engaged with a content object (either a SCO or Asset). Once the learner becomes engaged (content has been launched in the learner's Web browser), a learner session begins. If the launched content object is a SCO, as soon as the SCO initializes communication with the LMS, a communication session begins. A communication session ends when the SCO terminates communication with the LMS. Learner sessions can end leaving the SCO in a suspended state (the learner is not through with the SCO) or leaving the SCO in a normal state (the learner ended the learner attempt by meeting requirements defined by the SCO). For a SCO, the learner attempt ends when a learner session ends in a normal state. For an Asset, the learner attempt ends once the Asset is taken away from the learner. Figure 2.1.1a depicts the four terms and their relationships with one another for a specific SCO.

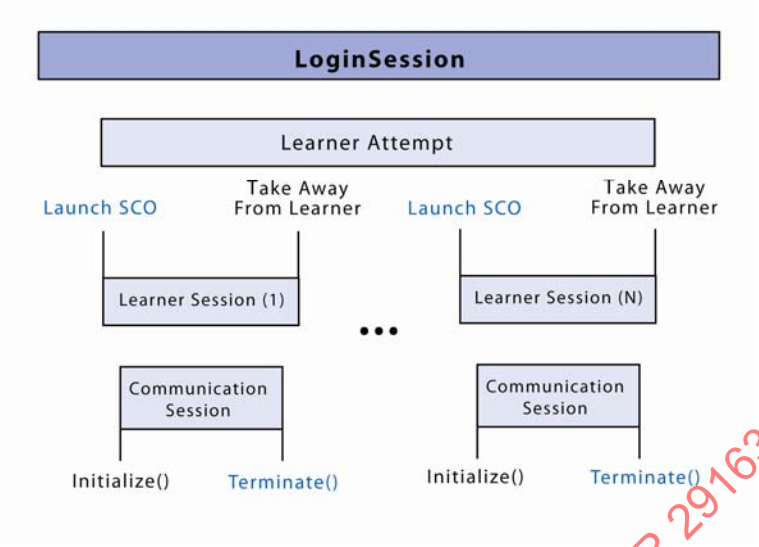

Figure 2.1.1a: Temporal Model Relationships for a Specific SCO

2.1.1.1 Managing Learner Attempts and Learner Sessions

A learner attempt is associated with an important LMS requirement defining the management of the set of run-time data model elements that the SCO may communicate to the LMS. When a new learner attempt begins for a SCO, the LMS is required to create and initialize a new set of run-time data for the SCO to access and use (refer to Section 4: *SCORM® Run-Time Environment Data Model* for more details). SCORM does not specify exactly when the new set of run-time data is created, but all data model accesses must appear (to the SCO) as if they are being performed on a new set of run-time data.

What the LMS does with the previous attempt's data is outside the scope of SCORM. The LMS may elect to store this data for historical purposes or for other purposes such as reporting, auditing, diagnostic or statistical. The LMS may elect to discard any run-time data collected during the previous attempt. The LMS is only required to keep run-time data for the learner attempt if the learner attempt was suspended. SCORM does not specify any LMS requirements on the persistence or access to previous learner attempt data. However, if the learner session is suspended, hence causing the learner attempt to be incomplete (i.e., suspended), the LMS is responsible for ensuring that any run-time data that was set prior to the suspension is available when the next learner session for the SCO begins; that is, the next time the (suspended) learning activity associated with the SCO is identified for delivery, the previous learner attempt is resumed and the run-time data from the previous learner attempt is provided in a new learner session.

The following figures (2.1.1.1a, 2.1.1.1b and 2.1.1.1c) illustrate several different relationships between a learner attempt and learner session(s). Figure 2.1.1.1a illustrates a single learner attempt being accomplished with one learner session.

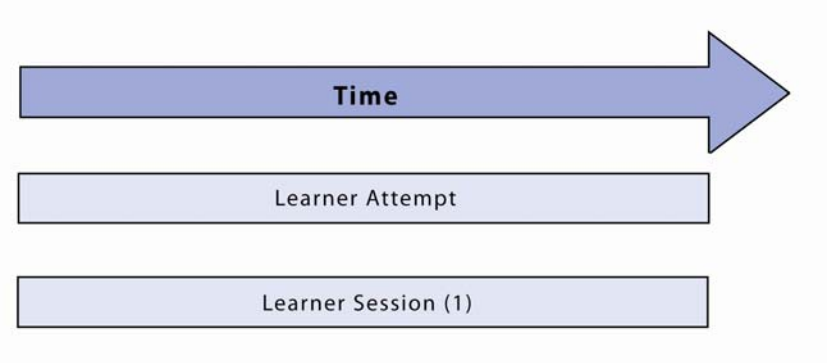

Figure 2.1.1.1a: Single Learner Attempt with one Learner Session

Figure 2.1.1.1b illustrates a single learner attempt spread over several learner sessions. These sessions have been suspended and the learner attempt has been subsequently resumed until a learning session ends in a normal state.

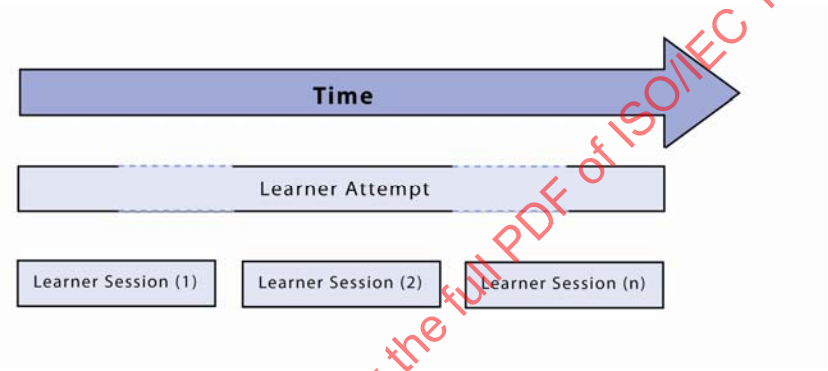

Figure 2.1.1.1b: Learner Attempt Spread Over Several Learner Sessions

Figure 2.1.1.1c illustrates successive learner attempts. Within each of these learner attempts, any number of learner sessions may take place.

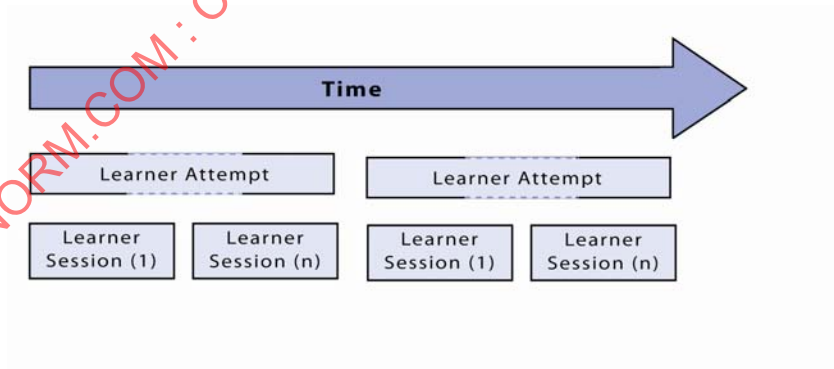

Figure 2.1.1.1c: Successive Learner Attempts, Spread Over Several Learner Sessions

2.1.2. Launching Content Objects

As described in the SCORM CAM [18], the SCORM Content Model is made up of three components:

- Assets
- SCOs
- Content Organizations

The SCORM CAM book describes the characteristics of launchable content objects; SCOs and Assets are the defined content model components that can be launched. Different launching requirements exist depending on the content object's type. The launching process defines the common way for LMSs to launch content objects to the learner's Web browser. The procedures and responsibilities for establishing communication between the launched content object and the LMS vary depending on the type of the launched content object.

It is the responsibility of the LMS to manage the sequencing between learning activities (refer to the SCORM SN book) based on well-defined behaviors and the evaluation of defined sequencing information applied to activities. The progression through learning activities that comprise a particular learning experience may be sequential, non-sequential, user-directed or adaptive, depending on the sequencing information defined and the interactions between a learner and content objects.

It is the responsibility of the LMS (or sequencing component/service thereof), based on some navigation event, to determine which learning activity to deliver. The LMS may identify the next learning activity in the sequence defined in the content structure, identify a user selected learning activity or determine which learning activity to deliver based on learner performance in previously experienced content objects in an adaptive fashion. A learning activity identified for delivery will always have an associated content object. It is the responsibility of the LMS (or launch component/service thereof) to launch the content object associated with the identified learning activity. Upon determining the appropriate content object to launch, the LMS uses the Universal Resource Locator (URL) defined by the content object's launch location, defined in the content package (refer to Code Illustration 2-1), to navigate to, or replace the currently displayed content object with the content object referenced by the launch location.

```
<manifest>
   <organizations>
      <organization>
         <item>
            <item identifierref="RES_1">…</item>
            <item> … </item>
            <item> … </item>
         </item>
      </organization>
   </organizations>
   <resources>
      <resource identifier="RES_1"
                type="webcontent"
                adlcp:scormType="sco"
                href="Lesson1/Module1/sco1.htm"> … </resource>
   </resources>
</manifest>
```

Code Illustration 2-1: Href used for Launching

It is the responsibility of the LMS to determine the appropriate fully qualified URL. The SCORM CAM defines the requirements on how to build the fully qualified URL. The URL is built based on the following pieces (if they exist in a manifest):

- xml:base declarations
- Launch Parameters (i.e., query component [6] of a URL)
- Href declarations

Refer to the SCORM CAM for more information on the process involved in building the absolute URL for the content object.

The LMS may implement the launch in any manner desired or may delegate the actual launch responsibility to the client or server portion of the LMS as needed. The actual launch must be accomplished using the Hypertext Transfer Protocol (HTTP). Ultimately, the content object identified by the launch location in a content package is launched and delivered to the client Web browser.

2.1.2.1 Asset

For content objects that represent Assets, the SCORM launch model only requires that an LMS launch the Asset using the HTTP protocol. An Asset does not communicate to the LMS via the API and Data Model.

2.1.2.2 Sharable Content Object (SCO)

For content objects that represent SCOs, the SCORM launch model requires that an LMS launch and track one SCO at a time (per learner). In other words, the LMS launches and tracks one SCO at a time (per learner).

The LMS must launch the SCO in a Web browser window that is a dependent window (i.e., “pop-up” window) or child frame of the LMS window that exposes the API Instance as a Document Object Model (DOM) object [8]. The API Instance must be provided by the LMS.

It is the responsibility of the SCO to recursively search the parent and/or opener window hierarchy until the API Instance is found. Once the API Instance has been found the SCO may initiate communication with the LMS. Section 3.3: *SCO Responsibilities* defines the requirements of the SCO. The SCO is responsible for adhering to all requirements defined by the API Instance functions (refer to Section 3.1: *Application Programming Interface*).

2.1.3. Taking Content Objects Away

At the conclusion of a learner session, the content object currently being experienced by the learner will be taken away and replaced with the next content object identified for delivery (refer to Section 2.1.2: *Launching Content Objects*). Typically, content objects are taken away in response to a learner or system triggered navigation event (refer to the SCORM SN book for more details). After the current content object is taken away, the LMS needs to have the most accurate information regarding the learner's interactions with the content object to make correct sequencing evaluations. If the content object taken away is an Asset, the LMS will make assumptions regarding the learner's interactions. If the content object taken away is a SCO, the SCO may have communicated more finely grained information regarding the learner's interactions during the just ended learner session. It is the LMSs' responsibility to account for information communicated by a SCO through its run-time data model that may affect successive sequencing evaluations. The run-time data model section describes the LMS responsibilities for mapping the data model elements that may affect sequencing evaluations to the learning activity associated with the SCO. To ensure timely use of run-time data in sequencing evaluations, it is recommended that LMSs account for run-time data changes on each commit data event (refer to Section 3: *Application Programming Interface*).

In some cases, the content author does not want to allow the user to interact with the SCO after it has finished (whatever "finish" means in the context of the SCO). In such a case, the following behaviors are allowed, depending on the type of window in which the SCO was launched (refer to Section 3.2: *LMS Responsibilities*):

1. If the window in which the SCO was launched is a top-level window (i.e., the window has no parent window, but it has an opener) then the SCO may attempt to close the window after calling `Terminate("")`. There is no requirement that the SCO behave this way. It is recommended that an LMS monitor the status of the dependent pop-up window in which it launched the SCO to detect when such an event happens. This will allow an LMS to present the appropriate implementation-defined user interface to the learner.

2. If the window is not a top-level window (i.e., the window has a parent window), the SCO may not act on the parent window or any window in the chain of parents. For example, a SCO is not allowed to attempt to close the top window, unless it is its own window.

In such a case, the recommended behavior is for the SCO to display neutral, passive content while waiting to be taken away by the LMS.

SECTION 3
Application Programming Interface (API)

This page intentionally left blank.

3.1. Application Programming Interface (API)

3.1.1. API Overview

Early versions of SCORM, up to and including SCORM Version 1.2, were based on the run-time environment functionality defined in the AICC's CMI001 Guidelines for Interoperability [7]. Since then, the Aviation Industry CBT Committee (AICC) has submitted their work to the IEEE Learning Technology Standards Committee (LTSC) with the emphasis to standardize various pieces of the Computer Managed Instruction (CMI) Guidelines for Interoperability. SCORM describes the IEEE 1484.11.2-2003 Standard for Learning Technology - ECMAScript Application Programming Interface for Content to Runtime Services Communication [2] as used in SCORM. The standard describes the API for content to run-time service (RTS) communication. An RTS is defined as the software that controls the execution and delivery of learning content and that may provide services such as resource allocation, scheduling, input-output control and data management [2]. From a SCORM perspective, the term RTS and LMS are terms that could be used interchangeably. The API enables the communication of data between content and an RTS typically provided by an LMS via a common set of API services using the ECMAScript [9] (more commonly known as JavaScript) language. In this section, the term "content" used by the IEEE standard relates to a SCO (because in SCORM, SCOs are the content objects that communicate to an LMS using the API).

The use of a common API fulfills many of SCORM's high-level requirements for interoperability and reuse. It provides a standardized way for SCOs to communicate with LMSs, yet it shields the particular communication implementation from the SCO developer. How the LMS-provided API Instance communicates with the server-side component of the LMS is outside the scope of SCORM. This back-channel communication can be implemented anyway the LMS vendor likes.

There are several terms that are used throughout SCORM: API, API Implementation and API Instance. Figure 3.1.1a describes the terms and their relationships to each other.

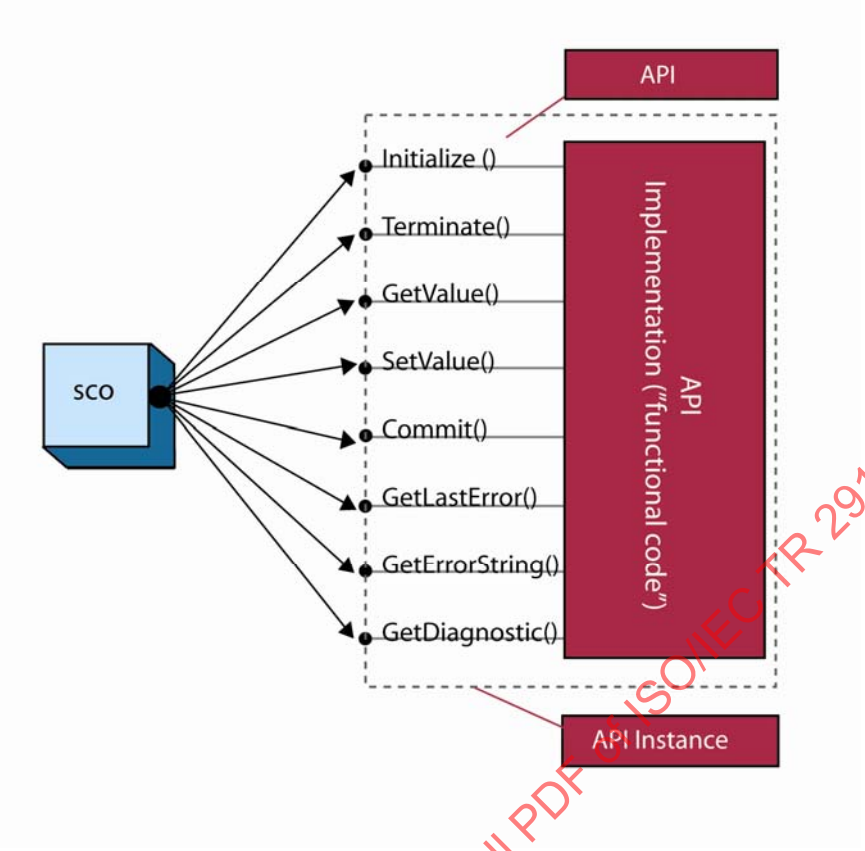

Figure 3.1.1a: Illustration of API, API Instance and API Implementation

In its simplest terms, the *API* is merely a set of defined functions that the SCO can rely on being available.

An *API Implementation* is a piece of functional software that implements and exposes the functions of the API. How an API Implementation functions should not matter to a SCO developer, as long as the API Implementation uses the same public interface and adheres to the semantics of the interface. The LMS need only provide an API Implementation that implements the functionality of the API and exposes its public interface to the client SCO.

An *API Instance* is an individual execution context and state of an API Implementation [2]. The API Instance represents the piece of executing software that the SCO interacts with during the SCOs operation.

A key aspect of the API is to provide a communication mechanism that allows the SCO to communicate with the LMS. It is assumed that once the SCO is launched it can then store and retrieve information with an LMS. All communication between the LMS and the SCO is initiated by the SCO. There is currently no supported mechanism for LMSs to initiate calls to functions implemented by a SCO.

The methods exposed by the API Implementation are divided into three categories. Table 3.1.1b defines these categories.

Table: 3.1.1b: Categories of API Methods

Method	Description
Session Methods	Session methods are used to mark the beginning and end of a communication session between a SCO and an LMS through the API Instance.
Data-transfer Methods	Data-transfer methods are used to exchange data model values between a SCO and an LMS through the API Instance.
Support Methods	Support methods are used for auxiliary communications (e.g., error handling) between a SCO and an LMS through the API Instance.

3.1.2. API Methods and Syntax

The use of a common API fulfills many of SCORM's high-level requirements for interoperability and reuse. It provides a standardized way for SCOs to communicate with LMSs, yet it shields the particular communication implementation from the SCO developer. This is true if a SCO can find the API Instance in a consistent manner. Otherwise, content developers must adapt their content to work on different LMS vendor's systems. This is one of the primary reasons why there are restrictions on where, in the Document Object Model (DOM) hierarchy, the LMS provides the API Instance and why there is a common name of the API Instance to search.

There are some general requirements dealing with the API that shall be adhered to:

- All function names are case sensitive and shall be expressed exactly as shown.
- All function parameters or arguments are case sensitive.
- Each call to an API function, other than the Support Methods, sets the error code.
- All parameters and return values are passed as characterstrings and shall be compatible with the data types and formats described by the data models that use the API for communication. All values that are type as integers, real numbers and times are to be encoded as they would be by the ECMAScript-to-string cast conversion. Several examples throughout SCORM include return data with quotes (""). The quotes are not intended to be part of the characterstring returned. The quotes are used to delineate the value as a characterstring.

A key aspect of the API is that it allows the SCO to communicate with the LMS. It is assumed that once the SCO is launched, it exchanges (i.e., "get" and "set") information with an LMS. All communication between the API Instance and the SCO is initiated by the SCO. In other words, the communication is initiated in one direction, from the SCO to the LMS. The SCO always invokes functions on the LMS's API Instance. The LMS does not invoke any functions defined by the SCO. This should not be confused with the notion of the API Instance returning a value. That is done purely in response to the call initiated by the SCO. There are currently no supported mechanisms for LMSs to initiate calls to functions implemented by a SCO.

All of the API functions are described in detail in the following sections. Some functions refer to a data model. The data model is described in detail in Section 4: *SCORM® Run-Time Environment Data Model*. Error handling and error codes are described in detail in Section 3.1.7: *API Implementation Error Codes*.

3.1.3. Session Methods

A SCO uses session methods to initiate and terminate data communication between itself and an API Instance.

3.1.3.1 Initialize

Method Syntax: `return_value = Initialize(parameter)`

Description: The function is used to initiate the communication session. It allows the LMS to handle LMS specific initialization issues.

Parameter: (“”) – empty characterstring. An empty characterstring shall be passed as a parameter.

Return Value: The function can return one of two values. The return value shall be represented as a characterstring. The quotes (“”) are not part of the characterstring returned, they are used purely to delineate the values returned.

- “`true`” – The characterstring “`true`” shall be returned if communication session initialization, as determined by the LMS, was successful.
- “`false`” – The characterstring “`false`” shall be returned if communication session initialization, as determined by the LMS, was unsuccessful. The API Instance shall set the error code to a value specific to the error encountered. The SCO may call `GetLastError()` to determine the type of error. More detailed information pertaining to the error may be provided by the LMS through the `GetDiagnostic()` function.

3.1.3.2 Terminate

Method Syntax: `return_value = Terminate(parameter)`

Description: The function is used to terminate the communication session. It is used by the SCO when the SCO has determined that it no longer needs to communicate with the LMS. The `Terminate()` function also shall cause the persistence of any data (i.e., an implicit `Commit(“”)` call) set by the SCO since the last successful call to `Initialize(“”)` or `Commit(“”)`, whichever occurred most recently. This guarantees to the SCO that all data set by the SCO has been persisted by the LMS.

Once the communication session has been successfully terminated, the SCO is only permitted to call the Support Methods.

Parameter: (“”) – empty characterstring. An empty characterstring shall be passed as a parameter.

Return Value: The method can return one of two values. The return value shall be represented as a characterstring. The quotes (“”) are not part of the characterstring returned, they are used purely to delineate the values returned.

- “`true`” – The characterstring “`true`” shall be returned if termination of the communication session, as determined by the LMS, was successful.
- “`false`” – The characterstring “`false`” shall be returned if termination of the communication session, as determined by the LMS, was unsuccessful. The API Instance shall set the error code to a value specific to the error encountered. The SCO may call `GetLastError()` to determine the type of error. More detailed information pertaining to the error may be provided by the LMS through the `GetDiagnostic()` function.

3.1.4. Data-Transfer Methods

A SCO uses data-transfer methods to direct the storage and retrieval of data to be used within the current communication session. The SCO uses these methods to transfer run-time data to and from the LMS. For example, the LMS can use this data to help determine completion/mastery of activities and make sequencing and navigation decisions.

3.1.4.1 GetValue

Method Syntax: `return_value = GetValue(parameter)`

Description: The function requests information from an LMS. It permits the SCO to request information from the LMS to determine among other things:

- Values for data model elements supported by the LMS.
- Version of the data model supported by the LMS.
- Whether or not specific data model elements are supported.

Parameter: The `parameter` represents the complete identification of a data model element.

Return Value: The method can return one of two values. The return value shall be represented as a characterstring.

- A characterstring containing the value associated with the `parameter`
- If an error occurs, then the API Instance shall set an error code to a value specific to the error and return an empty characterstring (“”). The SCO may call `GetLastError()` to determine the type of error. More detailed information pertaining to the error may be provided by the LMS through the `GetDiagnostic()` function.

The SCO should not rely on an empty characterstring returned from this function as being a valid value. The SCO should check to see if the error code indicates that no error was encountered. If this is the case, then the empty characterstring is a valid value returned from the LMS. If an error condition was encountered during the processing of the request, then this would be indicated by an appropriate error code (refer to Section 3.1.7: *API Implementation Error Codes*).

3.1.4.2 SetValue

Method Syntax: `return_value = SetValue(parameter_1, parameter_2)`

Description: The method is used to request the transfer to the LMS of the value of `parameter_2` for the data element specified as `parameter_1`. This method allows the SCO to send information to the LMS for storage. The API Instance may be designed to immediately persist data that was set (to the server-side component) or store data in a local (client-side) cache.

Parameter:

- `parameter_1` – The complete identification of a data model element to be set.
- `parameter_2` – The value to which the contents of `parameter_1` is to be set. The value of `parameter_2` shall be a characterstring that shall be convertible to the data type defined for the data model element identified in `parameter_1`.

Return Value: The method can return one of two values. The return value shall be represented as a characterstring. The quotes (“”) are not part of the characterstring returned, they are used purely to delineate the values returned.

- “`true`” – The characterstring “`true`” shall be returned if the LMS accepts the content of `parameter_2` to set the value of `parameter_1`.
- “`false`” – The characterstring “`false`” shall be returned if the LMS encounters an error in setting the contents of `parameter_1` with the value of `parameter_2`. The SCO may call `GetLastError()` to determine the type of error. More detailed information pertaining to the error may be provided by the LMS through the `GetDiagnostic()` function.

3.1.4.3 Commit

Method Syntax: `return_value = Commit(parameter)`

Description: The method requests forwarding to the persistent data store any data from the SCO that may have been cached by the API Instance since the last call to `Initialize(“”)` or `Commit(“”)`, whichever occurred most recently. The LMS would then set the error code to `0` (No Error encountered) and return “`true`”.

If the API Instance does not cache values, `Commit(“”)` shall return “`true`” and set the error code to `0` (No Error encountered) and do no other processing.

Cached data shall not be modified because of a call to the commit data method. For example, if the SCO sets the value of a data model element, then calls the commit data method, and then subsequently gets the value of the same data model element, the value returned shall be the value set in the call prior to invoking the commit data method. The `Commit("")` method can be used as a precautionary mechanism by the SCO. The method can be used to guarantee that data set by the `SetValue()` is persisted to reduce the likelihood that data is lost because the communication session is interrupted, ends abnormally or otherwise terminates prematurely prior to a call to `Terminate("")`.

Parameter: ("") – empty characterstring. An empty characterstring shall be passed as a parameter.

Return Value: The method can return one of two values. The return value shall be represented as a characterstring. The quotes ("") are not part of the characterstring returned, they are used purely to delineate the values returned.

- "`true`" – The characterstring "`true`" shall be returned if the data was successfully persisted to a long-term data store.
- "`false`" – The characterstring "`false`" shall be returned if the data was unsuccessfully persisted to a long-term data store. The API Instance shall set the error code to a value specific to the error encountered. The SCO may call `GetLastError()` to determine the type of error. More detailed information pertaining to the error may be provided by the LMS through the `GetDiagnostic()` function.

3.1.5. Support Methods

Support methods exist within the API to allow a SCO to determine error handling and diagnostic information. With each API function described so far, and only those described so far, error conditions may occur. When these error conditions are encountered the error code is changed to indicate the error encountered. The calls to support methods do not affect the error state. In other words, calling any of the support methods shall not change the current error code. These support methods allow the SCO to determine if an error occurred and how to handle any error conditions encountered.

3.1.5.1 GetLastError

Method Syntax: `return_value = GetLastError()`

Description: This method requests the error code for the current error state of the API Instance. If a SCO calls this method, the API Instance shall not alter the current error state, but simply return the requested information.

A best practice recommendation is to check to see if a Session Method or Data-transfer Method was successful. The `GetLastError()` can be used to return the current error

code. If an error was encountered during the processing of a function, the SCO may take appropriate steps to alleviate the problem.

Parameter: The API method shall not accept any parameters.

Return Value: The API Instance shall return the error code reflecting the current error state of the API Instance. The return value shall be a characterstring (convertible to an integer in the range from 0 to 65536 inclusive) representing the error code of the last error encountered.

3.1.5.2 GetErrorString

Method Syntax: `return_value = GetErrorString(parameter)`

Description: The `GetErrorString()` function can be used to retrieve a textual description of the current error state. The function is used by a SCO to request the textual description for the error code specified by the value of the `parameter`. The API Instance shall be responsible for supporting the error codes identified in Section 3.1.7: *API Implementation Error Codes*. This call has no effect on the current error state; it simply returns the requested information.

Parameter:

- `parameter`: Represents the characterstring of the error code (integer value) corresponding to an error message.

Return Value: The method shall return a textual message containing a description of the error code specified by the value of the `parameter`. The following requirements shall be adhered to for all return values:

- The return value shall be a characterstring that has a maximum length of 255 characters.
- SCORM makes no requirement on what the text of the characterstring shall contain. The error codes themselves are explicitly and exclusively defined. The textual description for the error code is LMS specific.
- If the requested error code is unknown by the LMS, an empty characterstring (“”) shall be returned. This is the only time that an empty characterstring shall be returned.

3.1.5.3 GetDiagnostic

Method Syntax: `return_value = GetDiagnostic(parameter)`

Description: The `GetDiagnostic()` function exists for LMS specific use. It allows the LMS to define additional diagnostic information through the API Instance. This call has no effect on the current error state; it simply returns the requested information.

Parameter:

- `parameter`: An implementer–specific value for diagnostics. The maximum length of the parameter value shall be 255 characters. The value of the `parameter` may be an error code, but is not limited to just error codes.

Return Value: The API Instance shall return a characterstring representing the diagnostic information. The maximum length of the characterstring returned shall be 255 characters. If the parameter is unknown by the LMS, an empty characterstring (“”) shall be returned.

If the `parameter` passed into the `GetDiagnostic()` function is an empty characterstring (“”), then it is recommended that the function return a characterstring representing diagnostic information about the last error encountered.

3.1.6. Communication Session State Model

The IEEE defines a conceptual state model that the API Instance transitions through during its existence. Figure 3.1.6a, describes the states that an API Instance transitions through for a given SCO at run-time. The states of the API Instance specify the transitions of the API Instance to specific events. Each of the defined API Instance states defines which functions a SCO may invoke. The states encountered by the API Instance are defined as:

- `Not Initialized`
- `Running`
- `Terminated`

An implementation is not required to implement a state model. The state model is just a conceptual model used to help illustrate the intended behavior of the API functions during a typical communication session.

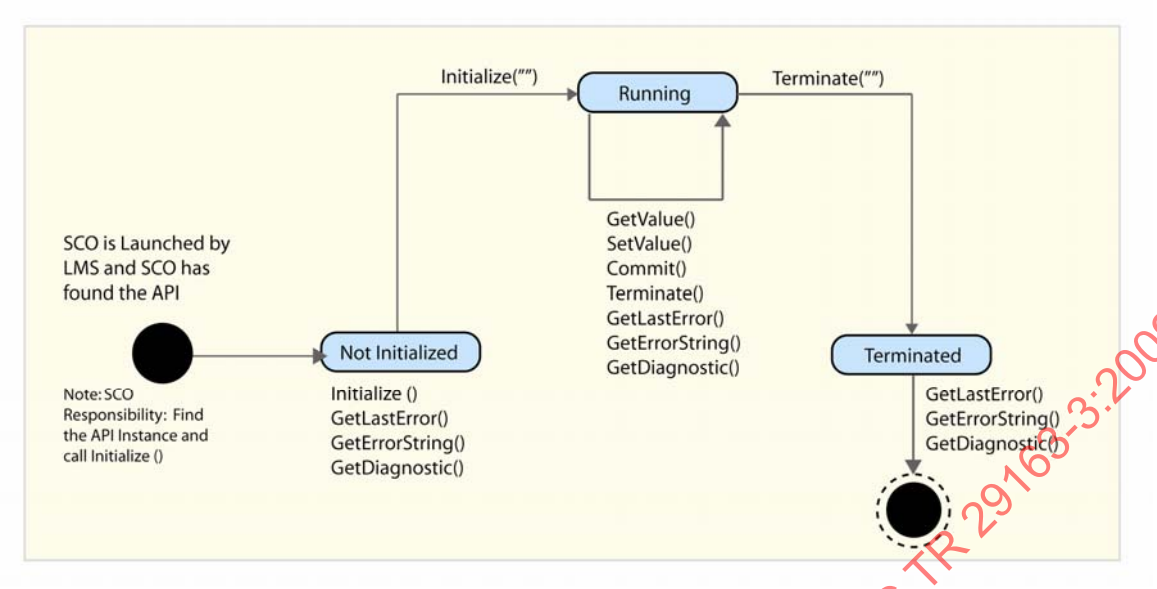

Figure 3.1.6a: Conceptual API Instance State Transitions

Not Initialized: This describes the conceptual communication state between the actual launching of the SCO and before the `Initialize("")` API method is successfully invoked by the SCO. During this state, it is the SCO's responsibility to find the API Instance provided by the LMS. The SCO is permitted to call the following set of API functions:

- `Initialize()`
- `GetLastError()`
- `GetErrorString()`
- `GetDiagnostic()`

Running: This describes the conceptual communication state once the `Initialize("")` API method is successfully invoked by the SCO and before the `Terminate("")` API method call is successfully invoked by the SCO. The SCO is permitted to call the following set of API functions:

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

Terminated: This describes the conceptual communication state once the `Terminate("")` API method is successfully invoked. The SCO is permitted to call the following set of API functions:

- `GetLastError()`
- `GetErrorString()`

- `GetDiagnostic()`

3.1.7. API Implementation Error Codes

All error codes are required to be integers represented as characterstrings. The IEEE standard [2] requires that all error codes be in the range of 0 to 65536 inclusive. The standard has also reserved the range of 0 to 999 inclusive for future editions of the standard. Additional error codes may be defined by implementations and profiles in the range of 1000 to 65535. SCORM does not define any additional error codes to be used for error conditions. This does not preclude implementations for defining error codes and using those codes in implementation-defined practices. SCORM only requires the use of the given error code in the defined error conditions (defined in this section).

Every API function, except for Support Methods: `GetLastError()`, `GetErrorString()` and `GetDiagnostic()`, sets the currently maintained error code of the API Instance. The SCO may invoke the `GetLastError()` function to assess whether or not the most recent API function call was successful, and if it was not successful, what went wrong. The `GetLastError()` function returns an error code that can be used to determine the type of error raised, if any, by the most recent API function call.

The IEEE has defined the following categories and numeric ranges for the various error codes:

Table 3.1.7a: Error Code Categories and Range

Error Code Category	Error Code Range
No Error	0
General Errors	100 – 199
Syntax Errors	200 – 299
RTS Errors	300 – 399
Data Model Errors	400 – 499
Implementation-defined Errors	1000 - 65535

3.1.7.1 Successful API Function Invocation

If, during the execution of an API function, no errors are encountered, the LMS shall set the API Instance's error code to 0. This indicates that the API function invocation encountered no errors while processing the request.

If after an API function call, the error code is 0, one can assume the following based on the actual function called:

- `Initialize("")`: The API Instance has successfully performed the appropriate LMS specific communication session initialization procedures. The communication session has been established and the API Instance is ready for other function calls. The conceptual communication state of the API Instance is now "`Running`".

- `GetValue(parameter)`: The requested data model element's value is returned. The value from the request shall be considered reliable and accurate according to the LMS. The conceptual communication state has not changed.
- `SetValue(parameter_1, parameter_2)`: The value passed in as `parameter_2` of the `SetValue()` was successfully set (stored as the value associated with the data model element described by parameter_1) by the LMS. A request to `GetValue()` for the data model element used in the `SetValue()`, `parameter_1`, shall return the value that was stored by the LMS. The conceptual communication state has not changed.
- `Commit("")`: Any values that were set (using the `SetValue()` method call) since `Initialize("")` or the last `Commit("")` method call, have been successfully forwarded to the persistent data store. This method guarantees that the data will be available during subsequent learner sessions within the same learner attempt with the SCO. The conceptual communication state has not changed.
- `GetLastError()`, `GetErrorString()` and `GetDiagnostic()`: These API methods do not affect or alter the error code for the API Instance.
- `Terminate("")`: The API Instance has successfully performed the appropriate LMS specific communication session termination procedures. The communication session has ended. The conceptual communication state of the API Instance is now "`Terminated`".

3.1.7.2 General Error Codes

The General Error codes describe errors that are encountered during the processing of API method requests. These errors are used based on several conditions:

- Current state of the conceptual communication state model
- Type of API request being processed by the API Instance

3.1.7.2.1 General Exception (101)

The General Exception error condition indicates that an exception occurred and no other specific error code exists. The API Instance shall use the General Exception error code in scenarios where a more specific error code is not defined.

3.1.7.2.2 General Initialization Failure (102)

The General Initialization Failure error condition indicates that a failure occurred while attempting to initialize the communication session. The General Initialization Failure error code shall be used by an API Instance when the communication session initialization process fails while the conceptual communication state is "`Not Initialized`" and no other specific error code exists. The API Instance shall set the error code to `102` and return "`false`" to the SCO. The conceptual communication state shall remain unchanged ("`Not Initialized`").

3.1.7.2.3 Already Initialized (103)

The Already Initialized error condition indicates that the SCO attempted to initialize the communication session after the communication session has already been initialized successfully. The Already Initialized error code shall be used by an API Instance when the SCO attempts to initialize the communication session (`Initialize("")`) more than once during that communication session. This error code shall be used when the conceptual communication state is "`Running`" and the request is made to initialize the communication session. In this scenario, the API Instance shall set the error code to `103` and return a "`false`" to the SCO. The conceptual communication state shall remain unchanged ("`Running`").

3.1.7.2.4 Content Instance Terminated (104)

The Content Instance Terminated error condition indicates that the communication session has already terminated. This error condition occurs when a SCO attempts to invoke the `Initialize("")` method after a successful call to the `Terminate("")` method has occurred. This error code shall be used when the conceptual communication state is "`Terminated`" and the request to initialize the communication session occurs. In this scenario, the API Instance shall set the error code to `104` and return a "`false`" to the SCO. The conceptual communication state shall remain unchanged ("`Terminated`").

3.1.7.2.5 General Termination Failure (111)

The General Termination Failure error condition indicates a failure occurred while attempting to terminate the communication session. The General Termination Failure error code shall be used by an API Instance when the communication session termination process fails while the conceptual communication state is "`Running`" and no other error information is available (i.e., a more specific communication session termination error condition). The API Instance shall set the error code to `111` and return "`false`" to the SCO. The conceptual communication state shall remain unchanged ("`Running`").

3.1.7.2.6 Termination Before Initialization (112)

The Termination Before Initialization error condition indicates that the SCO attempted to terminate the communication session before the communication session was ever initialized (conceptual communication state is "`Not Initialized`"). The Termination Before Initialization error code shall be used by an API Instance when the SCO tries to invoke `Terminate("")` prior to a successful call to `Initialize("")`. The API Instance shall set the error code to `112` and return "`false`" to the SCO. The conceptual communication state shall remain unchanged ("`Not Initialized`").

3.1.7.2.7 Termination After Termination (113)

The Termination After Termination error condition indicates that the SCO attempted to terminate the communication session after the communication session has already been terminated successfully. The Termination After Termination error code shall be used by an API Instance when the SCO has invoked the `Terminate("")` method after a previous

`Terminate("")` method has already been processed successfully. The API Instance shall set the error code to `113` and return "`false`" to the SCO. The conceptual communication state shall remain unchanged ("`Terminated`").

3.1.7.2.8 Retrieve Data Before Initialization (122)

The Retrieve Data Before Initialization error condition indicates that the SCO attempted to retrieve data prior to a successful communication session initialization. The Retrieve Data Before Initialization error code shall be used by an API Instance when the SCO attempts to invoke the `GetValue()` method prior to a successful call to the `Initialize("")` method. The API Instance shall set the error code to `122` and return an empty characterstring (""). The conceptual communication state shall remain unchanged ("`Not Initialized`").

3.1.7.2.9 Retrieve Data After Termination (123)

The Retrieve Data After Termination error condition indicates that the SCO attempted to retrieve data after the communication session has successfully terminated. The Retrieve Data After Termination error code shall be used by an API Instance when the SCO attempts to invoke the `GetValue()` method after a successful call to the `Terminate("")` method. The API Instance shall set the error code to `123` and return an empty characterstring (""). The conceptual communication state shall remain unchanged ("`Terminated`").

3.1.7.2.10 Store Data Before Initialization (132)

The Store Data Before Initialization error condition indicates that the SCO attempted to store data prior to a successful communication session initialization. The Store Data Before Initialization error code shall be used by an API Instance when the SCO attempts to invoke the `SetValue()` method prior to a successful call to the `Initialize("")` method. The API Instance shall set the error code to `132` and return "`false`". The conceptual communication state shall remain unchanged ("`Not Initialized`").

3.1.7.2.11 Store Data After Termination (133)

The Store Data After Termination error condition indicates that the SCO attempted to store data after the communication session has successfully terminated. The Store Data After Termination error code shall be used by an API Instance when the SCO attempts to invoke the `SetValue()` method after a successful call to the `Terminate("")` method. The API Instance shall set the error code to `133` and return "`false`". The conceptual communication state shall remain unchanged ("`Terminated`").

3.1.7.2.12 Commit Before Initialization (142)

The Commit Before Initialization error condition indicates that the SCO attempted to commit data to persistent storage prior to a successful communication session initialization. The Commit Before Initialization error code shall be used by an API Instance when the SCO attempts to invoke the `Commit("")` method prior to a successful

call to the `Initialize("")` method. The API Instance shall set the error code to `142` and return "`false`". The conceptual communication state shall remain unchanged ("`Not Initialized`").

3.1.7.2.13 Commit After Termination (143)

The Commit After Termination error condition indicates that the SCO attempted to commit data to persistent storage after the communication session has successfully terminated. The Commit After Termination error code shall be used by an API Instance when the SCO attempts to invoke the `Commit("")` method after a successful call to the `Terminate("")` method. The API Instance shall set the error code to `143` and return "`false`". The conceptual communication state shall remain unchanged ("`Terminated`").

3.1.7.3 Syntax Error Codes

The Syntax Error codes describe error conditions that are relevant to the syntax of the API methods. At this time, the IEEE standard has defined one error code dealing with syntax specific error conditions. The following section describes the defined error condition and its usage scenarios.

3.1.7.3.1 General Argument Error (201)

The General Argument Error error condition indicates that an attempt was made to pass an invalid argument to one of the API functions, and no other defined error condition can be used to describe the error. Data Model errors should be used to specify a more specific error condition , if one occurs. If no other error code can be used to describe the error condition, the API Instance should use error code `201`. One scenario where this error code shall be used occurs when parameters are passed to the following API calls:

- `Initialize("")`
- `Terminate("")`
- `Commit("")`

All three of these API calls have a restriction that an empty characterstring parameter is passed to it. If a SCO passes any other argument to these function calls the API Instance shall return "`false`" and set the error code to `201`. The conceptual communication state shall remain unchanged.

3.1.7.4 RTS Error Codes

The RTS Error codes describe error conditions that are relevant to an implementation of an RTS. At this time, the IEEE standard has defined three error codes dealing with run-time specific error conditions. The following sections describe these defined error conditions and their usage scenarios.

3.1.7.4.1 General Get Failure (301)

The General Get Failure error condition indicates a general get failure has occurred and no other information on the error is available (more specific error code). This error condition acts as a catch all condition for processing a `GetValue()` request. The General Get Failure error code shall be used by an API Instance when a retrieve data event (`GetValue()`) error has occurred and the API Instance cannot determine a more specific error condition to report (more specific error code). The API Instance shall set the error code to `301` and return an empty characterstring (“”). This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

3.1.7.4.2 General Set Failure (351)

The General Set Failure error condition indicates a general set failure has occurred and no other information on the error is available (more specific error code). This error condition acts as a catch-all condition for processing a `SetValue()` request. The General Set Failure error code shall be used by an API Instance when a store data event (`SetValue()`) error has occurred and the API Instance cannot determine a more specific error condition to report (more specific error code). The API Instance shall set the error code to `351` and return “`false`”. This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

3.1.7.4.3 General Commit Failure (391)

The General Commit Failure error condition indicates a general commit failure has occurred and no other information on the error is available (more specific error code). This error condition acts as a catch all condition. The General Commit Failure error code shall be used by an API Instance when a commit data event (`Commit(“”)`) error has occurred and the API Instance cannot determine a more specific error condition to report (more specific error code). The API Instance shall set the error code to `391` and return “`false`”. This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

3.1.7.5 Data Model Error Codes

One of the features of the API is to allow content to communicate data to and retrieve data from an LMS. During the processing of such events, certain error conditions may be encountered. The IEEE standard has defined several error conditions that describe general-purpose data model errors. The following sections describe these defined error conditions and their usage scenarios.

3.1.7.5.1 Undefined Data Model Element (401)

The Undefined Data Model Element error condition indicates that:

- The data model element passed as the `parameter` in the `GetValue(parameter)` is undefined and not recognized by the API Instance. This condition indicates that an attempt was made to use a data model element that is not recognized by the API Instance.
- The data model element passed as `parameter_1` in the `SetValue(parameter_1, parameter_2)` is undefined and not recognized by the API Instance. This condition indicates that an attempt was made to use a data model element that is not recognized by the API Instance.

An unrecognized or undefined data model element is any element that is not formally defined by the SCORM Run-Time Environment Data Model or an implementation-defined extension element that is not recognized by the API Instance. The Undefined Data Model Element error code shall be used by an API Instance when one of the two scenarios described above is encountered. The API Instance shall:

- For a `GetValue()` request: set the error code to `401` and return an empty characterstring(""). This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").
- For a `SetValue()` request: set the error code to `401` and return "`false`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

3.1.7.5.2 Unimplemented Data Model Element (402)

The Unimplemented Data Model Element error condition indicates that:

- The data model element passed as `parameter` in the `GetValue(parameter)` is recognized by the API Instance but is not implemented.
- The data model element passed as `parameter_1` in the `SetValue(parameter_1, parameter_2)` is recognized by the API Instance but is not implemented.

All of the SCORM Run-Time Environment Data Model elements are required to be implemented by an LMS. This error condition shall not occur when accessing SCORM Run-Time Environment Data Model elements, but may occur when accessing extension data model elements.

3.1.7.5.3 Data Model Element Value Not Initialized (403)

The Data Model Element Value Not Initialized error condition indicates that a SCO attempted to retrieve a data model value that has never been initialized. A data model element may be initialized in several manners:

- By the LMS: Some data model elements are initialized by values defined in a Content Package. Some data model elements may be initialized by some learner registration process or learner profiling requirements defined by the LMS.
- By the SCO: Some data model elements are initialized by the SCO.

The Data Model Element Value Not Initialized error code shall be used by an API Instance when the SCO attempts to invoke the `GetValue()` on an element that has no initial value. The API Instance shall set the error code to `403` and return an empty characterstring (“”). The empty characterstring (“”) value may be a valid value for a data model element request. Therefore, the value returned may not be reliable and the SCO should check the error code to determine whether the value is reliable. If the error code was set to `0` – No Error, then this indicates that the empty characterstring was the current value, stored by the LMS, for the data model element requested. This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

3.1.7.5.4 Data Model Element Is Read Only (404)

The Data Model Element Is Read Only error condition indicates that a SCO attempted to store a data model value for an element that is implemented as read-only. This error condition may be encountered with the use of the SCORM Run-Time Environment Data Model, SCORM Navigation Data Model or any other data model used in a SCORM environment (extension data model). The Data Model Element Is Read Only error code shall be used by an API Instance when the SCO attempts to invoke the `SetValue()` on a read-only data model element. The API Instance shall set the error code to `404` and return “`false`”. This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

3.1.7.5.5 Data Model Element Is Write Only (405)

The Data Model Element Is Write Only error condition indicates that a SCO attempted to retrieve a data model value for an element that is implemented as write-only. This error condition may be encountered with the use of the SCORM Run-Time Environment Data Model, SCORM Navigation Data Model or any other data model used in a SCORM environment (extension data model). The Data Model Element Is Write Only error code shall be used by an API Instance when the SCO attempts to invoke the `GetValue()` on a write-only data model element. The API Instance shall set the error code to `405` and return an empty characterstring (“”). This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

3.1.7.5.6 Data Model Element Type Mismatch (406)

The Data Model Element Type Mismatch error condition indicates that a SCO attempted to store a data model value for a data model element and the value was not of the correct data type. This error condition may be encountered with the use of the SCORM Run-

Time Environment Data Model, SCORM Navigation Data Model or any other data model used in a SCORM environment (extension data model). The Data Model Element Type Mismatch error code shall be used by an API Instance if the value passed as `parameter_2` in a `SetValue()` does not evaluate to a valid type or defined format for the data model element indicated in `parameter_1` of a `SetValue()`. The API Instance shall set the error code to `406` and return "`false`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

3.1.7.5.7 Data Model Element Value Out Of Range (407)

The Data Model Element Value Out Of Range error condition indicates that a SCO attempted to store a data model value for an element, however the value was not in the specified range of values for the element. This error condition may be encountered with the use of the SCORM Run-Time Environment Data Model, SCORM Navigation Data Model or any other data model used in a SCORM environment (extension data model). The Data Model Element Value Out Of Range error code shall be used by an API Instance if the value passed as `parameter_2` in a `SetValue()` is out of range for the data model element indicated in `parameter_1` of a `SetValue()`. The API Instance shall set the error code to `407` and return "`false`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

3.1.7.5.8 Data Model Dependency Not Established (408)

The Data Model Dependency Not Established error condition shall be used when relevant dependencies are not in place. A dependency represents one or more key values in a data model that shall have been set prior to other data model elements. The dependencies data model elements for the SCORM Run-Time Environment Data Model are described in Section 4: *SCORM® Run-Time Environment Data Model*.

This error condition is described by the IEEE standard as being used for situations that may arise during a `GetValue()` or `SetValue()` request, however, SCORM does not define any situations for use of this error code during the processing of `GetValue()` requests. For `SetValue()` requests, some data model elements have requirements that certain other data model elements be set prior to other data model elements. By setting elements in a specific order, this maintains the integrity of a dependency being met. If one of these dependency requirements have not been established the LMS shall set the API Instance error code to `408` and return "`false`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

3.1.7.6 SCORM Extension Error Conditions

Due to the nature of the SCORM Run-Time Environment Data Model elements and the binding of the data model elements (dot-notation), SCORM defines extension error conditions to cover error scenarios that may occur within a SCORM environment. There

is no mechanism, defined by the IEEE standard, to permit extension error codes to be set by an API Instance. In the following sections, SCORM defines a set of error conditions. If these error conditions are encountered, API Instance shall behave as follows:

- Set the error code to `301` (for `GetValue()` failures) or `351` (for `SetValue()` failures), and return "`false`".
- If requested by a SCO to return more information about the error encountered (`GetDiagnostic()`), it is recommended that the LMS return information detailing the error conditions that follow.

3.1.7.6.1 Data Model Element Does Not Have Children

The Data Model Element Does Not Have Children error condition indicates that a SCO attempted to retrieve a list of supported data model elements (children) for a data model element that does not have children (refer to the SCORM Run-Time Environment Data Model for information regarding the data model elements). The Data Model Element Does Not Have Children error condition should be used by an API Instance if the value passed as parameter in `GetValue()` is a request for a listing of child elements for a data model element that does not have any children (refer to the SCORM Run-Time Environment Data Model for more information on processing of children requests). The API Instance shall set the error code to `301` and return an empty characterstring (""). This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

Example: `GetValue("cmi.learner_name.children");`

For this example the API Instance shall set the error code to `301` and return an empty characterstring (""). The `learner_name` element is considered a child element and does not have any children. If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that the data model element does not have children, such as, "The data model element does not have children".

3.1.7.6.2 Data Model Element Cannot Have Count

The Data Model Element Cannot Have Count error condition indicates that a SCO attempted to retrieve the number of entries (count) currently stored in a data model element that is not an array (refer to the SCORM Run-Time Environment Data Model for data model elements that are arrays and array handling). The Data Model Element Cannot Have Count error condition shall be used by an API Instance if the value passed as `parameter` in a `GetValue()` is a request for the number of entries (count) currently stored in a data model element that is not an array. The API Instance should set the error code to `301` and return an empty characterstring (""). This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

Example:

- `GetValue("cmi.learner_name._count");`

For this example the API Instance shall set the error code to `301` and return an empty characterstring (""). The `learner_name` element is not a collection and therefore does not have a count. If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that the data model element is not an array and does not have a count, such as, "The data model element is not a collection and therefore does not have a count".

3.1.7.6.3 Data Model Element Collection Set Out Of Order

The Data Model Element Collection Set Out Of Order error condition indicates that a SCO attempted to set a value in an array where the index number used (*n*) is not the next available position in the array. All collections (refer to Section 4.1.1.3: *Handling Collections*) are required to be packed arrays (no skipped positions). When a new value is added to the array it must be in the next available position. The SCO can determine this position by using the `_count` keyword data model element (refer to Section 4.1.1.5: *Keyword Data Model Elements*).

The Data Model Element Collection Set Out Of Order error condition should be used by an API Instance if the value for the index position (*n*) passed as `parameter_1` in a `SetValue()`, for a new entry in the array, is not the next available position in the array. The API Instance shall set the error code to `351` and return "`false`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

Example:

- `SetValue("cmi.objectives.0.id","identifier_1");`
- `SetValue("cmi.objectives.2.id","identifier_2");`

For this example the API Instance shall set the error code to `351` and return "`false`". The second request is attempting to set the objective identifier in position 2, prior to position 1, in the array. If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that the data model element collection was set out of order, such as, "The data model element collection was attempted to be set out of order".

3.1.7.6.4 Data Model Collection Element Request Out Of Range

The Data Model Collection Element Request Out Of Range error condition indicates that a SCO attempted to retrieve a data model element found in a collection and the index value provided is larger than what is currently being maintained by the LMS. All collections (refer to Section 4.1.1.3: *Handling Collections*) are required to be packed arrays (no skipped positions). The SCO can determine the current number of data model

element values being stored by using the `_count` keyword data model element (refer to Section 4.1.1.5: *Keyword Data Model Elements*).

If the current data model collection only contains four values, this implies that the index positions that are accessible by the SCO are 0, 1, 2, and 3. If the SCO attempts to retrieve a value from a position with an index (n) greater than 3, the API Instance shall set the error code to `301` and return an empty characterstring "". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that the data model collection element does not exist due to an index out of range, such as, "The data model element request failed to be processed due to an index out of range error".

3.1.7.6.5 Data Model Element Not Specified

The Data Model Element Not Specified error condition indicates that a SCO attempted to make a `GetValue()` or `SetValue()` API method call without providing any indication of the data model element to retrieve or store. Both of the API method calls require the presence of a data model element:

- `GetValue(parameter_1)`
- `SetValue(parameter_1, parameter_2)`

The Data Model Element Not Specified error condition should be used by an API Instance if the value passed as `parameter_1` in a `GetValue()` request or `parameter_1` in a `SetValue()` request is not specified (i.e., empty characterstring – "").

- `GetValue("")`: The API Instance should set the error code to `301` – General Get Failure and return an empty characterstring ("").
- `SetValue("","3.4")`: The API Instance should set the error code to `351` – General Set Failure and return "`false`".

This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`"). If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that the data model element was not specified, such as, "The data model element was not specified".

3.1.7.6.6 Unique Identifier Constraint Violated

The Unique Identifier Constraint Violated error condition should be used by an API Instance if the value passed as `parameter_2` in a `SetValue()` request is to be used to set an identifier for which the identifier is required to be unique. The API Instance should set the error code to `351` and return "`false`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

Example:

- API Call 1: `SetValue("cmi.objectives.0.id","objective1)`
- API Call 2: `SetValue("cmi.objectives.1.id","objective2")`
- API Call 3: `SetValue("cmi.objectives.2.id","objective1")`

In the example above, the identifier for the objective data stored in position `2` of the array is trying to set the identifier of the objective to `objective1`. This identifier was used for the objective data stored at position `0` of the array. Since the cmi.objectives.n.id, is required to be a unique identifier, this would cause a General Set Failure error. The LMS shall not set the requested element, return "`false`" and set the error code to `351`. If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that there was a unique identifier constraint violated, such as, "The data model element's value is already in use and is not unique".

3.1.7.6.7 Smallest Permitted Maximum Exceeded

The Smallest Permitted Maximum (SPM) Exceeded error condition should be used in two situations:

- if the value passed as `parameter_2` in a `SetValue()` request is to be used to set a data model element that has a SPM and the length of `parameter_2` exceeds the SPM for the data model element, or
- if the value is being placed in a collection and the collection's SPM has been exceeded

These situations do not necessarily describe "error conditions", however depending on the implementation, these situations may be handled differently. In both situations described above, the API Instance should set the error code to `0` and return "`true`". This error condition can only happen if the conceptual communication state is "`Running`". If this error condition is encountered, the conceptual communication state shall remain unchanged ("`Running`").

If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that an SPM was violated, such as, "The value to be used to set the data model element exceeds the SPM for the data model element. The value was truncated."

The SPMs defined represent implementation storage requirements. An implementation has to support the storage of at least the SPM. An implementation may elect to support the storage of more than the SPM. If an implementation only supports the SPM and performs a truncation of the characterstring, then ADL recommends that the implementation make information available through the GetDiagnostic() method call indicating that truncation of a characterstring has occurred.

3.1.7.6.8 Data Model Element Does Not Have Version

The Data Model Element Does Not Have Version error condition indicates that a SCO attempted to retrieve the version of the data model (`_version`) on a data model element for which it is not permitted. The `_version` keyword should only be applied to the base data model identified by the SCORM Run-Time Environment Data Model (`cmi._version`). The Data Model Element Does Not Have Version error condition shall be used by an API Instance if the value passed as `parameter` in a `GetValue()` is something other than `cmi._version` (e.g., `cmi.learner_id._version`). The API Instance should set the error code to `301` and return an empty characterstring (“”). This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that the version request is invalid, such as, “The _version keyword was used incorrectly”.

3.1.7.6.9 Identifier Value Can Only Be Set Once

The Identifier Value Can Only Be Set Once error condition should be used by an API Instance if the value passed as `parameter_2` in a `SetValue()` request is to be used to set an identifier to a value different than the value that has already been set. The API Instance should set the error code to `351` and return “`false`”. This error condition can only happen if the conceptual communication state is “`Running`”. If this error condition is encountered, the conceptual communication state shall remain unchanged (“`Running`”).

Example:

- API Call 1: `SetValue(“cmi.objectives.0.id”,”objective1)`
- API Call 2: `SetValue(“cmi.objectives.0.id”,”objective2)`

In the example above, the identifier for the objective data stored in position `0` of the array was initially set to `objective1`. API Call 2 is trying to set the identifier of the objective to `objective2`. Since the value `objective2` is different than the value that was already set, this would cause a General Set Failure error. The LMS shall not set the requested element, return “`false`” and set the error code to `351`. If the SCO requests further information about the error (by invoking `GetDiagnostic()`), it is recommended that the LMS return a characterstring indicating that there was a unique identifier constraint violated, such as, “The data model element’s value is already set and cannot be changed”.

3.2. LMS Responsibilities

SCORM requires the LMS to provide an instance of the API as defined by the IEEE standard and SCORM. The API Instance shields the SCO from the particular implementation details. SCORM does not place any restrictions on the underlying communication infrastructure of the API Instance. The following sections describe those additional requirements, not described thus far, of an API Instance for an LMS implementation.

3.2.1. API Instance

SCORM requires that an LMS supply an API Instance that implements the required API functionality described earlier. In order for a SCO to utilize the API Instance developed by an LMS, the LMS has certain requirements on where and how to provide access to the API Instance. To provide for an interoperable means to locate the API Instance, the LMS's API Instance must be accessible via the DOM [8] as an object named `API_1484_11`. The LMS must provide the ability for the SCO to access the API Instance via ECMAScript.

In order for SCOs to find the LMS-provided API Instance, the LMS is responsible for launching SCOs in a particular DOM hierarchy. The LMS shall launch the SCO in a browser window that is a child window or a child frame of the LMS window that contains the API Instance.

Figure 3.2.1a describes the valid locations in the DOM hierarchy that an LMS is permitted to place its API Instance. The algorithm defined by the IEEE standard is built to search for the API Instance in specific locations and in a specific order. The specific locations that an LMS can provide an API Instance is determined based on the implementation requirements of the LMS.

Chain of Parents: In this case, the SCOs and Assets are being launched in a structure where the API Instance is in an HTML Frameset. In this case, the LMS can provide the API Instance somewhere in the hierarchy of parent frames within the frameset. The algorithm described in the IEEE standard will search for the API by walking the "chain of parents" either until the API Instance is found or until it reaches a point where there are no other parent frames.

Opener: In this case, the SCOs and Assets are being launched in a new window. This new window is sometimes referred to as a "pop-up" window. The new window contains the SCO or Asset. In this case, the LMS can provide the API Instance in the opener window (i.e., the window that was responsible for launching, or opening, the new window in which the SCO or Asset was launched in). The algorithm described in the IEEE standard will determine if the SCO was launched by an opener window. If so, the opener will be searched for the API Instance.

Chain of Parents of the Opener: In this case, the SCOs and Assets are being launched in a new window and the LMS has placed the API Instance in a parent frame of the opener window. The algorithm described in the IEEE standard will determine if the SCO was launched in an opener window. If so, the opener window will be searched for the API Instance. If the API Instance is not found in the opener window, the algorithm determines if the opener window has a parent. The algorithm is designed to search the parent window hierarchy until it either finds the API Instance or determines that there are no more parent windows.

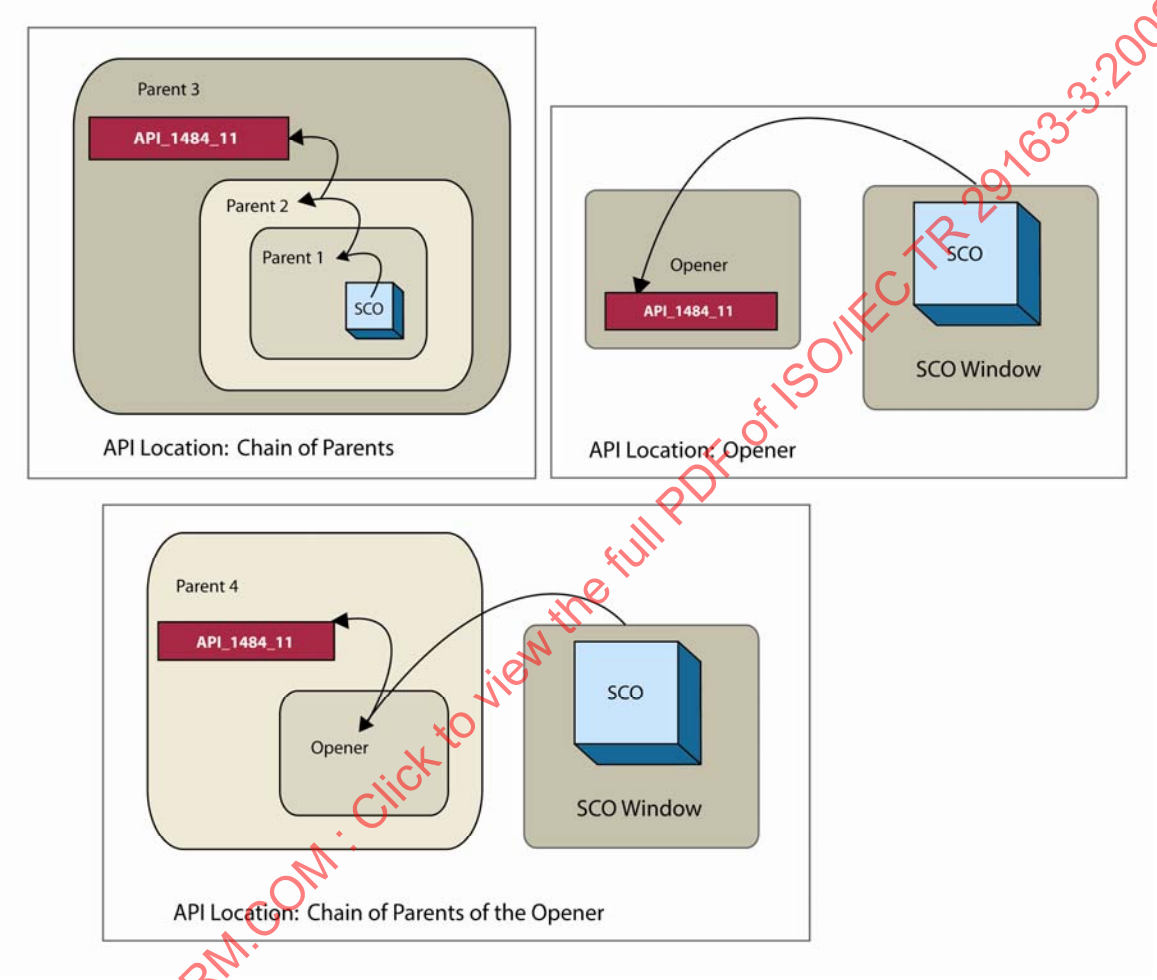

Figure 3.2.1a: Permitted DOM Locations of an API Implementation

There are ongoing research and development efforts investigating alternative methods (e.g., as a Web Service) for LMS vendors to provide SCOs access to the API Instance. However, SCORM only supports the method described above, the DOM and ECMAScript are reliable technologies that have been around for some time and are simple to use. Future versions of SCORM may introduce other communication protocols that support the fundamental requirements defined by the IEEE standard.

3.2.1.1 Version Attribute

The IEEE standard requires that the API Instance be accessible via a DOM Object. The standard also requires the DOM object to have an attribute named version. The value of the attribute represents the version of the API Instance. The value assigned to the attribute is required to consist of a period-delimited string containing major and minor release values as whole numbers. To indicate that the API Instance is built in conformance with the IEEE Standard, the first three characters are required to be `1.0.` If an implementation wants to append additional characters to the value, to represent internal versioning information, then the fourth character is required to be another period-delimiter (e.g., `1.0.`).

3.3. SCO Responsibilities

All SCOs have certain responsibilities when communicating across the API. SCOs must be able to consistently find the API Instance. This is one of the primary reasons why there are restrictions on where, in the DOM hierarchy, the LMS provides the API Instance and why there is a common name of the API Instance to locate. If the API Instance were allowed to exist anywhere in the DOM hierarchy, this would make it extremely difficult to provide a consistent communication mechanism and management of the run-time environment.

3.3.1. Finding the API Instance

In order for a SCO to begin tracking a learner's experience with an LMS, the SCO must be able to find the LMS provided API Instance. Since the content objects, in the SCORM environment, are launched in Web browsers, the Web browsers provide a DOM in which to place an API Instance. The DOM can be considered a defined structure or organization of the objects in a page. In order for SCOs to find the API Instance from one LMS to another, the IEEE standard has placed restrictions on where the API Instance can be placed in this hierarchy. The important fact is that the SCO must look in the following locations, in the order specified, for the API Instance:

1. The chain of parents of the current window, if any exist, until the top of the window of the parent chain is reached
2. The opener window, if any
3. The chain of parents of the opener window, if any exist, until the top window of the parent chain is reached

The SCO must search for the API Instance in this manner and stop as soon as an API instance is found. For the SCO to know what it is looking for, the IEEE standard has also defined a mandatory name for the object in the DOM that is associated with the API Implementation. The name defined for the API Implementation is `API_1484_11`.

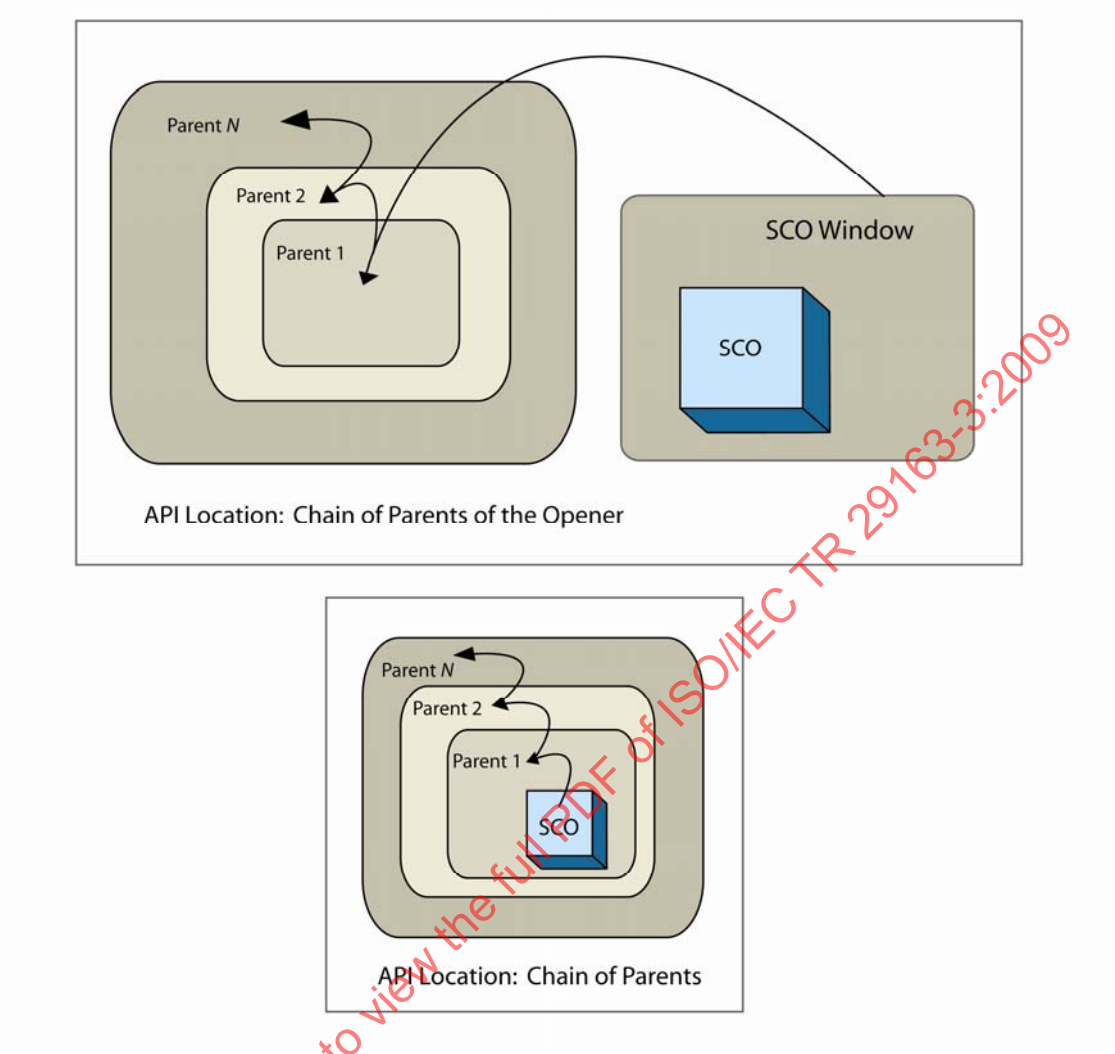

Figure 3.3.1a: Illustration of Finding the API

Once a SCO finds an API Instance, the SCO is required to, at a minimum, issue `Initialize("")` and `Terminate("")` API calls.

The IEEE standard has provided a simple piece of ECMAScript that will find the API Instance in a consistent manner. It is not a requirement to use this ECMAScript code. Other variations can be written.

```
var nFindAPITries = 0;
var API = null;
var maxTries = 500;
var APIVersion = "";

// The ScanForAPI() function searches for an object named API_1484_11
// in the window that is passed into the function.  If the object is
// found a reference to the object is returned to the calling function.
// If the instance is found the SCO now has a handle to the LMS
// provided API Instance.  The function searches a maximum number
// of parents of the current window.  If no object is found the
// function returns a null reference.  This function also reassigns a
// value to the win parameter passed in, based on the number of
// parents.  At the end of the function call, the win variable will be
// set to the upper most parent in the chain of parents.
function ScanForAPI(win)
{
   while ((win.API_1484_11 == null) && (win.parent != null)
           && (win.parent != win))
   {
      nFindAPITries++;
      if (nFindAPITries > maxTries)
      {
         return null;
      }
      win = win.parent;
   }
   return win.API_1484_11;
}

// The GetAPI() function begins the process of searching for the LMS
// provided API Instance.  The function takes in a parameter that
// represents the current window.  The function is built to search in a
// specific order and stop when the LMS provided API Instance is found.
// The function begins by searching the current window's parent, if the
// current window has a parent.  If the API Instance is not found, the
// function then checks to see if there are any opener windows.  If
// the window has an opener, the function begins to look for the
// API Instance in the opener window.
function GetAPI(win)
{
   if ((win.parent != null) && (win.parent != win))
   {
      API = ScanForAPI(win.parent);
   }
   if ((API == null) && (win.opener != null))
   {
      API = ScanForAPI(win.opener);
   }
   if (API != null)
   {
      APIVersion = API.version;
   }
}
```


Code Illustration 3.3.1b: Example ECMAScript for finding the API Implementation

This example ECMAScript code is based on the algorithm defined in the IEEE standard. The algorithm was updated to add narrative text describing the functions, removed an error message that is not necessary and updated to explain the win variable found in the algorithm in more of a context. Organizations are not required to use the ECMAScript

code as defined in IEEE or SCORM and are free to create their own ECMAScript. The organizations must make sure the algorithm searches for the API Instance as defined in the IEEE and SCORM.

3.3.2. API Usage Requirements and Guidelines

This section outlines several additional requirements and guidelines for using the API to communicate information to an LMS.

3.3.2.1 Unexpected events

SCOs are built in a variety of different ways. When developing a SCO, content authors need to be aware of the design of the SCO, how the SCO is intended to be delivered in an LMS, and the different ways a learner may interact with the SCO.

For example, some SCOs are built as a collection of pages, which allow internal SCO navigation from one page to another. In this design, some content authors may implement the SCO to invoke the `Initialize("")` call on the first page and `Terminate("")` only on the last page. What happens if there was an unexpected behavior that was encountered during the learning experience? The unexpected event may fall into several categories:

- Accidental exit
- Deliberate user action
- Catastrophic termination (e.g., lost connection, browser crash)

Some LMSs handle these different scenarios by simulating the effect of `Terminate("")` function if it detects that a SCO that successfully called `Initialize("")` became unexpectedly inaccessible before invoking the `Terminate()` function. This is done because there is no way to detect (or decide) the cause of the problem - was it an accident, a deliberate user action, or just a poorly built SCO? Of course, if there is a catastrophic event (lost connection, complete browser crash, etc.) then only the data that had been copied from a client side cache (if one is being used) through a `Commit("")` function survives in the server side database.

One of the main reasons why `Commit("")` is in the API is to minimize time-consuming communication delays between the client side of a runtime service and the server side that would occur if every data element value update was transmitted across the network in real time. Implementations of the API are free to provide a client side cache that only transmits and persists data state when the `Commit("")` function is invoked.

A SCO has no way to detect if the API Instance it is communicating through is providing a client side cache or sending all data updates to the server. To help minimize unexpected behaviors and problems that may be encountered, SCO developers should adhere to the following recommendations:

- Invoke `Commit("")` whenever something significant happens that must be recorded, regardless of what might happen later.
- Do NOT call `Commit("")` after every `SetValue()` call -- that defeats the performance enhancement and can lead to serious problems with some LMS implementations. DO call `Commit("")` only after a "batch" of `SetValue()` calls.
- There is typically no benefit in calling `Commit("")` immediately prior to calling `Terminate("")`. It does no harm since there would be nothing new to commit during the `Terminate("")` function call.
- Call `Terminate("")` prior to the SCO being taken away from the learner (i.e., unloaded from the browser). The SCO should keep track of whether it has successfully invoked `Terminate("")`, so the SCO can be built not to invoke `Terminate("")`again. A properly built LMS will be able to handle the call. It should ignore the second call and set the error status accordingly. If the client browser is Microsoft Internet Explorer, call `Terminate("")` in an `onbeforeunload` handler rather than in an `onunload` handler, because there is a much better chance that all the resulting work involved with unloading the SCO will actually happen in an orderly manner. Other browsers do not raise an event called `onbeforeunload` and therefore rely on the `onunload` event. It may be wise to provide a mechanism to terminate the communication session somewhere other than the `onunload` event.
- Use `SetValue()` and `Commit("")` on an ongoing basis during the communication session rather than trying to save a large amount of data immediately prior to `Terminate("")`. Do this because there is anecdotal evidence that some browsers or browser versions actually start loading the next page even while `onunload` is executing, and do not finish executing some operations triggered by `onunload` especially if the operations involve posting across a network. This could make for a difficult situation on the back end, and in some implementations, some data may not be committed properly or completely. If the data has been saved and committed before unload occurs, then this data is most likely safe.

SECTION 4
SCORM® Run-Time Environment Data Model

This page intentionally left blank.

4.1. Data Model Overview

The purpose of establishing a common data model is to ensure that a defined set of information about SCOs can be tracked by different LMS environments. If, for example, it is determined that tracking a learner's score is a general requirement, then it is necessary to establish a common way for content to report scores to LMS environments. If SCOs use a unique scoring representation, LMSs may not know how to receive, store or process the information.

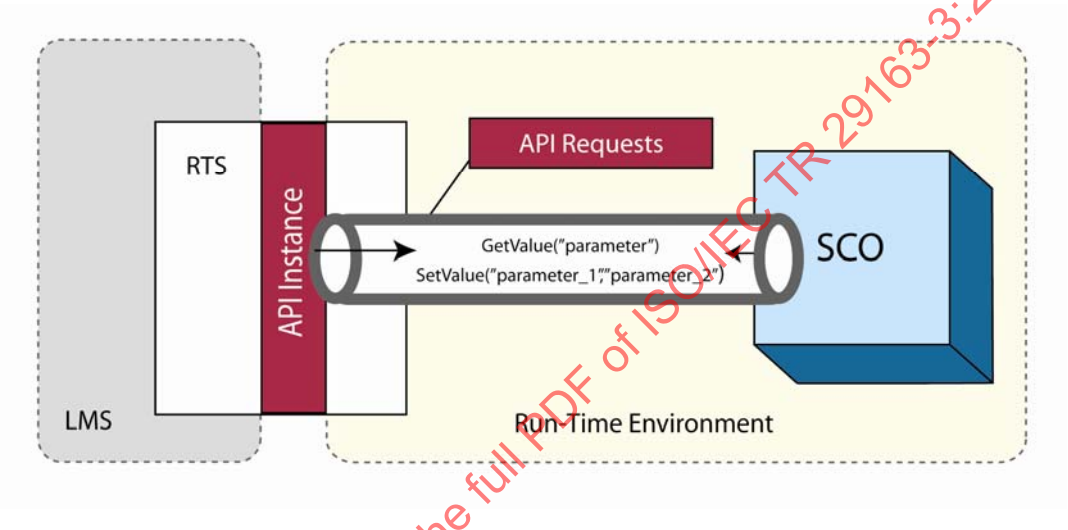

Figure 4.1.1a: Illustration of Using the Data Model with the API

The SCORM Run-Time Environment Data Model is based on the 1484.11.1 Standard for Learning Technology - Data Model for Content Object Communication [1] standard produced by the IEEE LTSC CMI. 1484.11.1 is a standard that defines a set of data model elements that can be used to communicate information from a content object (i.e., SCO in SCORM) to an LMS. This set of data includes, but is not limited to, information about the learner, interactions that the learner had with the SCO, objective information, success status and completion status. This information may be vital for many purposes. This data can be used to track the learner's progress and status, aid in sequencing decisions and report on the overall learner interaction with the SCO.

The data model in this section is defined as the SCORM Run-Time Environment Data Model. Prior to SCORM 2004, the SCORM Run-Time Environment Data Model was based on the AICC CMI001 Guideline for Interoperability [7]. Since the release of SCORM Version 1.2, AICC has submitted CMI001 to the IEEE for standardization. SCORM 2004 introduces the changes to the data model as defined by the IEEE 1484.11.1 Standard for Learning Technology Data Model for Content Object Communication [1]. Since the IEEE standard purely defines data model elements and their data types, SCORM needs to apply more requirements pertaining to the use, behavior and relationship with the API Instance. The SCORM Run-Time Environment Data Model

defines a particular binding (dot-notation), implementation guidance and behavioral requirements of the IEEE 1484.11.1 standard.

4.1.1. SCORM Run-Time Environment Data Model Basics

4.1.1.1 Data Model Elements

To identify the data model, all of the names of the data model elements described in the SCORM Run-Time Environment Data Model start with `cmi`. This signals to LMSs that these data model elements are part of the IEEE 1484.11.1 Data Model for Content Object Communication standard[1]. It is envisioned that as alternative data models are developed they will start with a different designation (e.g., adl.*elementName* instead of cmi.*elementName*) or may have a different binding other than the dot-notation.

All data model elements described by SCORM are required to be implemented and their behaviors supported by an LMS.

All data model elements are optional for use by SCOs. SCOs are required only to use the API functions `Initialize("")` and `Terminate("")`; they are not required to use `SetValue()` or `GetValue()`. SCOs may be very, very small and not designed to be tracked in detail. However, if they are to be tracked, they must conform to a common data model for reusability across multiple LMS environments.

All data model element names are bound to an ECMAScript characterstring using a dot-notation (e.g., `cmi.success_status`). During `SetValue()` method calls, all values to be used for setting the data model element are bound as ECMAScript characterstrings. The ECMAScript standard [9] supports and is in conformance with the Unicode Standard [13] (Version 2.1 or later). SCOs and LMSs need to be aware that since these characterstrings are Unicode encoded they may include Unicode escape sequences. When dealing with any data that may be rendered in the browser, SCOs must be aware of the level of support for the Unicode in the different browsers and versions of browsers.

4.1.1.2 Data Model Effects on Sequencing

SCORM Sequencing (refer to the SCORM SN book) describes how a series of content objects are identified for delivery based on defined sequencing information, sequencing behaviors and results of learner interaction with launched content objects. Assets have a limited effect on sequencing. An LMS only tracks the fact that an Asset is launched. Once the Asset has been launched, the Asset shall be considered “completed.” SCOs can affect sequencing by reporting the results of a learner’s interactions during a learner session with that SCO; this is done through the SCO’s Run-time Environment Data Model. An LMS is required to utilize information reported by the SCO, through the SCORM Run-time Environment Data Model, to affect the sequencing of subsequent learning activities. SCORM does not mandate a specific method or timing for how and when a SCO's run-time data is used for sequencing. SCORM prescribes only that the most recent information is used when a sequencing evaluation requires it. Specific LMS

requirements for this data mapping are provided in the following tables, broken down by each data model element (for more information refer to the *Sequencing Impacts* section of relevant tables).

For example, if the SCO reports the learner's completion of the SCO through `cmi.completion_status`, the activity identified with that SCO will be assumed to also be completed.

ADL Note: If the currently experienced content object is a SCO and the Learner Attempt on it ends due to an Abandon or Abandon All Navigation Request, no data mapping occurs. An abandoned Learner Attempt does not affect the state of the SCO's associated activity.

4.1.1.3 Handling Collections

Some data model elements collect sets of data related to their respective requirements. The collection of data is referred to as a record of data throughout this document. Each record of data shall be collected as a single entity in the array. The record of data is accessed by an index value representing the record of data's position in the array. All arrays shall be implemented with a starting index of 0 (zero-based arrays).

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

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

These data model elements exist with the intention that SCOs may track multiple comments, objectives and/or interactions.

There are two distinct error conditions that can occur when invoking a `GetValue()` request:

1. General Get Failure: Data Model Collection Element Request Out Of Range. The record of data does not exist at the array position (i.e., index) requested. For example, if data exists for objectives in array positions 0,1 and 2 and the SCO invokes a `GetValue()` request for objective information at position 4, the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring. It is recommended that the LMS provide to the SCO, if requested by a `GetDiagnostic()` request, more specific information about the error encountered (refer to Section 3.1.7.6.4: *Data Model Collection Element Request Out Of Range*)

   Another scenario that may happen is that a SCO attempts to create a new record of data in an array position. Some of the data model element collections contain requirements that certain data model elements be set first, which will allow the LMS to set up the record of data in the appropriate array position. If any of these

data model elements is set successfully, it would cause a newly created record of data to be added to the containing collection, increasing the collection's count by:

1. The following list identifies the set of data model elements that would cause the new record of data to be created:

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

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

   If an LMS receives a `GetValue()` request immediately following an invalid `SetValue()` request on one of these data model elements, then the behavior shall be:

   (1) Return an empty characterstring ("")

   (2) Set the API Error Code to `301 – General Get Failure`

2. Data Model Element Value Not Initialized. The record of data exists at the array position, however the data requested has not been initialized with a value yet. For example, if the SCO makes a request to get an objective's scaled score (`cmi.objectives.0.score.scaled`), however the scaled score was never initialized with data. In this case, the objective's record of data exists (i.e., the `cmi.objectives.0.id` has been set) at position 0, however the scaled score was never initialized. The LMS shall set the error code to `403` – Data Model Element Value Not Initialized and return an empty characterstring.

Because of the current dot-notation binding of the SCORM Run-Time Environment Data Model, these data model elements contain an integer value representing the index (or position) in the array. The index is just that, an index. Indices should not be considered unique for any given SCO, meaning that there is no guarantee from LMS to LMS (or learner session to learner session) that the same objective is stored at the same index. SCO developers should keep this in mind when using these data model elements.

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. This value should be used when searching for the index of a specific set of objective or interaction data. To retrieve values for a specific Objective or Interaction, it is recommended that the list of all of the SCO's Objectives or Interactions be searched

for a given identifier to determine its index rather than relying on a specific (hard-coded) index position. The index position is not guaranteed to be the same from learner session to learner session. All new collection elements shall be added sequentially. Indices used to access collection elements shall not have insignificant starting zeros – "15" not "0015". When a value is to be added to a collection, the SCO must determine the last index position used for that collection. The SCO and LMS shall not skip index positions (packed arrays) when constructing or appending to a collection element. The `_count` keyword data model element can be used to determine the current number of data model elements in the collection. For instance, to determine the number of objectives currently stored for the SCO, the following API call would be used:

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

All collection data model elements, besides `cmi.comments_from_lms` (this data model element is read-only from a SCO perspective), can have their sub-elements overwritten by the SCO. Overwriting or appending is a decision that is made by the SCO developer during the creation of the SCO.

Data model elements in a collection are referred to using a dot-number notation (represented by .n).

```
cmi.objective.n.completion_status
```

For instance, the value of the completion status data model element in the first objective in a SCO would be referred to as `"cmi.objective.0.completion_status"`, and the completion status data model element in the fourth objective would be referred to as `"cmi.objective.3.completion_status"`.

4.1.1.4 Smallest Permitted Maximum (SPM)

There are two cases when describing data model element requirements where there are SPMs defined. The two cases are for characterstring lengths and the number of data model elements contained in collections (arrays, bags, etc.). The SPM is defined as the minimum number of entries (in a collection) or characters (length of characterstrings) that an implementation must accept or process. Implementations are free to accept and process more than the SPM; however, they must support at least the SPM. For example,

- Characterstrings: If a characterstring is defined with an SPM of 100, then an implementation must accept and support at least 100 characters. Implementations may support more than the defined SPM. SCORM is silent on handling of characterstrings that contain more than the SPM number of characters (e.g., an implementation is free to truncate the characterstring). Because an implementation shall support at least the SPM, if an implementation truncates values, the implementation shall truncate at the SPM. Implementations should be aware of the consequences if the characterstring contains more than the defined SPM.

- Collections: If a collection (e.g., array, set, bag) is defined with an SPM of 100, then an implementation must accept and support at least 100 entries in that

collection. Implementations may support more than the defined SPM. SCORM is silent on handling of collections that contain more than the SPM number of entries (e.g., an implementation is free to not accept new entries). Implementations should be aware of the consequences if the number of entries exceeds the defined SPM.

The SPMs defined represent implementation storage requirements. As mentioned above, an implementation has to support the storage of at least the SPM. An implementation may elect to support the storage of more than the SPM. If an implementation only supports the SPM and performs a truncation of the characterstring, then ADL recommends that the implementation make information available through the `GetDiagnostic()` method call indicating that truncation of a characterstring has occurred.

For example, if an implementation truncates a characterstring (because it only supports the SPM), then if the SCO calls `GetDiagnostic()` the implementation could return a characterstring that states: "The SetValue() call was successful. The value was greater than the SPM and was truncated".

All validation of characterstrings and collections should be done prior to the actual storage of the value. If an implementation elects to truncate the characterstring at the SPM, the validation process will enforce the validation of the characterstring prior to this truncation. The SPM for the characterstrings are in place for content developers to utilize. Content developers need to be aware of the SPM and what may happen if an SPM is exceeded. It is recommended that content developers keep within the SPM constraints for maximum interoperability.

4.1.1.5 Keyword Data Model Elements

SCORM defines a set of keyword data model elements in the binding of the IEEE Content Object Communication Data Model. The three keyword data model elements are: `_version`, `_count` and `_children`. The keyword data model elements are managed by the LMS and are derived from the state of the other data model elements. These keywords allow SCOs to find out descriptive information about the data model elements. The keyword data model elements can only be applied to certain other data model elements (refer to Section 4.2 for more information). The keyword data model elements shall be implemented as read-only. If a SCO attempts to set (i.e., `SetValue()` API request) a keyword data model element (as defined in Section 4.2), then the LMS shall set the error code to `404 -` Data Model Element Is Read Only and return "`false`". Keywords cannot be applied to other keywords. For example, `cmi.interactions._children._version` is not permitted. If this situation is attempted by a SCO, then the LMS shall return an empty characterstring and set the error code to `401` – Undefined Data Model Element.

`_version`: The `_version` keyword data model element is used to determine the version of the data model supported by the LMS. This keyword data model element cannot be applied to any data model element (refer to Section 4.2.1: *Data Model Version* for implementation details).

_count: The _count keyword data model element is used to determine the number of data model elements currently in a collection. The count is the total number of data model elements in the collection. This value can be requested by the SCO to determine the next index position that is free to be used for storing information. This keyword data model element can only be applied to a data model element that is a collection.

_children: The _children keyword data model element is used to determine all of the data model elements in a parent data model element that are supported by the LMS. The characterstring returned from a valid _children request should be a comma separated listing of all of the data model elements supported by the LMS. The listing of data model elements returned shall match the data model elements found in Section 4.2 (i.e., case-sensitive naming of data model elements apply). The data model element names shall be considered reserved tokens and if white space is provided in the characterstring (i.e., used in separating the reserved tokens), then the white space shall be ignored. The order of the data model elements returned in the characterstring shall not matter. This keyword data model element can only be applied to a data model element that has children.

4.1.1.6 Reserved Delimiters

Due to the nature of the dot-notation binding, some of the data model elements need to convey more information than what can easily be represented as a characterstring. In these cases, a special reserved delimiter has been created to meet the requirements of the dot-notation binding. These special delimiters appear in either the characterstring used in a SetValue() request or in the characterstring returned from a GetValue() request. The cases where a special reserved delimiter may be needed are:

- Representing the language type for a particular characterstring (Data Type: localized_string_type)
- Representing the indication of whether or not order matters in a learner response to an interaction
- Representing the indication of whether or not case matters in a learner response to an interaction
- Representing a set of values in a list or a pair of values

For each of the cases described above, a default value is provided, if applicable. This default value shall be used if the special reserved delimiter is not specified. In all cases, the reserved delimiters shall not be counted toward the value of the SPM. Table 4.1.1.6a describes the specifics of the reserved delimiters.

Table 4.1.1.6a: Reserved Property Delimiters

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

{order_matters=<boolean>}	{order_matters=true}	{order_matters=true} {order_matters=false}

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

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

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

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

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

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

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

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

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

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

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

Valid Examples:

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

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

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

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

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

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

Invalid Examples:

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

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

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

  In this example, the lang delimiter is invalid because its value includes white space, which is not part of a valid language string. Because the delimiter is improperly formed, an LMS should not set the data model element associated

with the call, set the error code to `406 – Data Type Mismatch`, and return `false`.

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

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

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

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

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

Table 4.1.1.6b: Reserved Separator 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}`
Reserved Delimiter Syntax	Default Value	Example
`[.]`	Not applicable, needs to be provided	Used to separate a pair of values that are related for an interaction: `1[.]a`
`[,]`	Not applicable, needs to be provided	Used to separate a set of values for an interaction's collection: `1[.]a[,]2[.]c[,]3[.]b`
`[:]`	Not applicable, needs to be provided	Used to represent a separator between a range of numeric

		values: 1[:]100 – a range where the numeric value is between 1 and 100 (inclusive)

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

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

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

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

- .
- ,
- :

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

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

4.1.1.7 Data Types

Each of the data model elements has a defined data type. All applications of these data types are required to adhere to the defined requirements. This section outlines the set of data types and the specific requirements for each of the data types.

characterstring: A characterstring is a string of characters that are defined in ISO 10646. ISO 10646 is equivalent to the Unicode Standard.

Each data model element defined as type characterstring specifies an SPM number of characters that shall be supported by LMSs. It should be noted that this is the number of *characters* not the octet string which is created by a particular character set encoding scheme (e.g., UTF-8, UTF-16).

localized_string_type: A localized characterstring is a characterstring that has an indicator of the language of the characterstring. There are certain data model elements where the language information is important. SCORM applies a reserved delimiter for representing the language of the characterstring: `{lang=<language_type>}`.

The format of the characterstring is required to have the following syntax:

`"{lang=<language_type>}<actual characterstring>"`

Example:

"{lang=en}The content presented an excellent point dealing with the topic."

The *{lang=<language_type>}* shall represent the delimiter that indicates the language of the characterstring to follow. The *{lang=<language_type>}* is optional. The default language_type, if not specified, shall be "en" (English). The *<language_type>* value shall be implemented with an SPM of 250 characters.

language_type: A data type used to represent a language. The format of this data type is a characterstring consisting of a required language code (`langcode`) followed by multiple, optional, hyphen-prefixed subcodes (`subcode`):

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

where the `langcode`:

- 2-letter codes are defined by ISO 639-1 [14]
- 3-letter codes are defined by ISO 639-2 [15]
- The value "i" is reserved and used as a prefix for registrations defined by Internet Assigned Numbers Authority (IANA)
- The value "x" is reserved and used as a prefix for private use

and the `subcode`:

- 2-letter subcodes are ISO 3166-1 alpha-2 country codes[16]
- subcodes of from 3 to 8 letters are registered with IANA

The requirements for additional `subcode` values are not defined. Both the `langcode` and `subcode` values shall be 1 to 8 characters in length.

ISO 639-2 specifies two code sets (bibliographic and terminology) for the language code. Either code set may be used. The `langcode` and `subcode` is case insensitive. SCORM recommends that the `language_type` be represented with the following format:

- `langcode` is normally given in lower case, and
- `subcodes` (if any) are normally in upper case.

The `language_type` is also permitted to be an empty characterstring for certain data model elements. The `language_type` of a `localized_string_type` is not permitted to be an empty characterstring. The default value for the `language_type` portion of a `localized_string_type` is "en" (English). The `language_type` may be an empty

characterstring for all other data model elements defined as this type (refer to Section 4.2.13: *Learner Preference*).

long_identifier_type: The long_identifier_type represents a label or identifier. This label or identifier shall be unique within the context of the SCO. The long_identifier_type shall be a characterstring that conforms to the syntax defined for Universal Resource Identifiers (URI), refer to RFC 3986 [6]. SCORM recommends that the URI be a globally unique identifier in the form of a Uniform Resource Name (URN), refer to RFC 2141 [3].

All URNs are required to have the following syntax (phrases in quotes are required):

<URN> ::= “urn:”<NID>“:”<NSS>

where <NID> is the Namespace Identifier and <NSS> is the Namespace Specific String [3].

Example: urn:ADL:interaction-id-0001

Since an empty characterstring does not provide sufficient semantic information to uniquely identify a value, then a long_identifier_type value shall not be an empty characterstring and cannot contain all white space characters. The long_identifier_type value shall be implemented with an SPM of 4000 characters.

short_identifier_type: The short_identifier_type represents a label or identifier. This label or identifier shall be unique within the context of the SCO. The short_identifier_type shall be a characterstring that conforms to the syntax defined for URI, refer to RFC 3986 [6].

Since an empty characterstring does not provide sufficient semantic information to uniquely identify a value, then a short_identifier_type value shall not be an empty characterstring and cannot contain all white space characters. The short_identifier_type value shall be implemented with an SPM of 250 characters.

integer: The integer data type indicates that the data model element is a member of the set of positive whole numbers (i.e., 1,2,3, etc.), negative whole numbers (i.e., -1, -2, -3, etc.) and zero (0).

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

```
state (browse,normal,review)
```

Each state is bound to a reserved token. These reserved tokens are specific characterstring representations of the state values defined by the data model elements. These reserved tokens are defined in the Value Space section in the Data Model Element Implementation Requirements. Each defined state does not necessarily map one-to-one to a reserved token (e.g., `not_attempted` state value maps to the `not attempted` reserved token).

real (10,7): The real(10,7) data type denotes a real number with a precision of seven significant digits.

time (second, 10, 0): The time (second,10,0) data type represents a point in time. This data type shall have a required precision of 1 second and an optional precision of 0.01 seconds [1]. Implementations should be aware of this required precision versus the optional precision, for it may impact implementation decisions. For example, if an application is expecting the optional precision, it may not be supported. The SCORM dot-notation binding defines a particular format for a characterstring to represent a point in time. The format of the characterstring shall be as follows:

YYYY[-MM[-DD[Thh[:mm[:ss[.s[TZD]]]]]]] where

- YYYY: A four-digit year (1970 <= YYYY <=2038)
- MM: A two-digit month (01 through 12 where 01=January)
- DD: A two-digit day of month (01 through 31, depending on the value of month and year)
- hh: Two-digits of hour (00 through 23)
- mm: Two-digits of minute (00 through 59)
- ss: Two-digits of second (00 through 59)
- s: One or more digits representing a decimal fraction of a second. If fractions of a second are used, SCORM further restricts the string to a maximum of 2 digits (e.g., 34.45 – valid, 34.45454545 – not valid).
- TZD: Time zone designator (`Z` for UTC or `+hh:mm` or `-hh:mm`).
    - The `hh` and `mm` shall adhere to the requirements defined above for `hh` and `mm`.
    - The time zone designator uses the extended format as defined by ISO 8601-2000. The extended format requires the use of the colon (:) as a separator between the hours and minutes. If the difference between local time and UTC is required then the time can be expressed in hours and minutes (i.e., 03:10), or if the time difference is exactly an integral number of hours, then hours only (03).
    - The leading zero is required for hours less than 10.
    - If no TZD designator is provided, then the default time zone shall be interpreted as “local time”.
    - If TZD is defined, then hh:mm:ss.s shall also be defined, although they may be all zeros.
- At least the four-digit year must be present. If additional parts of the time are included, the character literals `-`, `T`, `:` and `.` are part of the character lexical representation [1].

Example:

- 2005
- 2005-07-25T03:00:00
- 2005-07-25T03:00:00-03:10
- 2005-07-25T03:30:35.5+05
- 2005-07-25T03:00:00Z

timeinterval (second, 10,2): The timeinterval (second, 10, 2) denotes that the value for the data model element timeinterval represents elapsed time with a precision of 0.01

seconds[1]. The SCORM dot-notation binding defines a particular format for a characterstring to represent a timeinterval.

The format of the characterstring shall be as follows:

P[yY][mM][dD][T[hH][nM][s[.s]S]] where:

- y: The number of years (integer, >= 0, not restricted)
- m: The number of months (integer, >=0, not restricted)
- d: The number of days (integer, >=0, not restricted)
- h: The number of hours (integer, >=0, not restricted)
- n: The number of minutes (integer, >=0, not restricted)
- s: The number of seconds or fraction of seconds (real or integer, >=0, not restricted). If fractions of a second are used, SCORM further restricts the string to a maximum of 2 digits (e.g., 34.45 – valid, 34.45454545 – not valid).
- The character literals designators `P`, `Y`, `M`, `D`, `T`, `H`, `M` and `S` shall appear if the corresponding non-zero value is present.
- Zero-padding of the values shall be supported. Zero-padding does not change the integer value of the number being represented by a set of characters. For example, PT05H is equivalent to PT5H and PT000005H.

Example:

- P1Y3M2DT3H indicates a period of time of 1 year, 3 months, 2 days and 3 hours
- PT3H5M indicates a period of time of 3 hours and 5 minutes

Implementers should be aware that the format and binding is for the communication of the data between a SCO and an LMS. Since the format is representing a period of time, then a duration like PT5M is equivalent to PT300S.

If the data model element that is of type timeinterval(second,10,2) contains a value, then

- The designator `P` shall be present;
- If the value of years, months, days, hours, minutes or seconds is zero, the value and corresponding character literal designation may be omitted, but at least one character literal designator and value shall be present in addition to the designator `P`;
- The designator `T` shall be omitted if all of the time components (hours, minutes and seconds) are not used. A zero value may be used with any of the time components (e.g., PT0S).

4.1.1.8 Extending the SCORM Run-Time Environment Data Model

The SCORM Run-Time Environment Data Model itself shall not be extended. If an LMS receives an API request in which the parameter is cmi.*elementName* (where *elementName* is some other token not defined in Section 4.2), then an LMS shall behave as follows:

- `GetValue(parameter)`: The LMS shall return an empty characterstring and set the error code to `401` – Undefined Data Model Element
- `SetValue(parameter_1, parameter_2)`: The LMS shall return "`false`" and set the error code to `401` – Undefined Data Model Element

There is work currently taking place to address additional ways of extending and defining other data models. SCORM may be updated to include these development efforts as they progress and become stable.

4.2. SCORM Run-Time Environment Data Model

The SCORM Run-Time Environment Data Model contains a set of data model elements that can be tracked by the SCO with an LMS during the run-time of the SCO. The data model elements can be used to track items like status, scores, interactions, objectives, etc. Some of the run-time environment data model elements impact each other or are used in coordination with others. Some of the data model elements, if used, impact the control and sequence of other SCOs that are being used in the same context (e.g., lesson or course). Table 4.2.a summarizes high level information about each data model elements.

Table 4.2a: SCORM Run-Time Environment Data Model Elements Summary

Data Model Element	Dot-Notation Binding	Description
Comments From Learner	cmi.comments_from_learner	Contains text from the learner.
Comments From LMS	cmi.comments_from_lms	Contains comments and annotations intended to be made available to the learner.
Completion Status	cmi.completion_status	Indicates whether the learner has completed the SCO.
Completion Threshold	cmi.completion_threshold	Identifies a value against which the measure of the progress the learner has made toward completing the SCO can be compared to determine whether the SCO should be considered completed.
Credit	cmi.credit	Indicates whether the learner will be credited for performance in this SCO.
Entry	cmi.entry	Contains information that asserts whether the learner has previously accessed the SCO.
Exit	cmi.exit	Indicates how or why the learner left the SCO.
Interactions	cmi.interactions	Defines information pertaining to an interaction for the purpose of measurement or assessment.
Launch Data	cmi.launch_data	Provides data specific to a SCO that the SCO can use for initialization.
Learner Id	cmi.learner_id	Identifies the learner on behalf of whom the SCO instance was launched.
Learner Name	cmi.learner_name	Represents the name of the learner.
Learner Preference	cmi.learner_preference	Specifies learner preferences associated with the learner's use of the SCO.
Location	cmi.location	Represents a location in the SCO.
Maximum Time Allowed	cmi.max_time_allowed	Indicates the amount of accumulated time the learner is allowed to use a SCO in the learner attempt.
Mode	cmi.mode	Identifies the modes in which the SCO may be presented to the learner.
Objectives	cmi.objectives	Specifies learning or performance objectives associated with a SCO.

Progress Measure	cmi.progress_measure	Identifies a measure of the progress the learner has made toward completing the SCO.
Scaled Passing Score	cmi.scaled_passing_score	Identifies the scaled passing score for a SCO.
Score	cmi.score	Identifies the learner's score for the SCO.
Session Time	cmi.session_time	Identifies the amount of time that the learner has spent in the current learner session for the SCO.
Success Status	cmi.success_status	Indicates whether the learner has mastered the SCO.
Suspend Data	cmi.suspend_data	Provides information that may be created by a SCO as a result of a learner accessing or interacting with the SCO.
Time Limit Action	cmi.time_limit_action	Indicates what the SCO should do when the maximum time allowed is exceeded.
Total Time	cmi.total_time	Identifies the sum of all of the learner's learner session times accumulated in the current learner attempt prior to the current learner session.

The following sections define the requirements for implementation of the SCORM Run-Time Environment Data Model. Each data model element is presented in a new section (i.e., 4.2.1, 4.2.2, etc.). Each section contains a table that describes the specific requirements for the data model element. These requirements apply to both LMS and SCO implementations. Some requirements impact LMS implementations; some impact SCO implementations and some impact both LMS and SCO implementations.

Table 4.2b describes the layout and format of the tables and provides descriptive information describing each portion of the tables.

Table 4.2b: Data Model Element Table Explanation

Dot-Notation Binding	Details
<dot-notation characterstring representation of the data model element>	This section provides descriptive data about the data model element. **Data Model Element Implementation Requirements:** This section of the table defines the data model element implementation requirements. This section outlines those requirements about the data type that both an LMS and a SCO shall adhere. This section of the table is broken up into three sub-sections Data Type, Value Space and Format. • **Data Type:** Describes the specific data type for the data model element as defined by the IEEE standard [1]. • **Value Space:** Represents the space of values that can be held by the data type. For example, the data type may be a characterstring, however that characterstring data values may be restricted to the ASCII character set. • **Format:** Describes any format restrictions placed on the value for the data type. For example, times may have a certain format (i.e., hh:mm:ss). **LMS Behavior Requirements:** • This section describes the set of requirements that an LMS is required to follow. **Sequencing Impacts:** • This section describes the impacts of the data model

	element on sequencing rules and behaviors. If this section is omitted, the data model element has no impacts on sequencing. **SCO Behavior Requirements:** • This section describes the set of requirements that a SCO is required to follow. **API Implementation Requirements:** • **GetValue():** This section outlines the specific behaviors that an LMS shall adhere to when processing GetValue() requests for the specified data model element. This section also outlines the error conditions that could occur using the specified data model element with a GetValue() request. • **SetValue():** This section outlines the specific behaviors that an LMS shall adhere to when processing SetValue() requests for the specified data model element. This section also outlines the error conditions that could occur using the specified data model element with a SetValue() request. **Additional Behavior Requirements:** • This section outlines any additional behavior requirements that are specific to the data model element. **Example:** • This section outlines valid API method calls using the data model element.

4.2.1. Data Model Version

The `cmi._version` keyword data model element can be used by the SCO to determine the data model version supported by the LMS. The associated `cmi._version` for the SCORM Run-Time Environment Version 1.3 shall be `1.0`. This value is defined by the IEEE standard [1]. This value can be requested by a SCO to determine the version of the SCORM Run-Time Environment Data Model supported by the LMS. For example, a SCO could be authored to support multiple versions of the SCORM Run-Time Environment Data Model if so desired.

Table 4.2.1a: Dot-notation Binding for the Data Model Version Data Model Element

Dot-Notation Binding	Details
cmi._version	Represents the version of the data model. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** The value shall consist of a period-delimited characterstring containing major and minor release values as whole numbers. Any characters appearing after the minor release value shall be separated from the minor release value by a period (".") [1]. The characterstring shall represent the value defined by the IEEE standard: `1.0` **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented as read-only. • The LMS is responsible for returning the characterstring `1.0`, when requested by the SCO. **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the data model version (*cmi._version*) data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value `1.0` (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi._version*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi._version")

4.2.2. Comments From Learner

There may be times where the content designer wishes to collect comments from the learner about the learning experience. The data model permits the tracking of comments from the learner on a per SCO basis. How the comments are collected or presented is outside the scope of SCORM. For example, an LMS may provide an option to collect comments on the SCO through some LMS provided user interface control or the SCO may have the ability to collect comments built directly into the SCO. How the comments are used is also outside the scope of SCORM. For example, once the comments have been collected, an LMS may provide the designers (of the SCO) the ability to create a report listing out the comments. These may be able to be collected for the entire content organization. These comments then may be used by the designer to evaluate the current design and structure of the content.

The `cmi.comments_from_learner` data model element provides the ability to not only collect the actual text of the comment but also the location (where in the content) and a timestamp (when).

The `cmi.comments_from_learner` contains freeform text generated by the learner. The structure of the text is not specified. The value of this data model element is intended to provide feedback about the SCO or the learning experience with the SCO from a specific learner. Using this data model element for other purposes may adversely affect interoperability.

The LMS shall support at least the SPM of 250 comments from the learner. Each comment from the learner contains a data model element that represents the textual comment made by the learner. This textual comment has a SPM of 4000. In either case, the LMS is free to support more than the SPM.

Table 4.2.2a: Dot-notation Binding for the Comments from Learner Data Model Element

Dot-Notation Binding	Details
cmi.comments_from_learner._children	The *cmi.comments_from_learner._children* data model element represents a listing of supported data model elements. This data model element is typically used by a SCO to determine which data model elements are supported by the LMS. The characterstring returned may be used by the SCO to dynamically build parameters for the GetValue() and SetValue() requests. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the data model elements in the Comments From Learner parent data model element that are supported by the LMS. Since all data model elements are required to be supported by the LMS, the characterstring shall represent the following data model elements: o comment o location o timestamp

	LMS Behavior Requirements: • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for returning a comma-separated list of all of the Comments From Learner child data model elements (refer to Data Model Element Implementation Requirements above). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.comments_from_learner._children* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of the Comments From Learner child data model elements supported by the LMS (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. The ordering of data model elements is not important. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_learner._children*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.comments_from_learner._children")
cmi.comments_from_learner._count	The *cmi.comments_from_learner._count* keyword describes the current number of learner comments that are being stored by the LMS for the SCO. The total number of entries currently being managed by the LMS shall be returned. **Data Model Element Implementation Requirements:** • **Data Type:** integer • **Value Space:** Non-negative integer • **Format:** The characterstring representing the number of learner comments that the LMS is currently managing. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • If the LMS receives a request to get the *cmi.comments_from_learner._count* value prior to any comments being set by the SCO, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.comments_from_learner._count* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the number of learner comments currently stored by the LMS and set the error code to `0` – No Error. o If no comments have been set by the SCO, then the LMS shall return `0` , set the error code to `0` - No Error. • **SetValue():** If the SCO invokes a SetValue() request to

	set the *cmi.comments_from_learner._count*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.comments_from_learner._count")
cmi.comments_from_learner.n.comment	The *cmi.comments_from_learner.n.comment* data model element shall describe textual input [1]. The characterstring value represents a localized characterstring. **Data Model Element Implementation Requirements:** • **Data Type:** localized_string_type (SPM: 4000) • **Value Space:** A characterstring (defined by ISO-10646-1) with localization information • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The value is supplied by the SCO. The LMS shall not make any assumption on an initial value for this data model element. If a GetValue() request is made before the actual comment has been set by the SCO, then the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.comments_from_learner.n.comment* data model element. • During a SetValue() request, the SCO should be aware that the delimiter is optional. If the delimiter is not provided as part of the characterstring, the LMS will assume that the default language is "en" (English). • During a GetValue() request, the SCO should be aware that the delimiter may be part of the characterstring returned by the LMS (depending on the LMS implementation). What the SCO does with the characterstring returned by the LMS is dependent on the implementation of the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_learner.n.comment* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 comments in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.

	○ If the SCO attempts to retrieve the *cmi.comments_from_learner.n.comment* and the record of data has been created but the comment data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** The LMS shall set the *cmi.comments_from_learner.n.comment* to the supplied value in the SetValue() request, set the error code `0` - No Error and return “`true`”. ○ If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return “`false`”. The LMS shall not alter the state of the data model element based on the request. ○ The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of comments from learner being stored (can be determined by requesting the *cmi.comments_from_learner._count*), then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for more information on processing this request. **Additional Behavior Requirements:** • If the data model element is implemented, the SCO has the responsibility to make sure that *cmi.comments_from_learner.n.comment* is set initially in a sequential order. The SCO has the ability to retrieve previously set comments and overwrite these comments, updating the comment, location and timestamp. The SCO should be aware of the smallest permitted maximum number of characters (4000) that shall be implemented by the LMS. **Example:** • GetValue(“cmi.comments_from_learner.0.comment”) • SetValue(“cmi.comments_from_learner.0.comment”,”So me comments about the SCO”)
cmi.comments_from_learner.n.location	The *cmi.comments_from_learner.n.location* data model element indicates the point in the SCO to which the comment applies. This data model element is implementation-defined by each SCO. If no value is specified for location, then the comment is applicable to the entire SCO (as a whole) rather than a specific location in the SCO. [1]. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 250) • **Value Space:** ISO-10646-1 • **Format:** The format of this data model element is defined and controlled by the SCO. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the characterstring data

type.

LMS Behavior Requirements:

- The data model element is mandatory and shall be implemented by the LMS as read/write.
- The value is controlled and supplied by the SCO. If a GetValue() request is made before the actual location has been set by the SCO, then the LMS shall behave according to the API Implementation Requirements below.

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.comments_from_learner.n.location* data model element.

API Implementation Requirements:

- **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_learner.n.location* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 comments in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (“”). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
  - If the SCO attempts to retrieve the *cmi.comments_from_learner.n.location* and the record of data has been created but the location data model element has not been set by the SCO, then the LMS shall set the error code to `403` – Data Model Element Value Not Initialized and return an empty characterstring (“”).
- **SetValue():** The LMS shall set the *cmi.comments_from_learner.n.location* to the supplied value in the SetValue() request, set the error code `0` - No Error and return “`true`”.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of comments from learner being stored (can be determined by requesting the *cmi.comments_from_learner._count*), then the LMS shall set the error code to `351` – General set failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for more information on processing this request.

Additional Behavior Requirements:

- The SCO has the ability to retrieve previously set

	comments and overwrite these comments, updating the comment, location and timestamp. The SCO should be aware of the smallest permitted maximum number of characters (250) that shall be implemented by the LMS. **Example:** • GetValue("cmi.comments_from_learner.0.location") • SetValue("cmi.comments_from_learner.0.location","PAGE1SECTION#3")
cmi.comments_from_learner.n.timestamp	The *cmi.comments_from_learner.n.timestamp* data model element indicates the point in time at which the comment was created or most recently changed. Implementation shall support, minimally, time periods in the range of January 1, 1970 through January 1, 2038 [1]. **Data Model Element Implementation Requirements:** • **Data Type:** time (second,10,0) • **Value Space:** The data type denotes the value for time is expressed as a value that is accurate to one second and optionally accurate to one hundredth of a second (0.01). • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the time (second,10,0) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read/write. • The value is controlled and supplied by the SCO. If a GetValue() request is made before the timestamp has been set, then the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.comments_from_learner.n.timestamp* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_learner.n.timestamp* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 comments in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.comments_from_learner.n.timestamp* and the record of data has been created but the *cmi.comments_from_learner.n.timestamp* data model element has not been set by the SCO, then the LMS shall set the error code to `403`

	Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** The LMS shall set the *cmi.comments_from_learner.n.timestamp* to the supplied value in the SetValue() request, set the error code `0` - No Error and return “`true`”. o If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return “`false`”. The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of comments from learner being stored (can be determined by requesting the *cmi.comments_from_learner._count*), then the LMS shall set the error code to `351` – General set failure and return “`false`”. Refer to Section 3.17.6: *SCORM Extension Error Conditions* for more information on processing this request. **Example:** • GetValue(“cmi.comments_from_learner.0.timestamp”) • SetValue(“cmi.comments_from_learner.0.timestamp”,“2003-07-25T03:00:00”)

4.2.3. Comments From LMS

The `cmi.comments_from_lms` data model element contains comments and annotations intended to be seen by all learners for the SCO for which they are defined. These comments are intended to be a mechanism for adding information of interest to all learners in a particular community, instructor notes, etc. SCORM does not define a mechanism for how these comments are initialized. LMSs are free to provide a mechanism to support the creation and initialization of this data. This support is not required for SCORM conformance.

How this information is presented or used is outside the scope of SCORM. One such use would be for the ability to retrieve the comments and display them to the learner upon launch of the SCO (or during some point in the learner session).

The value of `cmi.comments_from_lms` is intended to provide information about the SCO or the learning experience with the SCO. The structure of this data is not specified by SCORM.

The LMS shall support at least the SPM of 100 comments from the LMS. The LMS is free to support more than the SPM.

Table 4.2.3a: Dot-notation Binding for the Comment from LMS Data Model Element

Dot-Notation Binding	Details
cmi.comments_from_lms._children	The *cmi.comments_from_lms._children* data model element represents a listing of supported data model elements. This data model element is typically used by a SCO to determine which data model elements are supported by the LMS. The characterstring returned may be used by the SCO to dynamically build parameters for the GetValue() and SetValue() requests. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the data model elements in the Comments From LMS parent data model element that are supported by the LMS. Since all data model elements are required to be supported by the LMS, the characterstring shall represent the following data model elements: o comment o location o timestamp **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for returning a comma-separated list of all of the data model elements (refer to Data Model Element Implementation Requirements above). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is

	only permitted to retrieve the value of the *cmi.comments_from_lms._children* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. The ordering of elements is not important. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_lms._children*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.comments_from_lms._children")
cmi.comments_from_lms._count	The *cmi.comments_from_lms._count* keyword is used to describe the current number of comments from the LMS that are currently being stored by the LMS. The total number of entries currently being stored by the LMS shall be returned. **Data Model Element Implementation Requirements:** • **Data Type:** integer • **Value Space:** Non-negative integer • **Format:** The characterstring representing the number of comments that the LMS is currently persisting. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • If the LMS receives a request to get the *cmi.comments_from_lms._count* value prior to any comments being set by the SCO, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.coments_from_lms._count* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the number of comments currently stored by the LMS and set the error code to `0` – No Error. o If there are no comments defined for this element, then the LMS shall return `0` and set the error code to `0`. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_lms._count*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.comments_from_lms._count")
cmi.comments_from_lms.n.comment	The *cmi.comments_from_lms.n.comment* data model element shall describe comments or annotations associated with a SCO [1]. The characterstring value represents the localized characterstring.

Data Model Element Implementation Requirements:

- **Data Type:** localized_string_type (SPM: 4000)
- **Value Space:** A characterstring (defined by ISO-10646-1) with localization information
- **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type.

LMS Behavior Requirements:

- The data model element is mandatory and shall be implemented by an LMS as read-only.
- How this data model element is initialized is outside the scope of SCORM. The LMS may provide a means to allow the author/instructor to provide the comments and these comments may then be used to initialize this value. If a GetValue() request is made prior to the comment being set or initialized by the LMS, then the LMS shall behave according to the API Implementation Requirements below.

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.comments_from_lms.n.comment* data model element.
- During a GetValue() request, the SCO should be aware that the delimiter may be part of the characterstring returned by the LMS (depending on the LMS implementation). What the SCO does with the characterstring returned by the LMS is dependent on the implementation of the SCO. If no delimiter is provided the SCO shall assume the default language of "en".

API Implementation Requirements:

- **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_lms.n.comment* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 comments in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
  - If the SCO attempts to retrieve the *cmi.comments_from_lms.n.comment* and the record of data has been created but the comment data model element has not been set by the LMS, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring ("").

	• **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_lms.n.comment*, then the LMS shall set the error code to 404 – Data Model Element Is Read Only and return "`false`". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.comments_from_lms.0.comment")
cmi.comments_from_lms.n.location	The *cmi.comments_from_lms.n.location* data model element indicates the point in the SCO to which the comment applies. This data model element is implementation-defined by each SCO. If no value is specified for location, then the comment is applicable to the entire SCO (as a whole) [1]. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 250) • **Value Space:** ISO-10646-1 • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the characterstring data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read-only. • How this element is initialized is outside the scope of SCORM. The LMS may provide a means to allow the author/instructor to provide the comments (along with a location and date) and these comments may then be used to initialize this value. If a GetValue() request is made prior to the location being set or initialized by the LMS, then the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the *cmi.comments_from_lms.n.location* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_lms.n.location* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 comments in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.comments_from_lms.n.location* and the

	record of data has been created but the location data model element has not been set by the LMS, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_lms.n.location*, then the LMS shall set the error code to 404 – Data Model Element Is Read Only and return “`false`”. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.comments_from_lms.0.location”)
cmi.comments_from_lms.n.timestamp	The *cmi.comments_from_lms.n.timestamp* data model element indicates the point in time at which the comment was created or most recently changed. Implementation shall support, minimally, time periods in the range of January 1, 1970 through January 1, 2038 [1]. **Data Model Element Implementation Requirements:** • **Data Type:** time(second,10,0) • **Value Space:** The data type denotes that the value for time is a number expressed as a real data type with values that are accurate to one second and optionally accurate to one hundredth of a second (0.01). The number of seconds in the time value is the number of seconds since 00:00 on January 1, 1970 [1]. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the time (second,10,0) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read-only. • How this element is initialized is outside the scope of SCORM. The LMS may provide a means to allow the author/instructor to provide the comments (along with a location and date) and these comments may then be used to initialize this value. If a GetValue() request is made prior to the timestamp being set or initialized by the LMS, then the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.comments_from_lms.n.timestamp* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_lms.n.timestamp* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the

	LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 comments in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (“”). Refer to Section 3.1.7.6: *SCORM Extension Error Condition* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.comments_from_lms.n.timestamp* and the record of data has been created but the timestamp data model element has not been set by the LMS, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_lms.n.timestamp*, then the LMS shall set the error code to 404 – Data Model Element Is Read Only and return “`false`”. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.comments_from_lms.0.timestamp”)

4.2.4. Completion Status

The `cmi.completion_status` data model element indicates whether the learner has completed the SCO [1]. How the SCO determines its completion status is outside the scope of SCORM. This value indicates the overall completion status for the SCO as determined by the SCO developer.

Table 4.2.4a: Dot-notation Binding for the Completion Status Data Model Element

Dot-Notation Binding	Details
cmi.completion_status	**Data Model Element Implementation Requirements:** • **Data Type:** state (completed, incomplete, not_attempted, unknown) • **Value Space:** The IEEE defines four state values. SCORM binds these state values to the following restricted vocabulary tokens: o "completed": The learner has experienced enough of the SCO to consider the SCO complete [1]. How completion is determined is controlled and managed by the SCO. The completion status can be overridden by an LMS based on the requirements defined in 4.2.4.1: *Completion Status Evaluation*. o "incomplete": The learner has not experienced enough of the SCO to consider the SCO complete [1]. How completion is determined is controlled and managed by the SCO. o "not attempted": The learner is considered not to have used the SCO in any significant way [1]. o "unknown": No assertion is made [1]. This indicates that no applicable assertion can be made that indicates the completion status. • **Format:** The format of the data model value shall be one of the four restricted vocabulary tokens listed above ("completed", "incomplete", "not attempted", "unknown"). **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read/write. • If a *cmi.completion_threshold* is not defined, the determination of *cmi.completion_status* is controlled and managed by the SCO, the LMS cannot imply any value for the *cmi.completion_status* in any way. There is no requirement in SCORM that mandates a SCO to set *cmi.completion_status*. If the SCO does not set the *cmi.completion_status*, the LMS shall use the default value of "unknown" as the value for *cmi.completion_status*. • If a *cmi.completion_threshold* is defined, it is the responsibility of the LMS to maintain congruence between the *cmi.completion_threshold*, *cmi.progress_measure*, and the value used by the LMS for *cmi.completion_status*. The LMS must report (when requested via a GetValue() call) *cmi.completion_status* by adhering to the requirements defined in section 4.2.4.1: *Completion Status Evaluation.* **Sequencing Impacts:** • If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "unknown", the Attempt Progress Status for the learning activity associated with the SCO shall be false. • If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "completed", the Attempt Progress Status for the learning activity associated with the SCO shall be true, and the Attempt Completion Status for the learning activity associated with the SCO shall be true. • If the SCO or LMS (through the process described in Section 4.2.4.1) sets

	cmi.completion_status, of the SCO to "incomplete", the Attempt Progress Status for the learning activity associated with the SCO shall be true, and the Attempt Completion Status for the learning activity associated with the SCO shall be false. • If the SCO or LMS (through the process described in Section 4.2.4.1) sets *cmi.completion_status*, of the SCO to "not attempted", the Attempt Progress Status for the learning activity associated with the SCO shall be true and the Attempt Completion Status for the learning activity associated with the SCO shall be false. **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.completion_status* data model element. • The SCO should be aware that setting the *cmi.completion_status* will affect the learning activity associated with the SCO, therefore possibly affecting sequencing. • If there is sequencing information applied to the learning activity associated with the SCO that relies on completion status, the SCO must ensure completion information is accurately sent to the LMS (SetValue()) prior to the SCO's learner session ending. Otherwise, the LMS will use the value "unknown" as the completion status of the learning activity associated with the SCO when processing sequencing information. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.completion_status* currently managed by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If a *cmi.completion_threshold* is defined, then the LMS shall report (when requested via GetValue() call) *cmi.completion_status* by adhering to the requirements defined in section 4.2.4.1: *Completion Status Evaluation.* o If a *cmi.completion_threshold* is not defined, until the SCO reports a value, the default value of the *cmi.completion_status* is "unknown". • **SetValue():** If the SCO invokes a request to set the *cmi.completion_status* and the value is not a member of the restricted vocabulary tokens described above, then the LMS shall set the API Error Code to `406` – Data Model Element Type Mismatch and return "false". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.completion_status") • SetValue("cmi.completion_status","incomplete")

4.2.4.1 Completion Status Evaluation

Typically, the completion status of the SCO is determined by the SCO. The completion status of the SCO can be designed by the content developer in various ways, including but not limited to:

- Number of pages visited by the learner
- Result of a learner selecting a user interface control (e.g., learner pressing a button)
- Completion of the learner viewing a piece of video or reading a document

- Completion of various objectives in the SCO (i.e., `cmi.objectives.n.completion_status`)

Regardless of how the determination is made by the SCO, this process involves the SCO setting the `cmi.completion_status`. SCORM does not require the SCO to track (i.e., `GetValue()` or `SetValue()` function calls) any SCORM Run-Time Environment Data Model elements. With this in mind, the LMS has additional behaviors that are required in the determination of completion status. Under certain circumstances, the LMS is required to behave differently.

The `cmi.completion_status` data model element is impacted by two other SCORM Run-Time Environment Data Model Elements (Completion Threshold – `cmi.completion_threshold` and Progress Measure – `cmi.progress_measure`). The following table defines the possible states of these values and the defined LMS behavior. The Completion Threshold value may be defined in the `imsmanifest.xml` file (refer to Section 4.2.5: *Completion Threshold* for more details). Both the Progress Measure and Completion Status are determined and set by the SCO. Table 4.2.4.1a defines the LMS behaviors associated with the combinations of these values being set/defined or not set/defined. The LMS behavior shall be applied when a `GetValue()` request to retrieve the value of the `cmi.completion_status` is made by the SCO. Whether or not the actual value stored by the LMS for `cmi.completion_status` is overridden and persisted during this evaluation process is outside the scope of SCORM and is implementation specific.

Table 4.2.4.1a: GetValue() Evaluation of Completion Status

Completion Threshold (*)	Progress Measure	Completion Status	LMS GetValue() Behavior
None defined	No value set by the SCO	No value set by the SCO	The `cmi.completion_status` shall be evaluated as `unknown` and this value shall be returned to the SCO.
None defined	No value set by the SCO	One of the defined vocabularies	No action, the `cmi.completion_status` currently being maintained by the LMS (i.e., the value set in the last successful call to SetValue() by the SCO) shall be returned to the SCO.
None defined	0.5	One of the defined vocabularies	No action, the `cmi.completion_status` currently being maintained by the LMS (i.e., the value set in the last successful call to SetValue() by the SCO) shall be returned to the SCO.
0.8	0.5	One of the defined vocabularies	The `cmi.completion_status` shall be evaluated to `incomplete` and this value shall be returned to the SCO. RATIONALE: 0.5 < 0.8. Refer to the requirements defined by `cmi.completion_threshold` and `cmi.progress_measure`.
0.8	0.9	One of the defined vocabularies	The `cmi.completion_status` shall be evaluated to `completed` and this value shall be returned to the SCO. RATIONALE: 0.9 >= 0.8. Refer to the requirements

			defined by `cmi.completion_threshold` and `cmi.progress_measure`.
0.8	No value set by the SCO	No value set by the SCO	The `cmi.completion_status` shall be evaluated to `unknown` and this value shall be returned to the SCO.
0.8	0.5	No value set by the SCO	The `cmi.completion_status` shall be evaluated to `incomplete` and this value shall be returned to the SCO. RATIONALE: 0.5 < 0.8. Refer to the requirements defined by `cmi.completion_threshold` and `cmi.progress_measure`.
0.8	0.9	No value set by the SCO	The `cmi.completion_status` shall be evaluated to `completed` and this value shall be returned to the SCO. RATIONALE: 0.9 >= 0.8. Refer to the requirements defined by `cmi.completion_threshold` and `cmi.progress_measure`.
None defined	0.5	No value set by the SCO	The `cmi.completion_status` shall be evaluated to `unknown` and this value shall be returned to the SCO.
0.8	No value set by the SCO	One of the defined vocabularies	The `cmi.completion_status` shall be evaluated to `unknown` and this value shall be returned to the SCO.

*** ADL Note:** The Completion Threshold is defined by the ADL namespaced element `<adlcp:completionThreshold>` associated with the `<adlcp:item>` element that references the SCO Resource. If an `<adlcp:completionThreshold>` is defined, then the LMS is responsible for initializing the `cmi.completion_threshold` data model element to the value used. If no `<adlcp:completionThreshold>` is defined, then the `cmi.completion_threshold` is not defined and shall not be initialized to any value by the LMS.

4.2.5. Completion Threshold

The value stored in the `cmi.completion_threshold` data model element can be used to determine whether the SCO should be considered complete. This can be accomplished by comparing the `cmi.completion_threshold` to the `cmi.progress_measure`, made by the learner, towards the completion of the SCO.

Table 4.2.5a: Dot-notation Binding for the Completion Threshold Data Model Element

Dot-Notation Binding	Details
cmi.completion_threshold	**Data Model Element Implementation Requirements:** • **Data Type:** real(10,7) range (0..1) • **Value Space:** 0.0 <= *cmi.completion_threshold* <= 1.0 • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real(10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read-only. • The LMS is responsible for initializing this data model element using the ADL Content Packaging namespace element <adlcp:completionThreshold>. This element, if needed, shall only be placed on an <imscp:item> element that references a SCO resource, found in a Content Package Manifest. **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.completion_threshold* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.completion_threshold* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If the SCO attempts to retrieve the *cmi.completion_threshold* and no completion threshold was defined in the Content Package Manifest (<adlcp:completionThreshold> element), then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.completion_threshold*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return “`false`”. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.completion_threshold”)

4.2.6. Credit

The `cmi.credit` data model element indicates whether the learner will be credited for performance in the SCO [1]. How a SCO is prescribed to be taken for credit or no credit is outside the scope of SCORM. The default value for this data model element is the fact that the SCO is being taken for credit.

Typically, one does not choose which SCOs should be for credit and no credit. This is usually handled at the content organization (e.g., course) level. If an LMS provides a mechanism for allowing learners to register for content organization for credit and no credit, then the value that the learner chooses (or possibly the value that the instructor requires for the content organization) shall be used throughout all of the SCOs found in the content organization.

Table 4.2.6a: Dot-notation Binding for the Credit Data Model Element

Dot-Notation Binding	Details
cmi.credit	**Data Model Element Implementation Requirements:** • **Data Type:** state (credit, no_credit) • **Value Space:** The IEEE defines two state values. SCORM binds these state values to the following restricted vocabulary tokens: o "credit": The learner is taking the SCO for credit [1]. The default value for this data model element shall be "credit". If no mechanism is supplied to determine or assign this value, then the default of "credit" shall be used. Taking the SCO for credit affects the determination of the *cmi.success_status* data model element (refer to Section 4.2.16.1: *Mode and Credit Usage Requirements* for more details). o "no-credit": The learner is taking the SCO for no credit [1]. If the learner is taking the SCO for no credit, then values that affect the determination of success status or score shall not be interpreted by the LMS and shall not affect the current success status or score for the SCO. • **Format:** The format of the data model value shall be one of the two restricted tokens listed above ("credit", "no-credit"). **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • SCORM does not require that an LMS support a mechanism for prescribing the credit state for a SCO. Whether or not an LMS supports a mechanism for this functionality is outside the scope of SCORM. If the LMS does not support a mechanism, then the default value ("credit") of credit shall be returned in all cases. **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.credit* data model element. • What the SCO does with this data is totally up to the discretion of the SCO. The SCO could use the *cmi.credit* value to determine the importance of reporting other data model element values to the LMS (via a SetValue() call). **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.credit* currently stored by the LMS for the learner and set the error code to `0` – No Error.

	The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If the SCO requests the *cmi.credit* and the LMS does not support a mechanism or process for selecting "credit" vs. "no-credit", then the LMS shall return the default value ("credit"). o If the LMS supports such a mechanism or process, then the LMS shall return the current value being stored for *cmi.credit.* • **SetValue():** If the SCO invokes a request to set the *cmi.credit*, then the LMS shall set the API Error Code to `404` – Data Model Element Is Read Only and return "false". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.credit")

4.2.7. Entry

The `cmi.entry` data model element contains information that asserts whether the learner has previously accessed the SCO [1].

As defined in the Temporal Model (refer to Section 2.1.1: *Run-Time Environment Temporal Model*), an entry value of "ab-initio" indicates that the SCO has a default (clean) set of run-time data. There is no run-time data available from any previous learner attempts. An entry value of "resume" indicates that the SCO is accessing run-time data for the current learner attempt as set from the previous learner session on the SCO.

Table 4.2.7a: Dot-notation Binding for the Entry Data Model Element

Dot-Notation Binding	Details
cmi.entry	**Data Model Element Implementation Requirements:** • **Data Type:** state (ab_initio, resume, _nil_) [1] • **Value Space:** The IEEE defines three state values. SCORM binds these state values to the following restricted vocabulary tokens: o "ab-initio": Indicates that the learner has not accessed the SCO during the current learner attempt [1]. This value indicates that this is the first learner session associated with the current learner attempt on the SCO. o "resume": Indicates that (1) the learner has previously accessed the SCO during the current learner attempt, and (2) upon exiting, the *cmi.exit* data model element had the value of "suspend" or "logout" [1] or the LMS triggered a Suspend All Navigation Request. o "" (empty characterstring): Indicates all other conditions. There is no knowledge of previous access or no specific entry condition is indicated [1]. Another example could be that the SCO was already completed or mastered, and later it was launched for review purposes. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for initializing the *cmi.entry* data model element based on the following rules: o If this is the first learner session on a learner attempt, then the LMS shall set the *cmi.entry* to "ab-initio" upon initial launch of the SCO. o If the learner attempt on the SCO is being resumed from a suspended learner session (*cmi.exit* was set to "suspend" or "logout"), then the LMS shall initialize this value to "resume". o If the previous learner session on the SCO ended due to a Suspend All Navigation Request triggered by the LMS, then the LMS shall initialize this value to "resume". o For all other conditions, the LMS shall set the *cmi.entry* to an empty characterstring (""). • Upon receiving a Terminate("") request or the user navigates away, the LMS shall set the *cmi.entry* to either "" (empty characterstring) or "resume". This value shall be made available during the next learner session in the current learner attempt. This is determined by the LMS by evaluating the value of the *cmi.exit*. If the attempt of the activity is being suspended (either indicated by the SCO by setting the value of *cmi.exit* to "suspend" or by some other LMS provided suspension mechanism such as a Suspend All Navigation Request), then the LMS shall set *cmi.entry* to "resume" upon the next attempt of the activity (that references the associated SCO) for that learner. If the

	SCO set *cmi.exit* to a value other than "suspend" or did not set the value at all, then the LMS will set the *cmi.entry* to an empty characterstring ("").
	SCO Behavior Requirements:
	• The data model element is required to be implemented by the LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.entry* data model element.
	• What the SCO does with this data is totally up to the discretion of the SCO.
	API Implementation Requirements:
	• **GetValue():** The LMS shall return the associated *cmi.entry* value currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
	• **SetValue():** If the SCO invokes a request to set the *cmi.entry*, then the LMS shall set the API Error Code to `404` – Data Model Element Is Read Only and return "false". The LMS shall not alter the state of the data model element based on the request.
	Example:
	• GetValue("cmi.entry")

4.2.8. Exit

The `cmi.exit` data model element indicates how or why the learner left the SCO [1]. This value is used to indicate the reason that the SCO was last exited.

The `cmi.exit` data model element is involved with temporal aspects of the run-time execution of the SCO.

- If the `cmi.exit` is set to "suspend" then the SCOs current learner attempt does not end. The SCOs Run-Time Environment data model element values for the current learner session will be available to the SCO if the SCO is relaunched in a subsequent learner session.
- If the `cmi.exit` is set to "normal", "logout","time-out" or "" (empty characterstring) then the SCOs learner attempt ends. The SCOs Run-Time Environment data model element values of the current learner session will NOT be available if the SCO is relaunched.

ADL Note: If an LMS invokes a Suspend All navigation request, then the value of `cmi.exit` will be ignored. In these cases, the SCOs current learner attempt does not end. The SCOs Run-Time Environment data model element values shall be persisted and available to the SCO if the SCO is relaunched.

Table 4.2.8a: Dot-notation Binding for the Exit Data Model Element

Dot-Notation Binding	Details
cmi.exit	**Data Model Element Implementation Requirements:** • **Data Type:** state (timeout, suspend, logout, normal, _nil_) • **Value Space:** The IEEE defines five state values. SCORM binds these state values to the following restricted vocabulary tokens: o "time-out": The SCO terminated because the time limit specified by max_time_allowed had been exceeded [1]. o "suspend": The learner exited the SCO with the intent of returning to it at the point of exit [1]. o "logout": The SCO signaled a desire to terminate the entire learning activity of which the SCO is part [1]. **ADL Note:** The "logout" keyword is being deprecated and will not be supported in a future version of the SCORM. o "normal": The SCO exited normally [1]. o "" empty characterstring: The exit conditions are undetermined [1]. The default value. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as write-only. • The value is completely controlled by the SCO. The SCO is responsible for setting this value. If the SCO does not set the cmi.exit data model element, then the default value (empty characterstring – "") shall be used. If the LMS receives a request to get the *cmi.exit* value, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. • If there are additional learner sessions within a learner attempt, the *cmi.exit* becomes uninitialized (i.e., reinitialized to its default value of ("") - empty characterstring) at the beginning of each additional learner session within the learner attempt.

Sequencing Impacts:

- If the SCO sets *cmi.exit* to "time-out", the LMS shall process an "Exit All" navigation request when the SCO is taken away, instead of any pending (from the learner or LMS) navigation request.
- If the SCO sets *cmi.exit* to "suspend", the LMS shall set the "Activity is Suspended" value of the learning activity associated with the SCO to true.
- If the SCO sets *cmi.exit* to "logout", the LMS shall process a "Exit All" navigation request when the SCO is taken away, instead of any pending (from the learner or LMS) navigation request.

  ADL Note: The "logout" keyword is being deprecated and will not be supported in a future version of the SCORM.

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as write-only. The SCO is only permitted to store the value of the *cmi.exit* data model element.

API Implementation Requirements:

- **GetValue():** If the SCO invokes a request to get the *cmi.exit*, then the LMS shall set the error code to `405` – Data Model Element Is Write Only and return and empty characterstring ("").
- **SetValue():** This request sets the *cmi.exit* to the supplied value. The value must match one of the restricted vocabularies declared for this data model element. If the value supplied matches one of the tokens listed above, then the LMS shall set the value, return "`true`" and set the error code to `0` - No Error.
  - If the value supplied is not equivalent to one of the tokens listed above, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request.

Additional Behavior Requirements:

For each leaner attempt on a SCO, it is the responsibility of the LMS to make available a *cmi.entry* value of "ab-initio" on the first learner session of that learner attempt. This value indicates that this is the first time the learner is experiencing the SCO during this learner attempt on the SCO. It also indicates that a default (clean) data-model is being accessed by the SCO. The *cmi.entry* value shall be implemented by the LMS as read-only. Since the data model element shall be implemented as read-only, an LMS is responsible for managing this value (the SCO does not directly affect this value – cannot call SetValue() on *cmi.entry*). The LMS is responsible for reacting to other run-time interactions in determining this value. The following explanation is provided for further clarification:

The *cmi.entry* value is directly affected by the *cmi.exit* data model element. The SCO is responsible for setting, if so desired, the *cmi.exit* value. Depending on the value of the *cmi.exit*, the LMS shall react as follows:

1. If the SCO set the *cmi.exit* to "time-out", it is assumed that the learner attempt is ending. Since the learner attempt has ended, the next time the SCO is launched the LMS shall supply a "clean" set of data model values. In addition, the LMS shall end the attempt on the content organization for the learner. This behavior shall be exhibited when a Terminate("") request is received from the SCO that set the *cmi.exit* to "time-out" or the learner navigates away.
2. If the SCO set the *cmi.exit* to "suspend", then the LMS should set the *cmi.entry* to "resume". By setting the *cmi.exit* to "suspend", the SCO is indicating that the learner has exited the SCO with the intent of returning to the SCO at a later time. Since the learner attempt was suspended, once the learner attempt is resumed, the SCO shall have the same set of data that was acquired during the previously suspended learner attempt.
3. If the SCO set the *cmi.exit* to "logout", then this indicates that the SCO has been exited normally (as determined by the SCO developer) and that learner attempt on the SCO ended normally. A subsequent learner attempt on the

	SCO will involve a new set of run-time data. **ADL Note:** The "logout" keyword is being deprecated and will not be supported in a future version of the SCORM. 4. If the SCO set the *cmi.exit* to "normal", then this indicates that the SCO has been exited normally (as determined by the SCO developer) and that learner attempt on the SCO ended normally. A subsequent learner attempt on the SCO will involve a new set of run-time data. 5. If the SCO set the *cmi.exit* to "", then this indicates that the exit state is undetermined and the learner attempt on the SCO ended. A subsequent learner attempt on the SCO will involve a new set of run-time data. How or when the *cmi.entry* is set by the LMS is implementation specific. However, this value shall be available during the next learner session within the learner attempt. **Example:** • SetValue("cmi.exit","suspend").

4.2.9. Interactions

The interactions data model element defines a collection of learner responses that can be passed from the SCO to the LMS. Interactions are intended to be responses to individual questions or tasks that the SCO developer wants to record. There is no implied behavior an LMS shall have when interactions are requested to be set, other than storage of the data.

Interactions can be thought of as a collection of information (interaction data). The interaction data is depicted in Figure 4.2.9a. As defined by the IEEE standard, an LMS is required to support (i.e., store) at least 250 sets of interaction data [1]. An LMS can elect to provide support for more than 250; however, the requirement is to support at least 250 sets of interaction data.

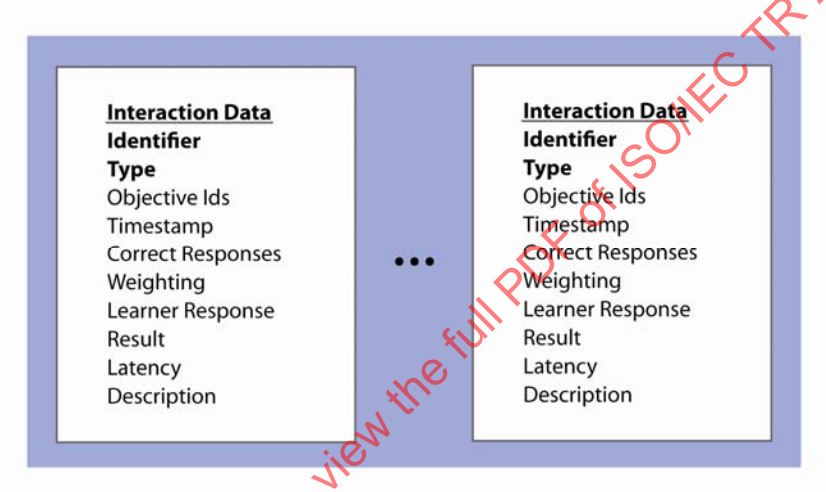

Figure 4.2.9a: Interactions and Interaction Data

There are two important pieces of data that are required to be in the interaction data, an identifier (`cmi.interactions.n.id`) and the type of interaction (`cmi.interactions.n.type`). The `cmi.interactions.n.type` data model element is required if the Interaction includes data pertaining to either the Correct Response or Learner Response. These two pieces of information are what distinguish the interaction data from other interaction data found in the collection of interactions (considered the dependencies of the interaction data). The identifier uniquely identifies one interaction from another within the scope of the SCO. The type uniquely identifies the type of interaction (true-false, matching, etc.).

The Interactions data model element can be used for two primary means by the SCO: journaling and status.

A journaling scheme requires the SCO to record interaction data every time the learner is engaged with the interaction (i.e., new interaction data is appended to the array of interactions). By applying this scheme for recording interactions, information can be gathered to study the learner's experience with the interactions found in the SCO. For

example, reports can be generated that state how many times the learner responded to the interaction, what was the latency for each interaction, what the learner's response was and the result of the response. This data can be gathered and used to potentially update the interaction for future use. LMSs may provide this type of information for independent research and analysis, however, the availability of this data and feature is outside the scope of SCORM.

A status scheme requires the SCO to record interaction data and keep the interaction updated based on the learner's experience with the SCO. For example, if the learner responds to an interaction, information can be set. If the learner then corrects his/her response, the same interaction data is updated to reflect the change (versus a new entry being added to the interactions array). In this scheme, the interaction data contains the last recorded state of the interaction. Also in this scheme, the ability to track how many times the learner updated the response to the interaction is lost.

There are pros and cons for using one particular scheme over another. SCO developers should be aware of the two schemes and use the one they desire. From the LMS perspective, the journaling scheme provides more burden on storage requirements. But then again, the only requirement for the amount of interaction data to store is 250. SCO developers need to understand the smallest permitted maximum of 250 indicates that the LMS is only required to support 250 sets of interaction data.

As with any data model element stored in arrays, the index position (n) is not what sets the uniqueness of the data being stored. The implementation requirements defined in the standard states that the arrays should be implemented as a bag [12]. This data structure allows the same object (interaction data) to be repeated in the array (unlike a set that requires the items in the set to be unique). If using a status scheme, it is recommended that SCOs be built, if using the interaction data model elements, not to rely on the index position for the uniqueness. Depending on the learner and the learning session, the same interaction data may not be stored at the same position in the array. It is highly recommended that the SCO be built to search throughout the interaction data looking for given identifiers, prior to updating the interaction data. If using the journaling scheme described above, then this recommendation does not need to be followed since every time a learner interacts a new entry in the array is created.

Table 4.2.9a: Dot-notation Binding for the Interactions Data Model Element

Dot-Notation Binding	Details
cmi.interactions._children	The *cmi.interactions._children* data model element represents a listing of supported data model elements. This data model element is typically used by a SCO to determine which data model elements are supported by the LMS. The characterstring returned may be used by the SCO to dynamically build parameters for the GetValue() and SetValue() requests. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the data model elements in the Interaction parent data model element that are supported by the LMS. Since all data model elements are required to be supported by the LMS, the characterstring shall

	represent the following data model elements: o id o type o objectives o timestamp o correct_responses o weighting o learner_response o result o latency o description **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for returning a comma-separated list of all of the data model elements (refer to Data Model Element Implementation Requirements above). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.interactions._children* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. The ordering of data model elements is not important. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.interactions._children*, the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "false". **Example:** • GetValue("cmi.interactions._children")
cmi.interactions._count	The *cmi.interactions._count* keyword is used to describe the current number of interactions being stored by the LMS. The total number of entries currently being stored by the LMS shall be returned. **Data Model Element Implementation Requirements:** • **Data Type:** integer • **Value Space:** Non-negative integer • **Format:** The characterstring representing the number of interactions that the LMS is currently persisting. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • If the LMS receives a request to get the *cmi.interactions._count* value prior to any interaction data being set, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.interactions._count* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the number of interactions currently stored by the LMS and set the error code to `0` – No

	Error. o Until interaction data is available for the SCO, the LMS shall return `0`, which indicates that there are no interaction data currently being stored. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.interactions._count*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.interactions._count")
cmi.interactions.n.id	The *cmi.interactions.n.id* data model element is a label for the interaction. [1]. **Data Model Element Implementation Requirements:** • **Data Type:** long_identifier_type • **Value Space:** A characterstring (SPM: 4000) that represents a valid URI as per RFC 3986 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the long_identifier_type data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read/write. • The LMS is only responsible for storage and retrieval of this data model element along with any additional requirements stated in the API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.id* data model element. • SCOs have the responsibility of setting interaction data in a sequential order. All arrays are implemented as packed arrays. Packed arrays are implemented with no array positions being skipped. • The *cmi.interactions.n.id* is required to be set first for each interaction that needs to be tracked by the SCO. • It is recommended that a SCO not alter (set) existing interaction ids during a learner attempt. If the SCO alters an *cmi.interactions.n.id* during a learner attempt, this could corrupt interaction data that has been collected in previous learner sessions and provide inconsistent information about the interaction. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.id* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. • **SetValue():** The LMS shall set the *cmi.interactions.n.id* to the

	supplied value in the SetValue() request, set the error code to `0` – No Error and return "`true`". o If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request . o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. **Additional Behavior Requirements:** The SCO is responsible for making sure that new interaction information is inserted (SetValue()) in the index list in a sequential order. The interaction's identifier is a required field that shall be set by the SCO if interaction data will be requested to be tracked by the SCO. **Example:** • GetValue("cmi.interactions.0.id") • SetValue("cmi.interactions.0.id","obj1")
cmi.interactions.n.type	The *cmi.interactions.n.type* data model element indicates which type of interaction is recorded in an instance of an interaction. It is also used to determine how the interaction response should be interpreted [1]. **Data Model Element Implementation Requirements:** • **Data Type:** state (true_false, multiple_choice, fill_in, long_fill_in, matching, performance, sequencing, likert, numeric, other) • **Value Space:** The IEEE defines ten state values. SCORM binds these state values to the following restricted vocabulary tokens: o "true-false": The interaction has two possible responses [1]. SCORM requires that the two possible responses be either "true" or "false". No abbreviated versions or alternative forms (i.e., t,f,1,0) shall be permitted. o "choice": The interaction has a set of two or more possible responses from which the learner may select [1]. o "fill-in": The interaction requires the learner to supply a short response in the form of one or more strings of characters. Typically, the correct response consists of part of a word, one word or a few words [1]. o "long-fill-in": The interaction requires the learner to supply a response in the form of a long string of characters. o "likert": The interaction asks the learner to select from a discrete set of choices on a scale [1]. o "matching": The interaction consists of two sets of items. Members of the first set are related to zero or more members of the second set. o "performance": The interaction requires the learner to perform a task that requires multiple steps [1]. o "sequencing": The interaction requires the learner to

identify a logical order for members of a list [1].
    - "numeric": The interaction requires a numeric response from the learner. The response is a simple number with an optional decimal point [1].
    - "other": Any other type of interaction not defined by the IEEE standard [1].
- **Format:** The format of the data model value shall be one of the restricted tokens listed above ("true-false", "choice", "fill-in", "long-fill-in", "matching", "performance", "sequencing", "likert", "numeric" or "other").

LMS Behavior Requirements:

- This data model element is mandatory and shall be implemented by an LMS as read/write.
- The LMS is only responsible for storage and retrieval of this data model element along with any additional requirements stated in the API Implementation Requirements.

SCO Behavior Requirements:

- The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.type* data model element.
- SCOs have the responsibility of setting interaction data in a sequential order. All arrays are implemented as packed arrays. Packed arrays are implemented with no array positions being skipped.
- The *cmi.interactions.n.type* is required to be set for each interaction that needs to be tracked by the SCO in which a correct_response and/or learner_response is also being tracked. The *cmi.interactions.n.type* shall be set prior to setting the correct_response or learner_response. If the *cmi.interactions.n.type* is not set prior to the correct_response or learner_response, then a data model dependency is not being met.
- It is recommended that a SCO does not alter (set) existing interaction types during a learner attempt. If the SCO alters an interaction type during a learner attempt, this could corrupt interaction data that has been collected in previous learner sessions reflective of that type. By altering the type of the interaction, correct_response and learner_response data may become invalid.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated *cmi.interactions.n.type* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
    - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
    - If the SCO attempts to retrieve the *cmi.interactions.n.type* and the record of data has been created but the type data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized

	and return an empty characterstring (“”). • **SetValue():** The LMS shall set the *cmi.interactions.n.type* to the supplied value in the SetValue() request, set the error code to `0` – No Error and return “`true`”. o If the SCO invokes a request to set the *cmi.interactions.n.type* and the value is not a member of the restricted vocabulary tokens described above, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return “`false`”. The LMS shall not alter the state of the data model element based on the request . o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.type*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return “`false`” and not change the current state of the data model element. **Example:** • GetValue(“cmi.interactions.0.type”) • SetValue(“cmi.interactions.0.type”,”true-false”)
objectives	
cmi.interactions.n.objectives._count	The *cmi.interactions.n.objectives._count* keyword is used to describe the current number of objectives (i.e., objective identifiers) being stored by the LMS for the given interaction (i.e., the interaction data stored at position *n*). The total number of entries currently being stored by the LMS shall be returned. **Data Model Element Implementation Requirements:** • **Data Type:** integer • **Value Space:** Non-negative integer • **Format:** The characterstring representing the number of objective identifiers that the LMS is currently persisting. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the integer data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • If the LMS receives a request to get the *cmi.interactions.n.objectives._count* value prior to any interaction objective identifiers being set, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.interactions.n.objectives._count* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the number of objective identifiers for the interactions currently stored by the LMS and

	set the error code to `0` – No Error. o Until objective identifiers are requested to be stored by the SCO, the LMS shall return `0`, which indicates that there is no objective identifiers for the interaction data currently being stored. If no objective identifiers are requested to be stored by the SCO, the LMS shall return `0`, if requested by the SCO. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.interactions.n.objectives._count*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.interactions.0.objectives._count")
cmi.interactions.n.objectives.m.id	The *cmi.interactions.n.objectives.m.id* data model element is a label for objectives associated with the interaction. The label shall be unique at least within the scope of the SCO [1]. The objective identifiers may or may not correspond to the objective identifiers found in the Objectives data model element (*cmi.objectives.n.id*). Whether or not there is a relationship to the objective identifiers is implementation specific. The SCO may be designed to track this information and relationship. The *cmi.interactions.n.objectives.m.id* is an array of objective identifiers. The LMS shall maintain an array of at a least 10 (required SPM) of objective identifiers. The LMS may extend the ability to store more, however, this is implementation specific. **Data Model Element Implementation Requirements:** • **Data Type:** long_identifier_type • **Value Space:** A characterstring (SPM: 4000) that represents a valid URI as per RFC 3986 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the long_identifier_type data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read/write. • The LMS is only responsible for storage and retrieval of this data model element along with any additional requirements stated in the API Implementation Requirements. • The LMS shall make be capable of storing at least 10 objective identifiers (SPM requirement). **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.objectives.m.id* data model element. • SCOs have the responsibility of setting interaction data in a sequential order. All arrays are implemented as packed arrays. Packed arrays are implemented with no array positions being skipped. • Prior to setting any associated objective identifiers, the SCO is required to have set the *cmi.interactions.n.id*. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.objectives.m.id* currently stored by the LMS for the interaction and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements

	identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objective identifiers in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (“”). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. • **SetValue():** The LMS shall set the *cmi.interactions.n.objectives.m.id* to the supplied value in the SetValue() request, set the error code to `0` – No Error and return “`true`”. o If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return “`false`”. The LMS shall not alter the state of the data model element based on the request . o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of objectives being stored (can be determined by requesting the *cmi.interactions.n.objectives._count*) or the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.objectives.m.id*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return “`false`” and not change the current state of the data model element. o If the supplied value of the SetValue() is a value that has already been used (not unique within the set of interaction objective identifiers) in an earlier array position within a learner attempt, then the LMS shall set the error code to 351 – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.interactions.0.objectives.0.id”) • SetValue(“cmi.interactions.0.objectives.0.id”,“urn:ADL:objective-id-0001”)
cmi.interactions.n.timestamp	The *cmi.interactions.n.timestamp* data model element is the point in time at which the interaction was first made available to the learner for learner interaction and response [1]. The value of the timestamp is represented as a point in time. If several interactions are presented at the same time, they have the same timestamp value. If an interaction was never available for response, such as an interaction that is not used in an adaptive test, no

timestamp value is available for that interaction. If a timestamp value is available for an interaction but no learner response data is available, it should be assumed the interaction has been available to the learner but the learner did not respond to the interaction.

Data Model Element Implementation Requirements:

- **Data Type:** time(second,10,0)
- **Value Space:** The data type denotes that the value for time is a number expressed as a real data type with values that are accurate to one second and an optional precision of one hundredth of a second (0.01). The number of seconds in the time value is the number of seconds since 00:00 on January 1, 1970 [1].
- **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the time(second,10,0) data type.

LMS Behavior Requirements:

- This data model element is mandatory and shall be implemented by an LMS as read/write.
- The LMS is only responsible for storage and retrieval of this data model element along with any additional requirements stated in the API Implementation Requirements.

SCO Behavior Requirements:

- The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.timestamp* data model element.
- SCOs have the responsibility of setting interaction data in a sequential order. All arrays are implemented as packed arrays. Packed arrays are implemented with no array positions being skipped.
- Prior to setting any associated interaction timestamp, the SCO is required to have set the *cmi.interactions.n.id*.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated interaction timestamp currently stored by the LMS for the interaction and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (“”). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
  - If the SCO attempts to retrieve the *cmi.interactions.n.timestamp* and the record of data has been created but the *cmi.interactions.n.timestamp* has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”).
- **SetValue():** The LMS shall set the *cmi.interactions.n.timestamp* to the supplied value in the SetValue() request, set the error code to `0` – No Error and return “`true`”.
  - If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set

	the error code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.timestamp*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return "`false`" and not change the current state of the data model element. **Example:** • GetValue("cmi.interactions.0.timestamp") • SetValue("cmi.interactions.0.timestamp"2003-07-25T03:00:00")
correct_responses	
cmi.interactions.n.correct_responses._count	The *cmi.interactions.n.correct_responses._count* keyword is used to describe the current number of correct responses being stored by the LMS. The total number of entries currently being stored by the LMS shall be returned. **Data Model Element Implementation Requirements:** • **Data Type:** integer • **Value Space:** Non-negative integer • **Format:** The characterstring representing the number of correct responses that the LMS is currently persisting. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the integer data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • If the LMS receives a request to get the *cmi.interactions.n.correct_responses._count* value prior to any correct responses for the interaction being set, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.interactions.n.correct_responses._count* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the number of correct responses for the interactions currently stored by the LMS and set the error code to `0` – No Error. o Until correct_responses are requested to be stored by the SCO, the LMS shall return `0`, which indicates that there are no correct responses for the interaction data currently being stored. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.interactions.n.correct_responses._count*, then the LMS shall

	set the error code to 404 – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.interactions.0.correct_responses._count")
cmi.interactions.n.correct_responses .n.pattern	The *cmi.interactions.n.correct_responses.n.pattern* data model element defines one correct response pattern for the interaction. The format of the pattern value depends on the type (*cmi.interactions.n.type*) of interaction. The *cmi.interactions.n.correct_responses* collection is a packed array of correct responses for an interaction. Depending on the type (*cmi.interactions.n.type*) of interaction, the number of correct response patterns required to be supported varies. Refer to Section 4.2.9.1: *Correct Responses Pattern Data Model Element Specifics* for more details on each type. **Data Model Element Implementation Requirements:** • The data type and format for the pattern varies depending on the type (*cmi.interactions.n.type*) of interaction. Section 4.2.9.1: *Correct Responses Pattern Data Model Element Specifics* defines the data model element implementation requirements and format for each type of interaction. • During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the pattern based on the interaction type. How the LMS and/or SCO handles and processes the pattern is outside the scope of SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of what the correct responses are for the interaction. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.correct_responses.n.pattern* data model element. The data model element requirements and format for the correct responses are described in detail in Section 4.2.9.1: *Correct Responses Pattern Data Model Element Specifics*. • Prior to setting any associated interaction correct response, the SCO is required to have set the interaction's *cmi.interactions.n.id* and *cmi.interactions.n.type* data model elements. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.correct_responses.n.pattern* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 correct_responses in the array), then the LMS shall set

	the error code to `301` – General Get Failure and return an empty characterstring (“”). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. • **SetValue():** The LMS shall set the *cmi.interactions.n.correct_responses.n.pattern* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return “`true`”. o If either *cmi.interactions.n.id* or *cmi.interactions.n.type* are not set prior to the request to set the *cmi.interactions.n.correct_responses.n.pattern*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return “`false`” and not change the current state of the data model element. o If the SCO tries to set the *cmi.interactions.n.correct_responses.n.pattern* to a value that does not meet the requirements defined in Section 4.2.9.1: *Correct Responses Pattern Data Model Element Specifics* for the interaction’s type (cmi.interactions.n.type), then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return “`false`”. The LMS shall not alter the state of the data model element based on the request. The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (which can be determined by requesting the *cmi.interactions._count*) or correct response patterns being stored (which can be determined by requesting the *cmi.interactions.n.correct_responses._count*), then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. **Example:** • GetValue(“cmi.interactions.0.correct_responses.1.pattern”) • SetValue(“cmi.interactions.0.correct_responses.0.pattern”,”true”)
cmi.interactions.n.weighting	The *cmi.interactions.n.weighting* data model element is the weight given to the interaction that may be used by the SCO to compute a value for a score. The interaction weights typically are used to explain the effect of an interaction on the value of the score but are not intended to be used by systems other than the SCO to compute the score [1]. How this value or any calculation of a total score is computed by the SCO is outside the scope of SCORM. **Data Model Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of the weighting of the interaction unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to

	API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.weighting* data model element. • Prior to setting any associated interaction weighting, the SCO is required to have set the *cmi.interactions.n.id* **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.weighting* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (“”). refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.interactions.n.weighting* and the record of data has been created but the *cmi.interactions.n.weighting* has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** The LMS shall set the *cmi.interactions.n.weighting* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return “`true`”. o If the SCO tries to set the *cmi.interactions.n.weighting* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return “`false`”. The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.weighting*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return “`false`” and not change the current state of the data model element. **Example:** • GetValue(“cmi.interactions.0.weighting”) • SetValue(“cmi.interactions.0.weighting”,”1.0”)
cmi.interactions.n.learner_response	The *cmi.interactions.n.learner_response* data model element consists of

the data generated when a learner responds to an interaction [1]. The learner's response shall have one of ten possible variants. Each variant depends on the type (i.e., *cmi.interactions.n.type*) of interaction.

Refer to Section 4.2.9.2: *Learner Response Data Model Element Specifics* for more details on each type.

Data Model Element Implementation Requirements:

- The data type for the value varies depending on the type (*cmi.interactions.n.type*) of interaction. Section 4.2.9.2: *Learner Response Data Model Element Specifics* defines the data model element implementation requirements and format for each type of interaction.
- During SetValue() and GetValue() processing, LMSs and SCOs need to be aware that the format of the learner response is based on the interaction type. How the LMS and or SCO handles and processes the learner response is outside the scope of SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.

LMS Behavior Requirements:

- The data model element is mandatory and shall be implemented by an LMS as read/write.
- The SCO is responsible for setting this value if appropriate. The LMS is not responsible for making any judgments of whether or not the learner response is correct. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements).

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.learner_response* data model element. The data model element requirements and format for the learner response are described in detail in Section 4.2.9.2: *Learner Response Data Model Element Specifics*.
- Prior to setting any associated interaction learner response, the SCO is required to have set the *cmi.interactions.n.id* and *cmi.interactions.n.type* data model elements.
- The *cmi.interactions.n.learner_response* is not required to be set, if the learner did not respond to an interaction, then it is up to the SCO to determine whether or not a *cmi.interactions.n.learner_response* is set. It should be considered a best practice not to set the *cmi.interactions.n.learner_response* if the learner did not respond to the interaction.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated *cmi.interactions.n.learner_response* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6:

	SCORM Extension Error Conditions for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.interactions.n.learner_response* and the record of data has been created but the learner_response data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (""). • **SetValue():** The LMS shall set the *cmi.interactions.n.learner_response* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`". o If the SCO tries to set the *cmi.interactions.n.learner_response* to a value that does not meet the requirements defined in Section 4.2.9.2: *Learner Response Data Model Element Specifics*, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the *cmi.interactions.n.id* and *cmi.interactions.n.type* are not set prior to the request to set the *cmi.interactions.n.learner_response*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return "`false`" and not change the current state of the data model element. **Example:** • GetValue("cmi.interactions.0.learner_response") • SetValue("cmi.interactions.0.learner_response","true")
cmi.interactions.n.result	The *cmi.interactions.n.result* data model element is a judgment of the correctness of the learner response [1]. **Data Model Element Implementation Requirements:** • **Data Type:** state (correct, incorrect, unanticipated, neutral, real (10,7)) • **Value Space:** The IEEE defines five state values. SCORM binds these state values to the following restricted vocabulary tokens: o "correct": The learner response was correct [1]. o "incorrect": The learner response was incorrect [1]. o "unanticipated": The learner response was not expected [1]. o "neutral": The learner response was neither correct nor incorrect [1]. o real (10,7): A real number. This result provides the capability of reporting a numeric estimate of the correctness of the learner response [1]. • **Format:** The format of the data model value shall be one of the restricted tokens listed above ("correct", "incorrect",

	"unanticipated", "neutral") or a real number with values that is accurate to seven significant decimal figures real. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of the result for the learner response of the interaction unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.result* data model element. • Prior to setting any associated interaction result, the SCO is required to have set the *cmi.interactions.n.id*. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.result* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.interactions.n.result* and the record of data has been created but the *cmi.interactions.n.result* has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (""). • **SetValue():** The LMS shall set the *cmi.interactions.n.result* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`". o If the SCO tries to set the *cmi.interactions.n.result* to a value that does not meet the Data Model Implementation Requirements, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension*

	Error Conditions. o If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.result*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return "`false`" and not change the current state of the data model element. **Example:** • GetValue("cmi.interactions.0.result") • SetValue("cmi.interactions.0.result","1.0") • SetValue("cmi.interactions.0.result","correct")
cmi.interactions.n.latency	The *cmi.interactions.n.latency* data element is the time elapsed between the time the interaction was made available to the learner for response and the time of the first response. The latency information is not available for an interaction if the learner did not respond. The latency is, in effect, the time difference between the *cmi.interactions.n.timestamp* of the interaction and the time of the first response [1]. **Data Model Element Implementation Requirements:** • **Data Type:** timeinterval (second,10,2) - a time interval with resolution to 0.01 seconds • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the timeinterval (second,10,2) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of the latency for the interaction unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.latency* data model element. • Prior to setting any associated interaction latency, the SCO is required to have set the *cmi.interactions.n.id*. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.latency* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.interactions.n.latency* and the record of data has been created but the *cmi.interactions.n.latency* has not

	been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** The LMS shall set the *cmi.interactions.n.latency* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return “`true`”. o If the SCO tries to set the *cmi.interactions.n.latency* to a value that does not meet the Data Model Implementation Requirements, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return “`false`”. The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.latency*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return “`false`” and not change the current state of the data model element. **Example:** • GetValue(“cmi.interactions.0.latency”) • SetValue(“cmi.interactions.0.latency”,” PT5M”) – A period of time of 5 minutes
cmi.interactions.n.description	The *cmi.interactions.n.description* data model element is a brief informative description of the interaction. **Data Model Element Implementation Requirements:** • **Data Type:** localized_string_type (SPM: 250) • **Value Space:** A characterstring that represents a localized characterstring. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments on the description for the interaction unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.interactions.n.description* data model element. • Prior to setting any associated interaction description, the SCO is required to have set the *cmi.interactions.n.id*.

- During a SetValue() request, the SCO should be aware that the {lang=<language_type>} delimiter is optional. If the delimiter is not part of the characterstring, the LMS will assume that the default language is "en".
- During a GetValue() request, the SCO should be aware that the {lang=<language_type>} delimiter may be part of the characterstring returned by the LMS (depending on the LMS implementation). What the SCO does with the characterstring returned by the LMS is dependent on the implementation of the SCO. If the characterstring does not contain the delimiter, then the SCO can assume that the language is "en".

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated *cmi.interactions.n.description* currently stored by the LMS for the interaction and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 interactions in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
  - If the SCO attempts to retrieve the *cmi.interactions.n.description* and the record of data has been created but the *cmi.interactions.n.description* has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring ("").
- **SetValue():** The LMS shall set the *cmi.interactions.n.description* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`".
  - If the SCO tries to set the *cmi.interactions.n.description* to a value that does not meet the Data Model Implementation Requirements, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of interactions being stored (can be determined by requesting the *cmi.interactions._count*), then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*.
  - If the *cmi.interactions.n.id* has not been set prior to the request to set the *cmi.interactions.n.description*, then the LMS shall set the error code to `408` – Data Model Dependency Not Established, return "`false`" and not change the current state of the data model element.

Example:

	• GetValue("cmi.interactions.0.description") • SetValue("cmi.interactions.0.description","Which of the following are red?") - default language of en is used.

4.2.9.1 Correct Responses Pattern Data Model Element Specifics

The `cmi.interactions.n.correct_responses.n.pattern` data model element holds a value that varies depending on the interaction type (`cmi.interactions.n.type`). This section defines the requirements of the data model value for the characterstring value defined by the pattern.

Table 4.2.9.1a: Correct Response Pattern Format Requirements

Interaction Data Type	Correct Response Pattern Format
true_false	The IEEE defines a correct response for the true_false interaction type as: true_false: state(true, false) **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The IEEE defines two state values. SCORM binds these state values to the following restricted vocabulary tokens: o "true": Indicates that the correct response is true. o "false": Indicates that the correct response is false. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall enforce that there may only be one correct response for this type of interaction. If a SCO tries to invoke a SetValue() to store an additional pattern (a correct_responses index, *n*, greater than zero), the LMS shall set the error code to 351 – General Set Failure and return "`false`". The LMS shall not add the new value to the valid list of patterns nor should the LMS alter the state of the currently stored pattern, if one exists. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern", "true")

multiple_choice	The IEEE defines a correct response for the multiple_choice interaction type as:
	
	multiple_choice:
	set of set of short_identifier_type
	// The set SPM: 10 sets
	// The set of short_identifier_type SPM: 36 short_identifier_type
	
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing a set of short_identifier_types, where each element of the set is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 36 (the required SPM) short_identifier_types, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of

	36 short_identifier_types. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type. • **Format:** The format of the characterstring shall be: o <short_identifier_type>[,]<short_identifier_type> The following requirements shall be adhered to when building the characterstring: • The set shall contain one or more sets of short_identifier_types. • The set of short_identifier_types may contain zero or more short_identifier_types. If a set of short_identifier_types is empty, it represents that the correct response is no choice. • If the set contains more than one short_identifier_type (interaction has multiple correct answers – all of which are required), then each short_identifier_type shall be separated by the special reserved token "[,]". • Each short_identifier_type shall occur in the set only once. • The order of the short_identifier_types is insignificant. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall support at least 10 (the required SPM) patterns for the correct responses collection. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 10. • The LMS shall enforce the uniqueness of the correct response sets. If a SCO tries to invoke a SetValue() to store a set that already exists in the correct responses collection, LMS shall set the error code to `351` – General Set Failure and return "`false`". The LMS shall not add the new value to the valid list of patterns nor should the LMS alter the state of any currently stored pattern. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern","choice1[,]choice2[,]choice3") • SetValue("cmi.interactions.0.correct_responses.1.pattern"," choice1[,]choice2") This is an example of a choice interaction where the correct responses collection has two patterns. The API method calls indicate that the correct response is either • "choice1", "choice2" and "choice3", or • "choice1" and "choice2"

fill_in	The IEEE defines a correct response for the fill_in interaction type as:
	fill_in:
	bag of record
	{
	case_matters:
	boolean,
	order_matters:
	boolean,
	match_text:
	array (0..9) of localized_string_type(250),
	// The match_text array SPM: 10 localized_string_types
	// The localized_string_type SPM: 250 characters
	}// The bag of record SPM: 5 records
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing an array of localized_string_types, where each element of the array is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 10 (the required SPM) localized_string_types, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is

implementation specific. The LMS is only responsible for managing the SPM of 10 localized_string_types. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type.

- **Format:** The format of the characterstring shall be the following:
  - {case_matters=<boolean>}{order_matters=<boolean>}<localized_string_type>[,]<localized_string_type>

The following requirements shall be adhered to in building the characterstring:

- The array may contain one or more localized_string_types.
- If the array contains more than one localized_string_type (the interaction has multiple correct answers – all of which are required), then they shall be separated by the special reserved token "[,]".
- Each localized_string may occur more than once.
- The `{case_matters=<boolean>}` delimiter indicates whether or not the case matters for evaluation of the correct response pattern. The case_matters value shall be applied to all of the localized_string_types in the array. The `{case_matters=<boolean>}` may come before or after the `{order_matters=<boolean>}` delimiter. This delimiter is optional; if it is not specified the default value for `case_matters` is "`false`". Refer to Section 4.1.1.6: *Reserved Delimiters* for more information on the requirements for the format of the `{case_matters=<boolean>}` delimiter.
- The `{order_matters=<boolean>}` indicates whether or not the order matters for evaluation of the correct response pattern. The order_matters value shall be applied to all of the localized_string_types in the array. The `{order_matters=<boolean>}` may come before or after the `{case_matters=<boolean>}` delimiter. This delimiter is optional; if it is not specified the default value for `order_matters` is "`true`". Refer to Section 4.1.1.6: *Reserved Delimiters* for more information on the requirements for the format of the `{order_matters=<boolean>}` delimiter.

LMS Behavior Requirements:

- Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection.
- The LMS shall support at least 5 (the required SPM) patterns for the correct responses collection. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 5.

Example:

- SetValue("cmi.interactions.0.correct_responses.0.pattern", "{case_matters=true}{lang=en}car")
- SetValue("cmi.interactions.0.correct_responses.1.pattern", "{case_matters=true}{lang=en}automobile")

This is an example of a fill_in interaction where the correct responses collection has two patterns. The API method calls indicate that the correct response is either "car" or "automobile". The correct response must be all lowercase and the language of the localized_string_type is English.

- SetValue("cmi.interactions.0.correct_responses.0.pattern","{lang=en}car")

This is an example of a fill_in interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is "car". The case of the correct response does not matter; however, the language of the localized_string_type is English.

- SetValue("cmi.interactions.0.correct_responses.0.pattern","car")

This is an example of a fill_in interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is "car". Since

	neither of the reserved delimiters is specified the default values are used. In this example the case of the correct response does not matter; however, the language of the localized_string_type is English. • SetValue("cmi.interactions.0.correct_responses.0.pattern","{case_matters=true}{ order_matters=true}car[,]automobile") This is an example of a fill_in interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is "car" and "automobile" in exactly that order. The correct response must be lowercase and the language of each localized_string_type is English.

long_fill_in	The IEEE defines a correct response for the long_fill_in interaction type as:
	long_fill_in:
	bag of record
	{
	case_matters:
	boolean,
	match_text:
	localized_string_type(4000),
	// The localized_string_type SPM: 4000 characters
	}// The bag SPM: 5 records
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing a localized_string_type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type.
	• **Format:** The format of the characterstring shall be:
	o {case_matters=<boolean>}<localized_string_type>
	The following requirements shall be adhered to when building the characterstring:
	o The characterstring must include one localized_string_type.
	o The `{case_matters=<boolean>}` delimiter indicates whether or not the case matters for evaluation of the correct response pattern. This delimiter is optional; if it is not specified the default value for `case_matters` is "`false`". Refer to Section 4.1.1.6: *Reserved Delimiters* for more information on the requirements for the format of the `{case_matters=<boolean>}` delimiter.
	LMS Behavior Requirements:
	• Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection.
	• The LMS shall support at least 5 (the required SPM) patterns for the correct responses collection. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 5.
	Example:
	The following examples do not include the entire Gettysburg Address. They are written to save space and to describe the data type only. A practical use would include the entire text of the Gettysburg Address.
	• SetValue("cmi.interactions.0.correct_responses.0.pattern", "{case_matters=true}{lang=en}Four score and seven years ago…")
	This is an example of a long_fill_in interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is some portion of the Gettysburg Address. The case of the correct response does matter and the language of the localized_string_type is English.

	• SetValue("cmi.interactions.0.correct_responses.0.pattern","{lang=en}Four score and seven years ago…") This is an example of a long_fill_in interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is some portion of the Gettysburg Address. Since the {case_matters=<boolean>} delimiter is not specified, the default value (false) is used. In this example, the case of the correct response does not matter and the language of the localized_string_type is English.
likert	The IEEE defines a correct response for the likert interaction type as: likert: short_identifier_type **Data Model Element Implementation Requirements:** • **Data Type:** short_identifier_type • **Value Space:** An ISO-10646-1 characterstring representing a short_identifier_type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type. • **Format:** The format of the characterstring shall be: o <short_identifier_type> **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall enforce that there may only be one correct response for this type of interaction – cmi.interactions.n.correct_responses.0.pattern. If a SCO tries to invoke a SetValue() to store an additional pattern (a correct_responses index greater than zero), the LMS shall set the error code to `351` – General Set Failure and return "`false`". The LMS shall not add the new value to the valid list of patterns nor should the LMS alter the state of the currently stored pattern, if one exists. **Example:** • SetValue("cmi.interactions.1.correct_responses.0.pattern","likert_1") This is an example of a likert interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is "likert_1". The SCO is responsible for understanding the significance of this value. For example, the identifier value of likert_1 might have a meaning of "strongly agree".

matching	The IEEE defines a correct response for the matching interaction type as:
	
	matching:
	bag of bag of record
	{
	source:
	short_identifier_type,
	target:
	short_identifier_type,
	}
	
	// The bag of bag SPM: 5 bags
	// The bag of record SPM: 36 records
	
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing a bag of records, where each record consists of two short_identifier_types – a 'source' and a 'target'. The record's short_identifier_types are separated by a special reserved token ("[.]"). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type. Each record in the bag is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 36 (the required SPM) records, separated by

	the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 36 records. • **Format:** The format of the characterstring shall be: o <short_identifier_type>[.]<short_identifier_type>[,]<short_identifier_type>[.]<short_identifier_type> The following requirements shall be adhered to when building the characterstring: o The bag shall contain one or more records. o Each record shall include both a 'source' and a 'target' value separated by the special reserved token "[.]". o The 'source' and 'target' values are short_identifier_types. o If the bag contains more than one record, they shall be separated by the special reserved token "[,]". o There is no restriction on the number of times a short_identifier_type occurs in a given characterstring. o The order of the records is insignificant. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall support at least 5 (the required SPM) patterns for the correct responses collection. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 5. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern", "1[.]a[,]2[.]c[,]3[.]b") This is an example of a matching interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is: **Source** | **Target** 1 → matches → a 2 → matches → c 3 → matches → b The SCO is responsible for understanding the significance of the correct response and its constituent source/target pairs.

performance	The IEEE defines a correct response for the performance interaction type as:
	
	performance:
	bag of record
	(
	order_matters:
	boolean,
	answers:
	array(0..124) of record
	// The array of record SPM: 125 records,
	(
	step_name :
	short_identifier_type,
	step_answer :
	choice
	(
	state(literal, numeric)
	)
	of
	(
	literal :
	characterstring (ISO-10646-1),
	// SPM 250

```
            numeric :
              record
              (
                min :
                 real (10,7),
                max :
                 real (10,7),
              )
          )
      )
  ) // The bag of record SPM: 5 records
```

Data Model Element Implementation Requirements:

- **Data Type:** characterstring
- **Value Space:** An ISO-10646-1 characterstring representing a record that consists of an `{order_matters=<boolean>}` delimiter and an array of records. Each of the array's records consists of one or two of the following values separated by a special reserved token ("[.]"):
  - The 'step name' – A short_identifier_type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type.
  - The 'step answer' – Either a characterstring or a numerical range. If the 'step answer' is a characterstring, the LMS shall support at least 250 (the required SPM) characters. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 250 characters. If the 'step answer' is a numeric range, its format shall be:
    - <min>[:]<max>

  Each record in the array is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 125 (the required SPM) records, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 125 records.
- **Format:** The format of the characterstring shall be:
  - {order_matters}<step_name>[.]<step_answer>[,]<step_name>[.]<step_answer>

The following requirements shall be adhered to when building the characterstring:

- The array shall contain one or more records.
- The `{order_matters=<boolean>}` delimiter indicates whether or not the order of the array records matters for evaluation of the correct response pattern. This delimiter is optional; if it is not specified the default value for `order_matters` is "`true`". Refer to Section 4.1.1.6: *Reserved Delimiters* for more information on the requirements for the format of the `{order_matters=<boolean>}` delimiter.
- Each of the array records shall include either a 'step_name' or a 'step answer', or both, separated by the special reserved token "[.]".
- The 'step name' value is a short_identifier_type.
- The 'step name' value is optional. If the array record does not include a 'step name', then the array record must include a 'step answer' preceded by "[.]".
- If the 'step answer' value is a not a numerical range, it shall be represented as a characterstring with an SPM of 250.
- If the 'step answer' value is a numerical range, it shall exhibit the following format, where <min> and <max> are both of real(10, 7) data type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real(10, 7) data type.
  - <min>[:]<max> – Indicates that the correct response is greater than or

	equal to the minimum value supplied and less than or equal to the maximum value supplied. If the <min> and the <max> are identical numbers, then the correct response is exactly that number. ▪ [:]<max> – Indicates that there is no lower bound for the correct response, only an upper bound. The value must be less than or equal to the maximum value supplied. ▪ <min>[:] – Indicates that there is no upper bound for the correct response, only a lower bound. The value must be greater than or equal to the minimum value supplied. ▪ [:] – Indicates that there is no upper or lower bound. o The 'step answer' value is optional. If the array record does not include a 'step answer', then the array record must include a 'step name' succeeded by "[.]". o If the array contains more than one record, they shall be separated by the special reserved token "[,]". o There is no restriction on the number of times a 'step name' occurs in a given characterstring. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall support at least 5 (the required SPM) patterns for the correct responses collection. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 5. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern", "{order_matters=false}[,]drink coffee[,][.]eat cereal") This is an example of a performance interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is to "drink coffee" and "eat cereal" in any order. The API method call also indicates that no 'step_name' was irrelevant and not provided. • SetValue("cmi.interactions.0.correct_responses.0.pattern","step_1[.]inspect wound[,]step_2[.]clean wound[,]step_3[.]apply bandage") This is an example of a performance interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is to perform "step_1" followed by "step_2" followed by "step_3", in exactly that order, with the corresponding answers.

sequencing	The IEEE defines a correct response for the sequencing interaction type as:
	
	sequencing:
	bag of array (0..35) of short_identifier_type,
	// The bag of array SPM: 5 arrays
	// The array of short_identifier_type SPM: 36 short_identifier_types
	
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing an array of short_identifier_types, where each element of the array is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 36 (the required SPM) short_identifier_types, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 36 short_identifier_types. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type.
	• **Format:** The format of the characterstring shall be:

	○ <short_identifier_type>[,]<short_identifier_type> The following requirements shall be adhered to when building the characterstring: ○ The array may contain zero or more short_identifier_types. ○ If the array contains more than one short_identifier_type (the interaction has multiple correct answers – all of which are required) they shall be separated by the special reserved token "[,]". ○ Each short_identifier_type may occur more than once within the array. ○ The order of the short_identifier_types is significant. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall support at least 5 (the required SPM) patterns for the correct responses collection. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 5. • The LMS shall enforce the uniqueness of the correct responses. If a SCO tries to invoke a SetValue() to store an array that already exists in the correct responses collection, LMS shall set the error code to `351` – General Set Failure and return "`false`". The LMS shall not add the new value to the valid list of patterns nor should the LMS alter the state of any currently stored pattern. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern","a[,]b[,]c") • SetValue("cmi.interactions.0.correct_responses.1.pattern","b[,]c[,]a") This is an example of a sequencing interaction where the correct responses collection has two patterns. The API method calls indicate that the correct response is the sequence of a then b then c, or b then c then a. • SetValue("cmi.interaction.0.correct_responses.1.pattern","buildHouse/buyMaterials[,]buildHouse/buildFoundation[,]buildHouse/buildFirstFloor[,]buildHouse/buildSecondFloor") This is an example of a sequencing interaction where the correct responses collection has one pattern. The API method call indicates that the only correct response is the sequence of a then b then a.
numeric	The IEEE defines a correct response for the numeric interaction type as: numeric: record (min: real (10,7), max: real (10,7)) The numeric correct response is represented as two numbers with an optional decimal point. The numbers may be used to express an inclusive range for the correct response. If the numbers are the same, then a single number is specified as the correct response [1]. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** An ISO-10646-1 characterstring representing a numeric range. • **Format:** The format of the characterstring shall be: ○ `<min>[:]<max>` The following requirements shall be adhered to when building the characterstring: ○ If the characterstring value is a numerical range, it shall exhibit the following format, where <min> and <max> are both of real(10, 7) data type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real(10, 7) data type.

	▪ <min>[:]<max> – Indicates that the correct response is greater than or equal to the minimum value supplied and less than or equal to the maximum value supplied. If the <min> and the <max> are identical numbers, then the correct response is exactly that number. ▪ [:]<max> – Indicates that there is no lower bound for the correct response, only an upper bound. The value must be less than or equal to the maximum value supplied. ▪ <min>[:] – Indicates that there is no upper bound for the correct response, only a lower bound. The value must be greater than or equal to the minimum value supplied. ▪ [:] – Indicates that there is no upper or lower bound. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored at a specific index of the correct responses collection. • The LMS shall enforce that there may only be one correct response for this type of interaction – cmi.interactions.n.correct_responses.0.pattern. If a SCO tries to invoke a SetValue() to store an additional pattern (a correct_responses index greater than zero), the LMS shall set the error code to `351` – General Set Failure and return "`false`". The LMS shall not add the new value to the valid list of patterns nor should the LMS alter the state of the currently stored pattern, if one exists. **Examples:** • SetValue("cmi.interactions.0.correct_responses.0.pattern","4[:]10") This is an example of a numeric interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is some real number between or equal "4" and "10". • SetValue("cmi.interactions.0.correct_responses.0.pattern", "[:]10") This is an example of a numeric interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is some real number less than or equal to "10". • SetValue("cmi.interactions.0.correct_responses.0.pattern", "4[:]") This is an example of a numeric interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is some real number greater than or equal to "4". • SetValue("cmi.interactions.0.correct_responses.0.pattern", "3.14159[:]3.14159") This is an example of a numeric interaction where the correct responses collection has one pattern. The API method call indicates that the correct response is a real number exactly equal to "3.14159".
other	The IEEE defines a correct response for the other interaction type as: other: characterstring (ISO-10646-1) // The characterstring SPM 4000 characters **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** An ISO-10646-1 characterstring • **Format:** The other correct response is used to support other types of interactions not defined by the IEEE standard. The format of the characterstring is left to the implementer of the interaction. **LMS Behavior Requirements:** • Each correct response pattern is represented by a characterstring that conforms to the **Data Model Element Implementation Requirements**, described above, and is stored in a specific index of the correct responses collection. • The LMS shall enforce that there may only be one correct response for this type

	of interaction – cmi.interactions.n.correct_responses.0.pattern. If a SCO tries to invoke a SetValue() to store an additional pattern (a correct_responses index greater than zero), the LMS shall set the error code to `351` – General Set Failure and return "`false`". The LMS shall not add the new value to the valid list of patterns nor should the LMS alter the state of the currently stored pattern, if one exists.

4.2.9.2 Learner Response Data Model Element Specifics

The `cmi.interactions.n.learner_response` data model element holds values that vary depending on the interaction type (`cmi.interactions.n.type`). This section defines the requirements of the data model value for characterstring defined by the learner_response.

Table 4.2.9.2a: Learner Response Format Requirements

Interaction Data Type	Learner Response Format
true_false	The IEEE defines the learner response for the true_false interaction as: true_false: state(true, false) **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The IEEE defines two state values. SCORM binds these state values to the following restricted vocabulary tokens: o "true": Indicates that the learner response is true or a synonym of true (e.g., agree, yes) o "false": Indicates that the learner response is false or a synonym of false (e.g., disagree, no) **Example:** • SetValue("cmi.interactions.0.learner_response","true") This is an example of a true_false interaction where the learner response is "true".
multiple_choice	The IEEE defines the learner response for the multiple choice interaction as: multiple_choice: set of short_identifier_type // The set of short_identifier_type SPM: 36 short_identifier_types **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** An ISO-10646-1 characterstring representing a set of short_identifier_types, where each element of the set is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 36 (the required SPM) short_identifier_types, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 36 short_identifier_types. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type.

	• **Format:** The format of the characterstring shall be: o <short_identifier_type>[,]<short_identifier_type> The following requirements shall be adhered to when building the characterstring: o The set may contain zero or more short_identifier_types. o If the set contains more than one short_identifier_type, then they shall be separated by the special reserved token "[,]". o Each short_identifier_type shall occur in the set only once. o The order of the short_identifier_types is insignificant. **Example:** • SetValue("cmi.interactions.0.learner_response", "choice1[,]choice2[,]choice3") This is an example of a choice interaction where the learner response is "choice1", "choice2" and "choice3".

fill_in	The IEEE defines the learner response for the fill_in interaction type as:
	fill_in:
	{
	array (0..9) of localized_string_type(250),
	// The array SPM: 10 localized_string_type
	// The localized_string type SPM: 250 characters
	}
	
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing an array of localized_string_types, where each element of the array is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 10 (the required SPM) localized_string_types, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 10 localized_string_types. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type.
	• **Format:** The format of the characterstring shall be:
	o <localized_string_type>[,]<localized_string_type>
	
	The following requirements shall be adhered to when building the characterstring:
	o The array may contain one or more localized_string_types.
	o If the array contains more than one localized_string_type (the interaction has multiple correct answers – all of which are required), then they shall be separated by the special reserved token "[,]".
	o Each localized_string may occur more than once.
	o The order of the localized_strings is significant.
	Example:
	• SetValue("cmi.interactions.0.learner_response", "{lang=en}car")
	This is an example of a fill_in interaction where the learner response is "car" and the language of the localized_string_type is English.
	• SetValue("cmi.interactions.0.learner_response","car")
	This is an example of a fill_in interaction where the learner response is "car" and the language of the localized_string_type is English.
	• SetValue("cmi.interactions.0.learner_response","car[,]automobile")
	This is an example of a fill_in interaction where the learner response is "car" followed by "automobile" and the language of both localized_string_types is English.

long_fill_in	The IEEE defines the learner response for the long_fill_in interaction type as:
	
	long_fill_in:
	{
	localized_string_type(4000),
	// The localized_string_type SPM: 4000 characters
	}
	
	Data Model Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Value Space:** An ISO-10646-1 characterstring representing a localized_string_type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type.
	• **Format:** The format of the characterstring shall be:
	o <localized_string_type>
	
	The following requirements shall be adhered to when building the characterstring:
	o The characterstring must include one localized_string_type
	
	Example:
	The following examples do not include the entire Gettysburg Address. They are written to save space and to describe the data type only. A practical use would include the entire text of the Gettysburg Address.
	• SetValue("cmi.interactions.0.learner_response", "{lang=en}Four score and seven years ago…")
	This is an example of a long_fill_in interaction where the learner response is "Four score and seven years ago…" and the language of the localized_string_type is English.
	• SetValue("cmi.interactions.0.learner_response","Four score and seven years ago…")
	This is an example of a long_fill_in interaction where the learner response is "Four score and seven years ago…" and the language of the localized_string_type is English.

likert	The IEEE defines the learner response for the likert interaction type as: likert: short_identifier_type **Data Model Element Implementation Requirements:** • **Data Type:** short_identifier_type • **Value Space:** An ISO-10646-1 characterstring representing a short_identifier_type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type. • **Format:** The format of the characterstring shall be: o <short_identifier_type> **Example:** • SetValue("cmi.interactions.0.learner_response", "strongly_disagree") This is an example of a likert interaction where the learner response is "strongly_disagree". • SetValue("cmi.interactions.1.learner_response","likert_1") This is an example of a likert interaction where the learner response is "likert_1".
matching	The IEEE defines the learner response for the matching interaction type as: matching: bag of record { source:

short_identifier_type,
target:
short_identifier_type,
} // The bag of record SPM: 36 records

Data Model Element Implementation Requirements:

- **Data Type:** characterstring
- **Value Space:** An ISO-10646-1 characterstring representing a bag of records, where each record consists of two short_identifier_types – a 'source' and a 'target'. The record's short_identifier_types are separated by a special reserved token ("[.]"). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type. Each record in the bag is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 36 (the required SPM) records, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 36 records.
- **Format:** The format of the characterstring shall be:
  - <short_identifier_type>[.]<short_identifier_type>[,]<short_identifier_type>[.]<short_identifier_type>

The following requirements shall be adhered to when building the characterstring:

- The bag shall contain zero or more records.
- Each record shall include both a 'source' and a 'target' value separated by the special reserved token "[.]".
- The 'source' and 'target' values are short_identifier_types.
- If the bag contains more than one record, they shall be separated by the special reserved token "[,]".
- There is no restriction on the number of times a short_identifier_type occurs in a given characterstring.
- The order of the records is insignificant.

Example:

- SetValue("cmi.interactions.0.learner_response", "2[.]c[,]1[.]a[,]3[.]b")

This is an example of a matching interaction where the learner response is:

Source		Target
1	→ matches →	a
2	→ matches →	c
3	→ matches →	b

The SCO is responsible for understanding the significance of the correct response and its constituent source/target pairs.

performance

The IEEE defines the learner response for the performance interaction type as:

performance:
array(0..249) of record
// The array of record SPM: 250 records
(
step_name:
short_identifier_type,
step_answer:
choice
(
state(literal, numeric)
)
of
(
literal:
characterstring (ISO-10646-1)

```
        // The characterstring SPM:  250 characters
      numeric:
        real(10,7)
    )
  )
```

Data Model Element Implementation Requirements:

- **Data Type:** characterstring
- **Value Space:** An ISO-10646-1 characterstring representing an array of records. Each of the array's records consists of one or two of the following values separated by a special reserved token ("[.]"):
  - The 'step name' – A short_identifier_type. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type.
  - The 'step answer' – Either a characterstring or a numerical value. If the 'step answer' is a characterstring, the LMS shall support at least 250 (the required SPM) characters. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 250 characters. If the 'step answer' is a numerical value, its format shall be real(10, 7). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type.

  Each record in the array is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 250 (the required SPM) records, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 250 records.
- **Format:** The format of the characterstring shall be:
  - <step_name>[.]<step_answer>[,]<step_name>[.]<step_answer>

The following requirements shall be adhered to when building the characterstring:

- The array shall contain one or more records.
- Each of the array's records shall include either a 'step_name' or a 'step answer', or both, separated by the special reserved token "[.]".
- The 'step name' value is a short_identifier_type.
- The 'step name' value is optional. If the array record does not include a 'step name', then the array record must include a 'step answer' preceded by "[.]".
- If the 'step answer' value is not a numerical range, it shall be represented as a characterstring with an SPM of 250.
- If the 'step answer' is a numerical value, its format shall be real(10, 7). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real(10, 7) data type.
- The 'step answer' value is optional. If the array record does not include a 'step answer', then the array record must include a 'step name' succeeded by "[.]".
- If the array contains more than one record, they shall be separated by the special reserved token "[,]".
- There is no restriction on the number of times a 'step name' occurs in a given characterstring.
- The order of the records is significant.

Example:

- SetValue("cmi.interactions.0.learner_response","step_1[.]inspect wound[,]step_2[.]clean wound[,]step_3[.]apply bandage")

This is an example of a performance interaction where the learner response indicates that the learner performed the following steps in the following order:

1. step_1 → inspect wound
2. step_2 → clean wound

	3. step_3 → apply bandage • SetValue("cmi.interactions.0.learner_response"," step_3[.]apply bandage[,]step_1[.]inspect wound[,]step_2[.]clean wound") This is an example of a performance interaction where the learner response indicates that the learner performed the following steps in the following order: 1. step_3 → apply bandage 2. step_1 → inspect wound 3. step_2 → clean wound
sequencing	The IEEE defines the learner response for the sequencing interaction type as: sequencing: array (0..35) of short_identifier_type // The array of short_identifier_type SPM: 36 short_identifier_types **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** An ISO-10646-1 characterstring representing an array of short_identifier_types, where each element of the array is separated by a special reserved token ("[,]"). The LMS shall support characterstrings that include at least 36 (the required SPM) short_identifier_types, separated by the reserved "[,]" token. The LMS may extend the ability to support more; however, this is implementation specific. The LMS is only responsible for managing the SPM of 36 short_identifier_types. Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the short_identifier_type data type. • **Format:** The format of the characterstring shall be: o <short_identifier_type>[,]<short_identifier_type> The following requirements shall be adhered to when building the characterstring: o The array may contain zero or more short_identifier_types. o If the array contains more than one short_identifier_type (the interaction has multiple correct answers – all of which are required), then they shall be separated by the special reserved token "[,]". o Each short_identifier_type may occur more than once in the array. o The order of the short_identifier_types is significant. **Example:** • SetValue("cmi.interactions.0.learner_response", "a[,]b[,]c") This is an example of a sequencing interaction where the learner response was the sequence of a then b then c. • SetValue("cmi.interaction.0.learner_response","b[,]c[,]a") This is an example of a sequencing interaction where the learner response was the sequence of b then c then a.
numeric	The IEEE defines the learner response for the numeric interaction type as: numeric: real (10,7) **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** An ISO-10646-1 characterstring • **Format:** The characterstring shall represent a real (10,7). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real(10, 7) data type.

	Example: • SetValue("cmi.interactions.0.learner_response","4") This is an example of a numeric interaction where the learner response was the number "4". • SetValue("cmi.interactions.0.learner_response", "10.5") This is an example of a numeric interaction where the learner response was the number "10.5".
other	The IEEE defines the learner response for the other interaction type as: other: characterstring (ISO-10646-1) // The characterstring SPM: 4000 characters **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** An ISO-10646-1 characterstring • **Format:** The other learner response is used to support other types of interactions not defined by the IEEE standard. The format of the characterstring is left to the implementer of the interaction.

4.2.10. Launch Data

During the learning experience, there may be a need to provide the SCO associated with a learning activity with some launch information. This is information that cannot be represented using parameters to the SCO prior to the launch. The `cmi.launch_data` data model element provides the content designer a means to supply this information.

The `cmi.launch_data` data model element provides data specific to a SCO that the SCO can use for initialization. The value of this data model element is not specified [1].

The allowable values for this data model element are defined by the implementer of the SCO. Typically, the documentation for the SCO would specify what data can or has to be provided [1].

This is information that typically cannot be passed as launch parameters to the SCO. The information is typically needed for every use of the SCO, in a certain context.

Table 4.2.10a: Dot-notation Binding for the Launch Data Data Model Element

Dot-Notation Binding	Details
cmi.launch_data	**Data Model Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 4000) • **Value Space:** ISO-10646-1 • **Format:** The format of this characterstring is left to the discretion of the SCO developer. The LMS simply returns the data, if requested to by the SCO (GetValue()). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the characterstring data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read-only. • If launch data is required to operate the SCO, the designers shall provide this launch data using the mechanisms described in SCORM Content Aggregation Model. The ADL Content Package extension namespace provides an element (<adlcp:dataFromLMS>) that is responsible for storing this information. The element can be associated with an <imscp:item> for which the SCO resource is referenced by. The LMS shall initialize the *cmi.launch_data* value using the value supplied in the <adlcp:dataFromLMS> element. If no element or value is present, then the LMS should not make any assumption on how to initialize this value. If an LMS receives a GetValue() request and no information is defined in the manifest, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.launch_data* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.launch_data* data model element and set the error code to `0` - No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If there is no defined <adlcp:dataFromLMS> for the <imscp:item> in which the SCO resource is referenced, then the LMS is not responsible for making an assumption of the initial value for the *cmi.launch_data*. If the SCO requests (GetValue()) the *cmi.launch_data* in these cases, then the LMS shall set the error code

	to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** If the SCO attempts to call SetValue() on the *cmi.launch_data* data model element, then the LMS shall set the error code to `404` - Data Model Element Is Read Only and return “`false`”. The LMS shall not alter the state of the data model element based on the request. **Additional Behavior Requirements:** • The SCO developer is responsible for defining the *cmi.launch_data*. The Content Package Manifest contains an element (<adlcp: dataFromLMS>) that shall be used by the SCO developer for declaring this data. The element is placed in the manifest as a sub-element of the <imscp:item> that references the SCO resource. The LMS is responsible for initializing the *cmi.launch_data* for the referenced SCO with the value defined in the element. The *cmi.launch_data* is permitted to be any valid representation of a characterstring (as defined by the above Data Model Element Implementation Requirements). The LMS shall at least provide the smallest permitted maximum of 4000 characters. A characterstring that is greater than 4000 characters is not guaranteed to be stored in its entirety by an LMS. If no data is defined by the SCO developer in the manifest and the SCO requests the *cmi.launch_data*, then the LMS shall return an empty characterstring (“”) and set the appropriate API Implementation error code. **Example:** • GetValue(“cmi.launch_data”)

4.2.11. Learner Id

The `cmi.learner_id` data model element identifies the learner on behalf of whom the SCO was launched. The `cmi.learner_id` shall be unique at least within the scope of the SCO [1]. How the `cmi.learner_id` is assigned is outside the scope of SCORM. One typical case on how learner_ids are assigned is through some learner registration process defined by the LMS. The `cmi.learner_id` identifies the learner in a given LMS.

Table 4.2.11a: Dot-notation Binding for the Learner ID Data Model Element

Dot-Notation Binding	Details
cmi.learner_id	**Data Model Element Implementation Requirements:** • **Data Type:** long_identifier_type • **Value Space:** A characterstring (SPM: 4000) that represents a valid Universal Resource Identifier (URI) as per RFC 3986 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the long_identifier_type data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS shall be responsible for initializing the *cmi.learner_id*. How this is done is currently outside the scope of SCORM (e.g., this is typically handled via a learner registration system within the LMS). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.learner_id* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated learner identifier currently stored by the LMS for the learner and set error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.learner_id*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "false". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.learner_id")

4.2.12. Learner Name

The `cmi.learner_name` data model element is the name provided for the learner by the LMS [1]. How the `cmi.learner_name` is assigned or created is outside the scope of SCORM. The `cmi.learner_name` may come from some LMS learner registration system or through some learner profile information. There may be other mechanisms for creation and assignment of the `cmi.learner_name`, however there is no restriction on how this process is accomplished.

How the LMS or SCO interprets and processes the delimiter for the characterstring is outside the scope of SCORM. The SCO may be built to parse out the language information prior to utilizing the characterstring returned by the LMS.

Table 4.2.12a: Dot-notation Binding for the Learner Name Data Model Element

Dot-Notation Binding	Details
cmi.learner_name	**Data Model Element Implementation Requirements:** • **Data Type:** localized_string_type (SPM: 250) • **Value Space:** A characterstring that represents a localized characterstring. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the localized_string_type data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for initializing *cmi.learner_name*. How this function is performed is currently outside the scope of SCORM. This function is typically handled by various mechanisms, some of which are described above. **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.learner_name* data model element. • During a GetValue() request, the SCO should be aware that the delimiter may be part of the characterstring returned by the LMS (depending on the LMS implementation). What the SCO does with the characterstring returned by the LMS is dependent on the implementation of the SCO. If no delimiter is provided by the LMS, the SCO shall assume that the characterstring is using the default value (“en”). **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_name* currently stored by the LMS for the learner and set the error code to `0` - No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.learner_name*, the LMS shall set the error code to `404` – Data Model Element Is Read Only and return “false”. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.learner_name”)

4.2.13. Learner Preference

The Learner preference data specifies learner preferences associated with the learner's use of the SCO [1]. There is no restriction on how this learner preference data is determined. For example, the LMS may provide some interface that permits a learner to set this data or some sort of profiling system that captures information about the learner. If such a mechanism is in place, this globally defined data should be used to initialize the set of data model elements defined in this section. Depending how this mechanism is implemented, the data may apply to all of the SCOs in a given content organization, a subset of SCOs or a single SCO.

If a mechanism exists to determine and set the learner preference information before run-time execution of a SCO, then certain obligations and requirements need to be adhered to. Since this data is considered global to the SCOs that are impacted (i.e., may apply to all of the SCOs in a content organization), the data should not be altered if a SCO sets a learner preference via a `SetValue()` call during a learner attempt. If learner preference data is set during a learner attempt, this data is only available for that attempt on the SCO (i.e., temporarily, for the learner attempt, overrides the globally defined preference). If a new attempt is underway, any SCO set learner preferences are lost from the previous learner attempt . The values are initialized back to default values as defined below or default to the global values acquired by another means.

If no such mechanism exists to determine the learner preference information before run-time execution of a SCO, default values or a default behavior have been defined for each data model element. The same requirements apply dealing with the persistence of the data, if set by the SCO. The data should only exist for the learner attempt on the SCO.

Table 4.2.13a: Dot-notation Binding for the Learner Preference Data Model Element

Dot-Notation Binding	Details
cmi.learner_preference._children	The *cmi.learner_preference._children* data model element represents a listing of supported data model elements. This data model element is typically used by a SCO to determine which data model elements are supported by the LMS. The characterstring returned may be used by the SCO to dynamically build parameters for the GetValue() and SetValue() requests. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the data model elements in the Learner Preference parent data model element that are supported by the LMS. Since all data model elements are required to be supported by the LMS, the characterstring shall represent the following data model elements: o audio_level o language o delivery_speed o audio_captioning **LMS Behavior Requirements:** • This data model element is mandatory and shall be

	implemented by an LMS as read-only. • The LMS is responsible for returning a comma-separated list of all of the data model elements (refer to Data Model Element Implementation Requirements above). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the *cmi.learner_preference._children* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. The ordering of data model elements is not important. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.learner_preference._children*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.learner_preference._children")
cmi.learner_preference.audio_level	The *cmi.learner_preference.audio_level* data model element is a multiplier value that specifies an intended change in perceived audio level relative to an implementation-specific reference level with 1 meaning "no change". For example, the value 0 specifies infinite attenuation, the value of 0.5 specifies an attenuation of 10 decibels and the value of 2 specifies an amplification of 10 decibels [1]. **Data Model Element Implementation Requirements:** • **Data Type:** real(10,7), range (0..*) • **Value Space:** A real number greater than or equal to 0. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read/write. • The default *cmi.learner_preference_audio_level* shall be 1. **SCO Behavior Requirements:** • This data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.learner_preference.audio_level* data model element. • If a SCO sets the *cmi.learner_preference.audio_level* during a learner attempt, this value is only persisted by the LMS for that learner attempt. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_preference.audio_level* value currently being stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.learner_preference.audio_level* prior to the value

	being set by the SCO or some other means, then the LMS shall return the default value of `1` and set the error code to `0` – No Error. • **SetValue():** The LMS shall set the *cmi.learner_preference.audio_level* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`". o If the SCO tries to set the *cmi.learner_preference.audio_level* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request. o If the SCO tries to set the *cmi.learner_preference.audio_level* to a value that is a real number, however, the value is not greater than 0 (outside the range), then the LMS shall set the error code to `407` - Data Model Element Value Out Of Range. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.learner_preference.audio_level") • SetValue("cmi.learner_preference.audio_level","3")
cmi.learner_preference.language	The *cmi.learner_preference.language* data model element is the learner's preferred language for SCOs with multilingual capability [1]. **Data Model Element Implementation Requirements:** • **Data Type:** language_type (SPM 250) or empty characterstring • **Value Space:** ISO-646 [4] • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the language_type data type. The default language shall be "" (empty characterstring). **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read/write. • If an LMS does not provide a mechanism for initializing the learner preferences, then the default value for the *cmi.learner_preference.language* should be an empty characterstring (""). **SCO Behavior Requirements:** • This data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.learner_preference.language* data model element. • Since the default value for the *cmi.learner_preference.language* is an empty characterstring, the meaning of this value shall be determined by the SCO. For example, an empty characterstring could mean that the default value is English. An empty characterstring could mean that there is no preferred language and the SCO could be built to determine if the learner has a preferred language. • If a SCO sets the *cmi.learner_preference.language* during a learner attempt, this value is only persisted by the LMS for that learner attempt. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_preference.language* value currently being stored

	by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.learner_preference.language* prior to the value being set by the SCO or some other means, then the LMS shall set the error code to `0` – No Error and return the default value of "" (empty characterstring). • **SetValue():** The LMS shall set the *cmi.learner_preference.language* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`". o If the SCO tries to set the *cmi.learner_preference.language* to a value that does not meet the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.learner_preference.language") • SetValue("cmi.learner_preference.language","fr-CA")
cmi.learner_preference.delivery_speed	The *cmi.learner_preference.delivery_speed* data model element is a multiplier that specifies the learner's preferred relative speed of content delivery expressed as a change in speed relative to an implementation-specific reference speed. For example, 2 is twice as fast as the reference speed and 0.5 is one half the reference speed. The default value shall be 1 [1]. **Data Model Element Implementation Requirements:** • **Data Type:** real(10,7), range (0..*) • **Value Space:** A real number greater than or equal to 0. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read/write. • If an LMS does not provide a mechanism for initializing the learner preferences, then the default value for the *cmi.learner_preference.delivery_speed* should be "1". **SCO Behavior Requirements:** • This data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.learner_preference.delivery_speed* data model element. • If a SCO sets the *cmi.learner_preference.delivery_speed* during a learner attempt, this value is only persisted by the LMS for that learner attempt. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_preference.delivery_speed* value currently being stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation

	Requirements. o If the SCO invokes a request to get the *cmi.learner_preference.delivery_speed* prior to the value being set by the SCO or some other means, then the LMS shall return the default value of `1` and set the error code to `0` – No Error. • **SetValue():** The LMS shall set the *cmi.learner_preference.delivery_speed* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`". o If the SCO tries to set the *cmi.learner_preference.delivery_speed* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request. o If the SCO tries to set the *cmi.learner_preference.delivery_speed* to a value that is a real number, however the value is less than 0 (outside the range), then the LMS shall set the error code to `407` - Data Model Element Value Out Of Range. The LMS shall not alter the state of the data model element based on the request. **Additional Behavior Requirements:** The SCO is responsible for setting the speed value based on a learner's preference. How this value is collected, determined or applied to the SCO is outside the scope of SCORM. **Example:** • GetValue("cmi.learner_preference.delivery_speed") • SetValue("cmi.learner_preference.delivery_speed","0.5")
cmi.learner_preference.audio_captioning	The *cmi.learner_preference.audio_captioning* data model element specifies whether captioning text corresponding to audio is displayed [1]. **Data Model Element Implementation Requirements:** • **Data Type:** state (off, no_change, on) • **Value Space:** The IEEE defines three state values. SCORM binds these state values to the following restricted vocabulary tokens: o `-1`: Represents the "off" state. The captioning is off and text corresponding to audio is not displayed [1]. o `0`: Represents the "no_change" state. The current default captioning setting is maintained [1]. This is the default value. o `1`: Represents the "on" state. The captioning is on and text corresponding to audio is displayed [1]. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read/write. **SCO Behavior Requirements:** • This data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.learner_preference.audio_captioning* data model element. • If a SCO sets the *cmi.learner_preference.audio_captioning* during a learner attempt, this value is only persisted by the LMS for that learner attempt. **API Implementation Requirements:**

	• **GetValue():** The LMS shall return the associated *cmi.learner_preference.audio_captioning* value currently being stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.learner_preference.audio_captioning* prior to the value being set by the SCO or some other means, then the LMS shall return the default value of `0` and set the error code to `0` – No Error. • **SetValue():** If the SCO invokes a request to set the *cmi.learner_preference.audio_captioning* and the value is not a member of the restricted vocabulary tokens described above; then the LMS shall set the API Error Code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request. **Additional Behavior Requirements:** How this data model element is initialized is outside the scope of SCORM. The value may be initialized in a number of ways. For example: • SCO may be built to set the initial audio captioning value based on a learner's preference, or • the value may be initialized by some learner preference or profile data collected by the LMS. How this value is collected, determined or applied to the SCO is outside the scope of SCORM. **Example:** • GetValue("cmi.learner_preference.audio_captioning") • SetValue("cmi.learner_preference.audio_captioning","-1")

4.2.14. Location

The `cmi.location` data model element is a location in the SCO. Its value and meaning are defined by the SCO. The first time the learner attempts the SCO or if there is no preferred initial location, the value shall be an empty characterstring ("") [1].

The LMS should not interpret or change this data. The data is opaque to the LMS. The format of the `cmi.location` is only understood by the SCO setting the value. If the SCO communicates a `cmi.location`, this data model element may be used to indicate a "bookmark" or "checkpoint" within the SCO. This data model element may be used as a starting point upon re-entry into the SCO after a suspended learner session. In this case, the `cmi.location` data model element corresponds to the SCO exit point the last time the learner experienced the SCO. The behavior and use of this data model element is managed by the SCO.

Table 4.2.14a: Dot-notation Binding for the Location Data Model Element

Dot-Notation Binding	Details
cmi.location	**Data Model Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 1000) • **Value Space:** ISO-10646-1 [5] • **Format:** The format of this characterstring is left to the discretion of the SCO developer. The LMS simply stores the data, if requested to by the SCO (SetValue()), and returns the data, if requested by the SCO (GetValue()). Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the characterstring data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • This data model element is controlled by the SCO. No initialization steps are required by the LMS. If a GetValue() request is made prior to the value being set by the SCO, then the LMS shall behave according to the API Implementation Requirements defined below. **SCO Behavior Requirements:** • The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.location* data model element. • If storing the data model element, then the value is SCO-implementation defined and the format of the characterstring shall adhere to ISO 10646-1. The LMS is responsible for simply storing this data and then returning the data when requested by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.location* currently stored by the LMS for the learner and set the error code to `0` - No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If the SCO requests the *cmi.location* prior to the value being initialized by the SCO, then the LMS shall return an empty characterstring ("") and set the error code to `403` – Data Model Element Value Not Initialized. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.location* and the supplied value meets the requirements defined in the Data Model Element Implementation Requirements, then the LMS shall store the

	supplied value for *cmi.location*, set the error code to `0` – No Error and return "`true`". **Example:** • GetValue("cmi.location") • SetValue("cmi.location","chkPt1.p3.f5")

4.2.15. Maximum Time Allowed

The `cmi.max_time_allowed` data model element is the amount of accumulated time the learner is allowed to use a SCO in the learner attempt. [1]. The learner attempt begins with the beginning of the first learner session and continues until the activity terminates.

Table 4.2.15a: Dot-notation Binding for the Max Time Allowed Data Model Element

Dot-Notation Binding	Details
cmi.max_time_allowed	**Data Model Element Implementation Requirements:** • **Data Type:** timeinterval (second,10,2) - a time interval with resolution to 0.01 seconds • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the timeinterval (second,10,2) data type. **LMS Behavior Requirements:** • This data model element is mandatory and the LMS shall be implemented by the LMS as read-only. • The LMS is responsible for initializing this value based on the value provided by the SCO developer. This value can be provided in the Content Package Manifest associated with the content organization. The LMS shall use the imsss:attemptAbsoluteDurationLimit attribute of the <imsss:limitConditions> element, if defined for the <imscp:item> referencing the SCO Resource in the manifest, to initialize this value. **SCO Behavior Requirements:** • This data model element is required to be implemented by the LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.max_time_allowed* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.max_time_allowed* data model element and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o If there is no maximum time allowed (i.e., imsss:attemptAbsoluteDurationLimit attribute for the <imsss:limitConditions> element) defined in the manifest and the SCO requests the value for this data model element, then the LMS shall set the error code to `403` – Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.max_time_allowed*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return “`false`”. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.max_time_allowed”)

4.2.16. Mode

The `cmi.mode` data model element identifies one of three possible modes in which the SCO may be presented to the learner [1]. This value can be used to indicate a SCO's behavior after launch. Many SCOs have a single "behavior." Some SCOs, however, can present different amounts of information, present information in different sequences, present information reflecting different training strategies or store different sets of data based on the mode that the SCO is currently in.

Table 4.2.16a: Dot-notation Binding for the Mode Data Model Element

Dot-Notation Binding	Details
cmi.mode	**Data Model Element Implementation Requirements:** • **Data Type:** state (browse, normal, review) [1] • **Value Space:** The IEEE defines three state values. SCORM binds these state values to the following restricted vocabulary tokens: o "browse": The SCO is presented without the intent of recording any information about the current learner session [1]. o "normal": The SCO is presented with the intent of recording information about the current learner session [1]. This is the default value if no mechanism is in place to identify the mode. o "review": The SCO has previously recorded information about the learner attempt and is presented without the intent of updating this information with data from the current learner session [1]. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • There is currently no mechanism in place to determine the mode of a SCO. This is currently left to the implementation of an LMS. If the LMS wants to provide a way of previewing (or browsing) a content organization or a way of reviewing a content organization, then this is one mechanism for initializing the *cmi.mode* in which the content (SCO) in the content organization should be viewed. The "normal" mode shall be the default mode for all SCOs. **Sequencing Impacts:** • The *cmi.mode* data model element has no impact on sequencing. The value of *cmi.mode* does not impact any sequencing behaviors. If the *cmi.mode* value is "browse" or "review", the LMS should treat any data sent by the SCO as informative (in order to make sequencing decisions). Whether or not an LMS persists any of the data sent by the SCO, while in a mode of review or browse, is outside the scope of the SCORM. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.mode* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.mode* currently stored by the LMS for the learner and set the error code to `0` – No Error. If a mechanism is not in place to support different modes, then the LMS shall only return "normal" for all cases. • **SetValue():** If the SCO invokes a request to set the *cmi.mode*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". The LMS shall not alter the state of the data model element based on the request. **Example:**

	• GetValue("cmi.mode")

4.2.16.1 Mode and Credit Usage Requirements

The `cmi.mode` and `cmi.credit` data model elements have a relationship to one another. The following table describes the relationships:

Table 4.2.16.1a: Mode and Credit Values

cmi.mode value	*cmi.credit* value
"browse"	"no-credit"
"review"	"no-credit"
"normal"	"credit" or "no-credit"

An LMS is required to implement these data model elements as read-only. The SCO is not permitted to alter the states of these values with a `SetValue()` API call. The LMS must guarantee that these values are kept in sync with each other. If credit is "`no-credit`", then any information (success_status, interactions, etc.) communicated by the SCO is informative. The data recorded during the learner's interaction with the SCO is used to make sequencing decisions. Other than that, the LMS is free to do what it wants with the data.

4.2.17. Objectives

Instructional designers may wish to associate learning or other types of objectives with a learning activity and its associated content object. SCORM does not define what an objective is or place requirements on its use. However, SCORM does define how the status of objectives, regardless of type, may be tracked during a learner experience, and how tracked objective status may be used to affect sequencing evaluations.

For purposes of a learner experience, objectives are tracked by associating a set of objective status information with each tracked objective through the use of an identifier. The identifier used does not have or imply any semantics; it is only used to relate the results of a learner experience with a content developer defined objective.

The Objectives are treated as a grouping of sets of objective status information for a given SCO, which are used to track learning or other types of objectives associated with the SCO. A SCO may have zero or more sets of objective status information. Information tracked in the Objectives parent data model element is only available to the SCO, and it is available for the duration of a learner attempt. An identifier is required to differentiate between sets of objective status information. For a given SCO, all objective identifiers must be unique.

Each set of objective status information consists of the following elements:

- Identifier: an identifier for the set of objective status information
- Score: a score (if applicable)
- Success Status: an indication of the success status for the objective (if applicable)
- Completion Status: an indication of the completion status for the objective (if applicable)
- Progress Measure: an indication of a measure towards the progress of completing the associated objective.
- Description: a brief, informative description of the objective

During the learner's interaction with the SCO, the SCO is permitted to update status/score information and not have to create a new entry in the array. The objectives array should act as a status of the objectives during the learner's interaction. The objective identifier is what makes the objective unique, therefore when new objective status information is needed to be tracked (new objective identifier), then a new entry in the array should be made.

SCO developers can describe the tracked objectives of the SCO – the significance of objective identifiers – by using the Classification category of the LOM metadata. The Purpose, Taxonpath and Description elements can also be used to describe and identify the SCO's objective(s). However, applying metadata to a SCO is for informative purposes only. Currently there is no requirement defined for an LMS to process the metadata associated with a SCO and there is no behavior defined for an LMS to interoperably utilize the metadata associated with a SCO.

The LMS shall support at least the SPM of 100 objective status information. The LMS is free to support more than the SPM.

4.2.17.1 Associating Objective Status Information to SCOs

More than one set of objective status information may be associated with a given SCO. The mechanism defined to access multiple sets of objective status information is an indexed list (refer to Section 4.1.1.3: *Handling Collections*). When setting (`SetValue()`) objective status information, the SCO is responsible for inserting objective status information in sequential order.

Table 4.2.17.1a: Scenarios for Storing Collection Data

Incorrect	Correct
cmi.objectives.0.id = "ID1" cmi.objectives.2.id = "ID3" cmi.objectives.1.id = "ID2"	cmi.objectives.0.id = "ID1" cmi.objectives.1.id = "ID2" cmi.objectives.2.id = "ID3"

To avoid the incorrect scenario shown above, in Table 4.2.17.1a, the SCO can invoke `GetValue()` on `cmi.objectives._count` to retrieve the next available position in the indexed list. The LMS is responsible for managing this list as a zero-based indexed list. When a SCO requests the `cmi.objectives._count`, the LMS shall return the total number of objectives currently being managed by the LMS. For example, if the LMS returns "`2`" from the `GetValue("cmi.objectives._count")` request, the SCO shall use this value in inserting the next set of objective information. The "`2`" indicates that the SCO has already set (`SetValue()`) `cmi.objectives.0 and cmi.objectives.1` objective information.

The order of the sets of objective status information in the indexed list does not define a significant relationship between the objectives and the order may change between learner sessions. The recommended method of accessing sets of objective status information is to search the indexed list for the desired objective identifier. The index where the identifier is found should be used to access and modify the other elements of objective status information.

4.2.17.2 Initialization of Run-Time Objectives from Sequencing Information

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

SCORM defines how identifiers are used to relate the objectives defined on the learning activity for sequencing purposes to the objectives status information (`cmi.objectives`)

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

1. **`cmi.objectives.n.id`** shall be initialized with the objective's ID (the objective ID attribute associated to the objective in the activity's sequencing information).
2. **`cmi.objectives.n.success_status`** shall be initialized with the Tracking Information associated with the objective (possibly through a **read** Objective Map).
3. **`cmi.objectives.n.score.scaled`** shall be initialized with the Tracking Information associated with the objective (possibly through a **read** Objective Map).

When the new learner attempt on the SCO begins and after LMS initialization of the SCO's objective data model elements for that objective, the number of objectives being stored by the LMS (`cmi.objectives._count`) shall equal the number of objectives defined in the sequencing information (children of the <imsss:objectives> element) applied to the SCO's associated activity that have defined Objective IDs, regardless of whether the activity is tracked or not. During the learner experience, the SCO may modify the status of these objectives; the updated status information will be used by the LMS during sequencing evaluations.

When a subsequent learner session on the SCO begins within the same learner attempt, the LMS shall update the SCO's objective data model elements (`cmi.objectives.n.xxx`) for the objectives defined in the activity's sequencing information, using the most recent information available to the LMS's sequencing implementation. Any additional objectives created by the SCO during a previous learner session will not be affected.

ADL Note:

- If a learner attempt on a SCO is suspended and then later resumed, the LMS is required to update the SCO's objective data model elements values to reflect changes in sequencing Tracking Information occurring with any of the SCO's associated activity's objectives while the SCO is suspended.
- The SCO's objective data model elements shall be initialized even if the SCO's associated activity is not tracked (`tracked = False`). In these cases, 'read' objective maps shall be applied to initialize objective state. When no 'read' objective maps are defined, the initialized objectives shall have default (`unknown`) state.

- Run-time data related to objectives (`cmi.objectives.n.xxx`) will not be initialized for an activity's associated SCO unless an objective ID attribute is defined in the sequencing information. In addition, if multiple objectives are defined for an activity, the order of the SCO's objectives (`cmi.objectives.n.xxx`) need not coincide with the order of the activity's objectives – only the IDs are significant.

Table 4.2.17a: Dot-notation Binding for the Objectives Data Model Element

Dot-Notation Binding	Details
cmi.objectives._children	The *cmi.objectives._children* data model element represents a listing of supported data model elements. This data model element is typically used by a SCO to determine which data model elements are supported by the LMS. The characterstring returned may be used by the SCO to dynamically build parameters for the GetValue() and SetValue() requests. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the data model elements in the Objectives parent data model element that are supported by the LMS. Since all data model elements are required to be supported by the LMS, the characterstring shall represent the following data model elements: o id o score o success_status o completion_status o progress_measure o description **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for returning a comma-separated list of all of the data model elements (refer to Data Model Element Implementation Requirements above). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.objectives._children* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. The ordering of data model elements is not important. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.objectives._children*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.objectives._children")
cmi.objectives._count	The *cmi.objectives._count* keyword is used to describe the current number of objectives being stored by the LMS. The total number of

	entries currently being stored by the LMS shall be returned. The LMS is responsible for supporting the smallest permitted maximum of 100 objectives. **Data Model Element Implementation Requirements:** • **Data Type:** integer • **Value Space:** Non-negative integer • **Format:** The characterstring representing the number of objectives that the LMS is currently persisting. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read-only. • If the LMS receives a request to get the *cmi.objectives._count* value prior to any objectives being set, then the LMS shall adhere to the requirements listed below for API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.objectives._count* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the number of objectives currently stored by the LMS and set the error code to `0` – No Error. o Until objective information is available for the SCO, the LMS shall return `0`, which indicates that there is no objective information currently being stored. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.objectives._count*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.objectives._count")
cmi.objectives.n.id	The *cmi.objectives.n.id* data model element is an identifier for an objective and shall be unique, at least within the scope of the SCO. The *cmi.objectives.n.id* data model element shall contain a valid value if either the score or status data model elements described below is implemented [1]. If a SCO is requesting to store objective information, then the SCO is required to set the identifier first (unless it was initialized by another means), prior to any other objective information. Once the *cmi.objectives.n.id* has a value, the data model element is not allowed to be reset to a different value. **Data Model Element Implementation Requirements:** • **Data Type:** long_identifier_type • **Value Space:** A characterstring (SPM: 4000) that represents a valid Universal Resource Identifier (URI) as per RFC 3986 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the long_identifier_type data type. **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read/write. The data model element is only permitted to be written to once. SCOs are not permitted to change the *cmi.objectives.n.id* value once it has been established. The value may be set more than once, as long as the value stays the same. • If the <imsss:objectives> are defined for an <imscp:item> element in the Content Package Manifest, then the LMS is

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

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.id* data model element.
- If a SCO is requesting to store objective information, a SCO shall ensure that an id is set to uniquely distinguish one objective from another. The identifier shall be set first (unless it was initialized by another means), prior to any other objective information.
- It is recommended that a SCO does not alter (set) existing objective IDs during a learner attempt. If the SCO alters an objective ID during a learner attempt, this could corrupt objective data that has been collected in previous learner sessions and have impact on sequencing decisions made by the LMS.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated objectives identifier currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
    - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objectives in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
- **SetValue():** The LMS shall set the *cmi.objectives.n.id* to the supplied value in the SetValue() request, set the error code to `0` – No Error and return "`true`".
    - If the supplied value of the SetValue() does not meet the requirements of the Data Model Element Implementation Requirements, then the LMS shall set the error code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request .
    - Collection data model elements are required to be set in sequential order. If a SCO does not set objectives in a sequential order, then the LMS shall set the error code to `351` – General Set Failure and return

	"`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o If the supplied value of the SetValue() is a value that has already been used (not unique within the set of objective information) in an earlier array position within a learner attempt, then the LMS shall set the error code to 351 – General Set Failure and return "false". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. The LMS shall not alter the state of the data model element based on the request. o If the *cmi.objectives.n.id* data model element has already been established with a value (either through a mapping based on the *Objective Progress Information* defined for the associated Activity or an explicit SetValue() call by the SCO) and the SCO issues a SetValue() call to change the current value to a different value, the LMS shall set the error code to 351 – General Set Failure and return "false". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. The LMS shall not alter the state of the data model element based on the request. • If the value in the SetValue() request is identical to the current value, then the LMS shall set the error code to 0 – No Error and return "true". **Additional Behavior Requirements:** • The SCO is responsible for making sure that new objective information is inserted (SetValue()) in the index list in a sequential order. The *cmi.objectives.n.id* is required to be set first prior to any other objective information. **Example:** • GetValue("cmi.objectives.0.id") • SetValue("cmi.objectives.0.id","obj1")
cmi.objectives.n.score	
cmi.objectives.n.score._children	The *cmi.objectives.n.score._children* data model element represents a listing of supported data model elements. This data model element is typically used by a SCO to determine which data model elements are supported by the LMS. The characterstring returned may be used by the SCO to dynamically build parameters for the GetValue() and SetValue() requests. **Data Model Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the data model elements in the Score parent data model element that are supported by the LMS. Since all data model elements are required to be supported by the LMS, the characterstring shall represent the following data model elements: o scaled o raw o min o max **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for returning a comma-separated list of all of the data model elements (refer to Data Model Element

	Implementation Requirements above). **SCO Behavior Requirements:** • This data model element is required to be implemented by an LMS as read-only. The SCO is only permitted to retrieve the value of the *cmi.objectives.n.score._children* data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (refer to Data Model Element Implementation Requirements above) and set the error code to `0` – No Error. The ordering of data model elements is not important. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.objectives.n.score._children*, then the LMS shall set the error code to `404` – Data Model Element Is Read Only and return "`false`". **Example:** • GetValue("cmi.objectives.0.score._children")
cmi.objectives.n.score.scaled	The *cmi.objectives.n.score.scaled* data model element is a number that reflects the performance of the learner for the objective. The value of the data model element is scaled to fit the range –1 to 1 inclusive [1]. If there is sequencing information applied to the learning activity associated with the SCO that relies on a measure, the SCO should ensure score information is accurately sent to the LMS (SetValue()) prior to the SCO's learner session ending. If the measure is not reported by the SCO, then the LMS will use the value "unknown" as the objective measure for the appropriate objective (based on objective IDs) of the learning activity associated with the SCO when processing sequencing information. **Data Model Element Implementation Requirements:** • **Data Type:** real (10,7) range (-1..1) • **Value Space:** A real number with a value that is accurate to seven significant decimal figures. The value shall be in the range of -1.0 to 1.0, inclusive. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by the LMS as read/write. • The SCO is responsible for determining the *cmi.objectives.n.score.scaled*. The LMS cannot make any judgments of the objective's scaled score unless reported otherwise from the SCO. If an LMS receives a retrieve (GetValue()) request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **Sequencing Impacts:** • If the SCO does not set *cmi.objectives.n.score.scaled* for an objective of the SCO, the *Objective Measure Status* for the associated objective (based on objective IDs) of the learning activity associated with the SCO shall be false. • If the SCO sets *cmi.objectives.n.score.scaled* for an objective of the SCO, the *Objective Measure Status* for the objective (based on objective IDs) of the learning activity associated with the SCO shall be true, and the *Objective Normalized*

Measure for the objective (based on objective IDs) of the learning activity associated with the SCO shall equal the value of score.scaled.

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.score.scaled* data model element.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.scaled* currently stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objectives in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
  - If the SCO attempts to retrieve the *cmi.objectives.n.score.scaled* and the record of data has been created but the scaled data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring ("").
- **SetValue():** The LMS shall set the *cmi.objectives.n.score.scaled* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`".
  - If the SCO tries to set the *cmi.objectives.n.score.scaled* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request.
  - If the SCO tries to set the *cmi.objectives.n.score.scaled* to a value that is a real number but the value is not in the range of –1 to 1, inclusive, then the LMS shall set the error code to `407` - Data Model Element Value Out Of Range, return "`false`". The LMS shall not alter the state of the data model element based on the request.
  - Since the *cmi.objectives.n.id* is required to be set first prior to any other objective information, if the SCO attempts to set *cmi.objectives.n.score.scaled* (prior to setting the identifier), then the LMS shall set the error code to `408` – Data Model Dependency Not Established and return "`false`". The LMS shall not alter the state of the data model element based on the request.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number

	that is greater than the current number of objectives being stored, then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. **Example:** • GetValue("cmi.objectives.0.score.scaled") • SetValue("cmi.objectives.0.score.scaled","0.750033") • SetValue("cmi.objectives.0.score.scaled","0.75")
cmi.objectives.n.score.raw	The *cmi.objectives.n.score.raw* data model element is a number that reflects the performance of the learner, for the objective, relative to the range bounded by the values of min and max [1]. **Data Model Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of the objective's raw score unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall behave in accordance with the API Implementation Requirements. **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.score.raw* data model element. The raw score for the objective may be determined and calculated in any manner and is controlled by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.raw* currently stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objectives in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.objectives.n.score.raw* and the record of data has been created but the raw data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring ("""). • **SetValue():** The LMS shall set the *cmi.objectives.n.score.raw*

	data model element to the supplied value passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`". o If the SCO tries to set the *cmi.objectives.n.score.raw* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of objectives being stored, then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o Since the *cmi.objectives.n.id* is required to be set first prior to any other objective information, if the SCO attempts to set *cmi.objectives.n.score.raw* (prior to setting the identifier), then the LMS shall set the error code to `408` – Data Model Dependency Not Established and return "`false`". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.objectives.0.score.raw") • SetValue("cmi.objectives.0.score.raw","75.0033") • SetValue("cmi.objectives.0.score.raw","0.75")
cmi.objectives.n.score.min	The *cmi.objectives.n.score.min* data model element is the minimum value, for the objective, in the range for the raw score [1]. **Data Model Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of the objective's minimum score unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements). **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.score.min* data model element. The minimum score is determined by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.min* currently stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element

	Implementation Requirements. o The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objectives in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (“”). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request. o If the SCO attempts to retrieve the *cmi.objectives.n.score.min* and the record of data has been created but the min data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring (“”). • **SetValue():** The LMS shall set the *cmi.objectives.n.score.min* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return “`true`”. o If the SCO tries to set the *cmi.objectives.n.score.min* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return “`false`”. The LMS shall not alter the state of the data model element based on the request. o The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of objectives being stored, then the LMS shall set the error code to `351` – General Set Failure and return “`false`”. Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*. o Since the *cmi.objectives.n.id* is required to be set first prior to any other objective information, if the SCO attempts to set *cmi.objectives.n.score.min* (prior to setting the identifier) then the LMS shall set the error code to `408` – Data Model Dependency Not Established and return “`false`”. The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue(“cmi.objectives.0.score.min”) • SetValue(“cmi.objectives.0.score.min”,”1.0”) • SetValue(“cmi.objectives.0.score.min”,”500”)
cmi.objectives.n.score.max	The *cmi.objectives.n.score.max* data model element is the maximum value, for the objective, in the range for the raw score [1]. **Data Model Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. • **Format:** Refer to Section 4.1.1.7: *Data Types* for more information on the requirements for the format of the real (10,7) data type. **LMS Behavior Requirements:** • The data model element is mandatory and shall be implemented by an LMS as read/write.

- The SCO is responsible for setting this value if appropriate. The LMS cannot make any judgments of the objective's maximum score unless reported otherwise from the SCO. If an LMS receives a GetValue() request prior to the value being set by the SCO, then the LMS shall set the appropriate error code (refer to API Implementation Requirements).

SCO Behavior Requirements:

- The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.score.max* data model element. The max score is determined by the SCO.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.max* currently stored by the LMS for the learner and set the API Implementation's error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objectives in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
  - If the SCO attempts to retrieve the *cmi.objectives.n.score.max* and the record of data has been created but the max data model element has not been set by the SCO, then the LMS shall set the error code to `403` - Data Model Element Value Not Initialized and return an empty characterstring ("").
- **SetValue():** The LMS shall set the *cmi.objectives.n.score.max* data model element to the parameter passed as parameter_2 of the SetValue() call, set the error code to `0` - No Error and return "`true`".
  - If the SCO tries to set the *cmi.objectives.n.score.max* to a value that is not a real number, then the LMS shall set the error code to `406` - Data Model Element Type Mismatch, return "`false`". The LMS shall not alter the state of the data model element based on the request.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of objectives being stored, then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*.
  - Since the *cmi.objectives.n.id* is required to be set first prior to any other objective information, if the SCO attempts to set *cmi.objectives.n.score.max* (prior to setting the identifier) then the LMS shall set the error code to `408` – Data Model Dependency Not Established and return "`false`". The LMS shall not alter the state of the data model element based on the request.

	Example: • GetValue("cmi.objectives.0.score.max") • SetValue("cmi.objectives.0.score.max","1.0") • SetValue("cmi.objectives.0.score.max","500")
cmi.objectives.n.success_status	The *cmi.objectives.n.success_status* data model element indicates whether the learner has mastered the objective [1]. How the SCO determines the *cmi.objectives.n.success_status* for the objective is outside the scope of SCORM. The SCO could base this decision on a certain percentage of interactions being passed that map to the objective, a total score for a test or quiz, based on the objectives, compared against a mastery score, etc. This value indicates the overall success status for the SCO as determined by the SCO developer. If there is sequencing information applied to the learning activity associated with the SCO that relies on objective status, the SCO must ensure objective information is accurately sent to the LMS (SetValue()) prior to the SCO's learner session ending. Otherwise, the LMS will use the value "unknown" as the objective status for the appropriate objective (based on objective IDs) of the learning activity associated with the SCO when processing sequencing information. **Data Model Element Implementation Requirements:** • **Data Type:** state (passed, failed, unknown) • **Value Space:** The IEEE defines three state values. SCORM binds these state values to the following restricted vocabulary tokens: o "passed": The learner has passed the SCO [1]. Indicates that the necessary number of objectives was mastered or a necessary score was achieved. o "failed": The learner has failed the SCO [1]. Indicates that the learner did not master the necessary number of objectives or that a required score was not achieved. o "unknown": No assertion is made [1]. This indicates that no applicable assertion can be made that indicates the success status. • **Format:** The format of the data model value shall be one of the three restricted values listed above ("passed", "failed", "unknown"). **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by an LMS as read/write. • Normally the SCO will report its own *cmi.objectives.n.success_status* to the LMS, however there is no requirement in SCORM that mandates a SCO to set *cmi.objectives.n.success_status*. If the *cmi.objectives.n.success_status* is not reported by the SCO, then the LMS shall use the default of "unknown" as the value of the objectives success status. **Sequencing Impacts:** • If the SCO sets *cmi.objectives.n.success_status* for an objective of the SCO to "unknown", the *Objective Progress Status* for the objective (based on objective IDs) of the learning activity associated with the SCO shall be false. • If the SCO sets *cmi.objectives.n.success_status* for an objective of the SCO to "passed", the *Objective Progress Status* for the objective (based on objective IDs) of the learning activity associated with the SCO shall be true, and the *Objective Satisfied Status* for the objective (based on objective IDs) of the learning activity associated with the SCO shall be true. • If the SCO sets *cmi.objectives.n.success_status* for an objective

of the SCO to "failed", the *Objective Progress Status* for the objective (based on objective IDs) of the learning activity associated with the SCO shall be true, and the *Objective Satisfied Status* for the objective (based on objective IDs) of the learning activity associated with the SCO shall be false.

SCO Behavior Requirements:

- The data model element is required to be implemented by the LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.success_status* data model element.

API Implementation Requirements:

- **GetValue():** The LMS shall return the objectives associated *cmi.objectives.n.success_status* currently stored by the LMS for the learner and set the error code to `0` – No Error. The characterstring returned shall adhere to the requirements identified in the Data Model Element Implementation Requirements. Until the *cmi.objectives.n.success_status* has been set, the default value of the data model element shall be "unknown".
  - If the SCO attempts to retrieve the *cmi.objectives.n.success_status* and the record of data has been created but success_status data model element has not been set by the SCO nor determined by the LMS, then the LMS shall return the default value of "unknown" and set the error code to `0` – No Error.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a GetValue() request where the index (n) is a number larger than what the LMS is currently maintaining (e.g., the request indicated an n value of 5 when there are only 3 objectives in the array), then the LMS shall set the error code to `301` – General Get Failure and return an empty characterstring (""). Refer to Section 3.1.7.6: *SCORM Extension Error Conditions* for further recommendations on processing this request.
- **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.objectives.n.success_status* and the supplied value meets the requirements defined in the Data Model Element Implementation Requirements, then the LMS shall store the supplied value for *cmi.objectives.n.success_status*, set the error code to `0` – No Error and return "`true`".
  - If the SCO invokes a request to set the *cmi.objectives.n.success_status* and the value is not a member of the restricted vocabulary tokens described above, then the LMS shall set the API Error Code to `406` – Data Model Element Type Mismatch and return "`false`". The LMS shall not alter the state of the data model element based on the request.
  - The data model binding for collections is represented as packed arrays. If the SCO invokes a SetValue() request where the index (n) provided is a number that is greater than the current number of objectives being stored, then the LMS shall set the error code to `351` – General Set Failure and return "`false`". Refer to Section 3.1.7.6: *SCORM Extension Error Conditions*.
  - Since the *cmi.objectives.n.id* is required to be set first prior to any other objective information, if the SCO attempts to set *cmi.objectives.n.success_status* (prior

	to setting the identifier) then the LMS shall set the error code to `408` – Data Model Dependency Not Established and return "`false`". The LMS shall not alter the state of the data model element based on the request. **Example:** • GetValue("cmi.objectives.n.success_status") • SetValue("cmi.objectives.n.success_status","passed")
cmi.objectives.n.completion_status	The *cmi.objectives.n.completion_status* data model element indicates whether the learner has completed the associated objective [1]. How the SCO determines the *cmi.objectives.n.completion_status* for the objective is outside the scope of SCORM. For example, the SCO could base this decision on a number of interactions associated with the objective being completed. Since the determination of *cmi.objectives.n.completion_status* is controlled and managed by the SCO, the LMS cannot imply that the SCO is completed in any way. If no *cmi.objectives.n.completion_status* is reported by the SCO, then the LMS can only rely on the fact that the *cmi.objectives.n.completion_status* is "unknown". **Data Model Element Implementation Requirements:** • **Data Type:** state (completed, incomplete, not_attempted, unknown). • **Value Space:** The IEEE defines four state values. SCORM binds these state values to the following restricted vocabulary tokens: o "completed": The learner has experienced enough of the SCO for the associated objective to be considered complete [1]. How completion is determined is controlled and managed by the SCO. o "incomplete": The learner has not experienced enough of the SCO for the associated objective to be considered completed [1]. How completion is determined is controlled and managed by the SCO. o "not attempted": The learner has experienced the SCO, but the associated objective has not been attempted [1]. The SCO is responsible for determining whether or not the objective was attempted. o "unknown": No assertion is made [1]. This indicates that no applicable assertion can be made that indicates the completion status. • **Format:** The format of the data model value shall be one of the four restricted vocabulary tokens listed above ("completed", "incomplete", "not attempted", "unknown"). **LMS Behavior Requirements:** • This data model element is mandatory and shall be implemented by the LMS as read/write. • Normally the SCO will report its own objectives completion status to the LMS, however there is no requirement in SCORM that mandates a SCO to set *cmi.objectives.n.completion_status*. If the SCO does not set the *cmi.objectives.n.completion_status*, then the LMS shall treat the *cmi.objectives.n.completion_status* as "unknown". **SCO Behavior Requirements:** • The data model element is required to be implemented by an LMS as read/write. The SCO is permitted to retrieve and store the value of the *cmi.objectives.n.completion_status* data model element.