Sharable Content Object Reference Model (SCORM®)

Version 1.3 Working Draft 1

The SCORM Run-Time Environment

October 22, 2003

This page intentionally left blank.

Advanced Distributed Learning
Sharable Content Object Reference Model
(SCORM®)
Version 1.3
The SCORM Run-Time Environment

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

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

This page intentionally left blank.

Chief Technical Architect
Philip Dodds

Key ADL Technical Team Contributors to the SCORM Version 1.3:

William Capone
Clark Christensen
Jeff Falls
Dexter Fletcher
Matthew Handwork
Rob Harrity
Sue Herald
Alan Hoberney
Paul Jesukiewicz
Kirk Johnson
Mary Krauland
Jeff Krinock
Lori Morealli
Angelo Panar
Douglas Peterson
Jonathan Poltrack
Betsy Spigarelli
Schawn Thropp
Bryce Walat
Jerry West

Key ADL Community Contributors to the SCORM Version 1.3:

Mike Bednar
Bill Blackmon
Howard Fear
Lenny Greenberg
Peter Hope
Boyd Nielsen
Claude Ostyn
Nina Pasini
Dan Rehak
Tyde Richards
Roger St. Pierre
Brendon Towle

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/)
Erik Duval
Eddy Forte
Florence Haenny
Ken Warkentyne

Aviation Industry CBT Committee (AICC) (http://www.aicc.org/)
Jack Hyde
Bill McDonald
Anne Montgomery

Institute of Electrical and Electronics Engineers (IEEE) Learning Technology Standards Committee (LTSC) (http://ltsc.ieee.org/)
Erik Duval
Mike Fore
Wayne Hodgins
Tyde Richards
Robby Robson

IMS Global Learning Consortium, Inc. (http://www.imsglobal.org/)
Thor Anderson
Steve Griffin
Mark Norton
Ed Walker

(At Large)

Bob Alcorn
Tom Grobicki
Tom King
Chris Moffatt
Chantal Paquin
Mike Pettit
Tom Rhodes
Kenny Young

…and many others.

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

COPYRIGHT

DISTRIBUTION

Permission to distribute this document is granted under the following conditions:

1. The use of this document, its images and examples is for non-commercial, educational or informational purposes only.
2. The document, its images and examples are intact, complete and unmodified. The complete cover page, as well as the COPYRIGHT, DISTRIBUTION and REPRODUCTION sections are consequently included.

REPRODUCTION

Permission to reproduce this document completely or in part is granted under the following conditions:

1. The reproduction is for non-commercial, educational or informational purposes only.
2. Appropriate citation of the source document is used as follows:
    a. Source: Advanced Distributed Learning (ADL), Sharable Content Object Reference Model (SCORM®) Version 1.3 Working Draft, 2003.

For additional information or questions regarding copyright, distribution and reproduction, contact:

ADL Co-Laboratory
1901 North Beauregard Street
Alexandria, VA 22311
USA
(703) 575-2000

Table of Contents

This page intentionally left blank.

SECTION 1
The SCORM® Run-Time Environment

This page intentionally left blank.

1.1. The SCORM Run-Time Environment and the SCORM Bookshelf

As the SCORM has evolved, its editors have introduced changes designed to make researching and understanding the SCORM's critical concepts simpler for end users. The SCORM's organizational structure is often described as a set of books on a bookshelf (Figure 1.1a The SCORM Bookshelf). The books themselves contain application details and requirements of various standards and specifications. The coverage of each of the SCORM books is summarized as follows:

- **The SCORM Overview** book summarizes the history and background of the ADL Initiative and the SCORM, including coverage of the specifications and standards bodies from which the SCORM borrows. It also expands upon the description of the SCORM bookshelf, covering in detail how the various SCORM books are related to one another.
- **The SCORM Content Aggregation Model (CAM)** book describes the components used in a learning experience, how to package those components for exchange from system to system, how to describe those components to enable search and discovery and how to define sequencing rules for the components. The CAM promotes the consistent storage, labeling, packaging, exchange and discovery of content.
- **The SCORM Run-Time Environment (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.
- **The SCORM Sequencing and Navigation (SN)** book describes how SCORM-conformant content may be sequenced to the learner through a set of learner-initiated or system-initiated navigation events. The branching and flow of that content may be described by a predefined set of activities, typically defined at design time. The book also describes the requirements of how a SCORM-conformant LMS interprets the sequencing rules expressed by a content developer along with the set of learner-initiated or system-initiated navigation events and their effects on the run-time environment.

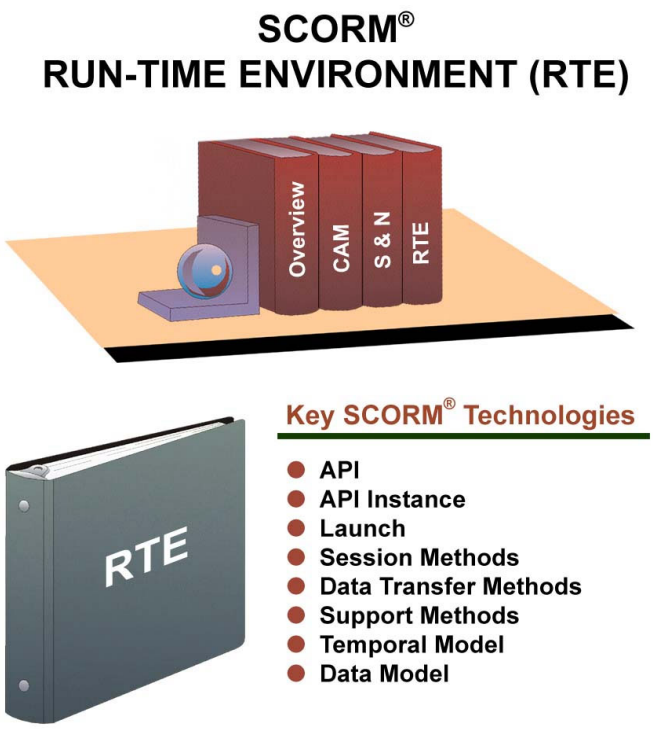

Figure 1.1a: The RTE as part of the SCORM bookshelf

1.1.1. What is Covered in the SCORM Run-Time Environment book?

Table 1.1.1a shows some specifics about which aspects of the SCORM are covered by which books. As described in the table column labeled Concepts Covered, the SCORM Run-Time Environment (RTE) book covers the essential LMS content-launch process and the communication relationship between LMSs and content.

Communication between content and LMSs facilitates use of SCORM Sequencing and Navigation in order to present content to users based on user choices and performance at run-time. This communication also enables LMSs to track learner progress and performance while content is presented to the learner. This book describes that communication process and the various elements that comprise it in detail.

General subjects discussed include the SCORM topics listed in the table column labeled Key Objects or SCORM Terminology Covered: Application Programming Interface (API), API Instance, Launch, Temporal Model, Session Methods, Data Transfer Methods, and Support Methods.

Section 1.2 Run-Time Environment Overview, provides a more detailed look at the topics covered in this document.

SCORM Book	Concepts Covered	Key SCORM Technology Covered	Areas of Overlap
Overview	High-level conceptual information	Incidental mention of numerous elements of SCORM terminology	Covers areas of the SCORM RTE, CAM and SN books at a high-level
Content Aggregation Model (CAM)	Assembling, labeling and packaging content	SCO, Asset, Content Aggregation, Package, Package Interchange File (PIF), Meta-data, Manifest, Sequencing Information, Navigation Information	SCOs and manifests. SCOs communicate with an LMS via the RTE. Manifests contain Sequencing and Navigation information
Run-Time Environment (RTE)	LMS Management of the Run-Time Environment which includes launch, content to LMS communication, tracking, data transfer, error handling	API, API Instance, Launch, Session Methods, Data Transfer Methods, Support Methods, Temporal Model, Run-Time Data Model	SCOs which are covered in the SCORM CAM book, are content objects which use the RTE
Sequencing and Navigation (SN)	Sequencing content, navigation	Activity Tree, Learning Activities, Sequencing Information, Navigation Information, Navigation Data Model	Sequencing and Navigation affects how content is assembled in a manifest.

Table 1.1.1a: SCORM Book Coverage

1.1.2. Using this book

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

Early portions of this book, Section 1 through Section 2, 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 SCORM Version 1.3 updates to the RTE. Section 2.1, Run-Time Environment Management, for instance, discusses how the new Sequencing and Navigation book impacts the SCORM RTE.

Section 3 Application Programming Interface, 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, focusing as they do on specific aspects of the SCORM, are intended to stand alone, there are areas of overlap or mutual coverage. For instance, while this book focuses primarily on communication between content and LMSs, it frequently refers to the types of content objects conducting that communication: Sharable Content Objects (SCOs). Their definition and the complete treatment of SCOs are found in the CAM book.

Similarly, while the SN book covers the details of SCORM sequencing and navigation processes, to include 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 Book

The SCORM Content Aggregation Model (CAM) book defines responsibilities and requirements for building content aggregations (e.g., course, lessons, modules, etc). The book contains information on creating content packages, applying meta-data 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 Content Aggregation Model book to the SCORM Run-Time Environment book.

Meta-data is "data about data". Simply put, SCORM]meta-data describes the different components of the SCORM content model (Content Aggregations, Activities, SCOs and Assets). Meta-data, a form of labeling, enhances search and discovery of these components. At this time, there are no defined relationships between the SCORM meta-data and the SCORM Run-Time Environment Model and SCORM meta-data has no impact on run-time behaviors or events For these reasons, meta-data is not discussed in detail in the SCORM RTE book. It is anticipated, as the SCORM evolves, that this relationship may change.

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 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 Run-Time Environment Model.

- Content object launch locations and launch parameters are also described as elements in the SCORM Content Package. These elements are essential to the launch and delivery of content objects. The SCORM Run-Time Environment book details these elements and their effects on launching content objects.
- Several elements in the SCORM Content Package affect initialization and management of a content object's run-time data model. The SCORM Run-Time Environment book details these elements and the required LMS behaviors.
- Other elements in the SCORM Content Package describe initial values for specific elements of a content object's run-time data model. The SCORM Run-Time Environment book details these 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 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 (see the SCORM Sequencing and Navigation 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 Content Aggregation Model book.

1.1.3.2 The SCORM Sequencing and Navigation Book

The SCORM Sequencing and Navigation book is based on the IMS Simple Sequencing (SS) Specification Version 1.0, which defines a method for representing the intended behavior of an authored learning experience such that any conformant LMS will sequence discrete learning activities in a consistent way.

The SCORM Sequencing and Navigation Model defines how IMS SS applies and is extended in a SCORM environment. It defines the required behaviors and functionality that SCORM conforming 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 Sequencing and Navigation book describes how learner-initiated 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 Run-Time Environment 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 Run-Time Environment 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 Run-Time Environment 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 the SCORM, content objects are either:

- Sharable Content Objects (SCOs), which communicate during run-time, or
- Assets, which do not communicate during run-time.

The SCORM Run-Time Environment 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 Run-Time Environment book is an essential piece of the SCORM Bookshelf (See the SCORM Version 1.3 Overview [10] for more details), however, it cannot stand-alone. The SCORM Run-Time Environment 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 Version 1.3 Sequencing and Navigation 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 Run-Time Environment 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 the SCORM and its concepts as a whole. It is strongly recommended that each book of the SCORM be read to more fully understand the purpose, details, relationships and advantages of all of the SCORM concepts.

Two goals of the SCORM are that content objects be 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 Run-Time Environment are Launch, Application Programming Interface (API) and Data Model.

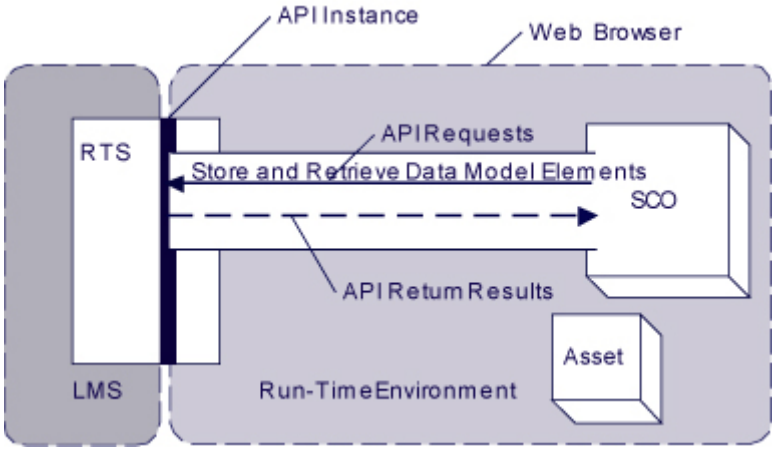

Figure 1.2a: SCORM Conceptual Run-Time Environment.

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 the 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 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 elements that both the LMS and SCO are expected to "know about." The LMS must maintain the state of SCO's data elements across learner sessions, and the SCO must utilize only these predefined data elements to ensure reuse across multiple systems.

This page intentionally left blank.

SECTION 2
Managing The Run-Time Environment

This page intentionally left blank.

2.1. Run-Time Environment Management

While the learner interacts with content objects (the learning experience), the LMS, evaluates learner performance and navigation requests (see the SCORM Version 1.3 Sequencing and Navigation book). When the LMS identifies an activity for delivery to the learner, the activity will have 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 (see the SCORM Content Aggregation Model).

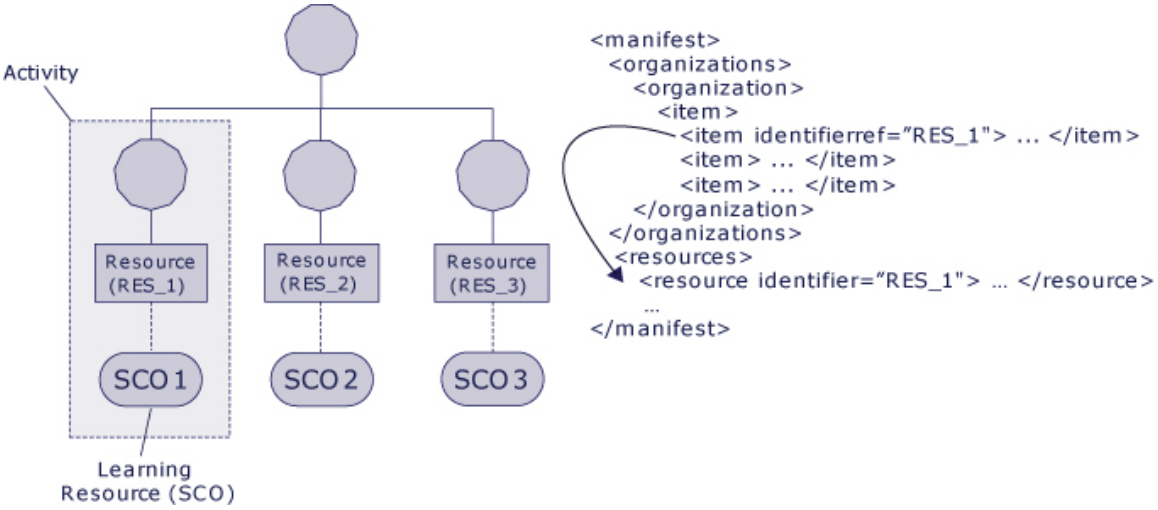

Figure 2.1a: Launching Content Objects

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. Note, 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 IEEE P1484.11.1 Draft 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. The attempt begins with the beginning of the first learner session and continues until the activity terminates. The state of the attempt 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].

These three terms are relevant when it comes to managing the Run-Time Environment for a SCO. For an Asset, the Run-Time Environment 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 (see the SCORM Version 1.3 Sequencing and Navigation 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 three terms and their relationships with one another for a specific SCO.

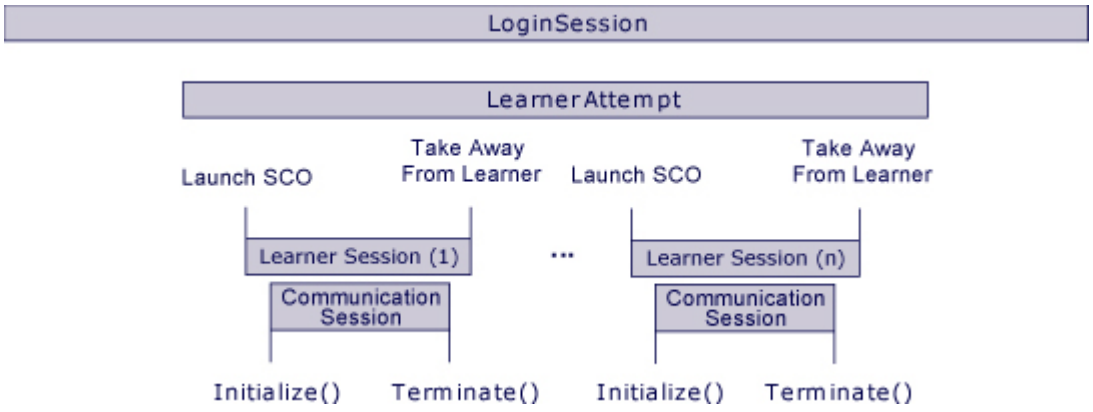

Figure 2.1.1a: Temporal Model Relationships for a Specific SCO

2.1.1.1 Managing Learner Attempt 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). The 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 the 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. The 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 the 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 pervious 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.

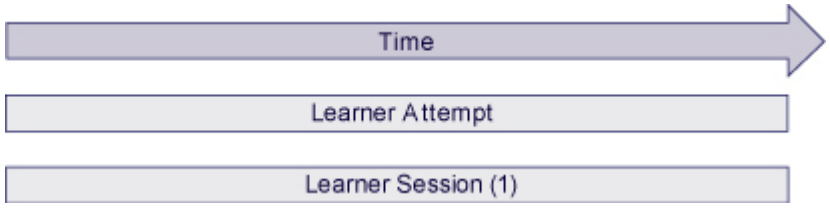

Figure 2.1.1.1a: Single Learner Attempt with one Learner Session

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.

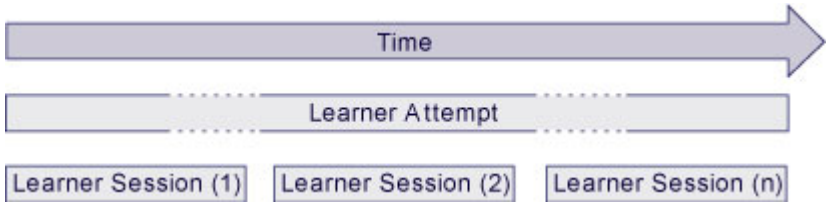

Figure 2.1.1.1b: Learner Attempt spread over several Learner Sessions

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

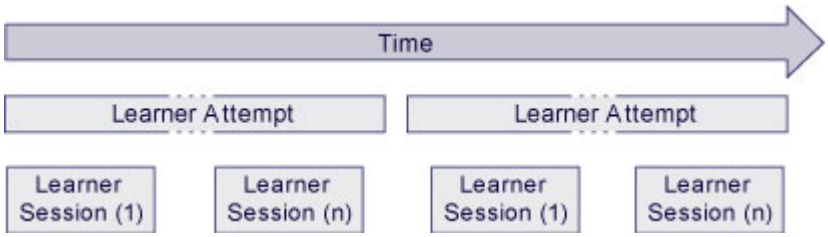

Figure2.1.1.1c: Successive Learner Attempts, each of which is spread over several Learner Sessions

In some cases, it may be necessary for a learning activity to have one and only one set of run-time data, spanning all learner attempts on the SCO associated with the learning activity. This requirement can be specified by declaring that the learning activity needs to persist its state (run-time data) between attempts (see the SCORM Content Aggregation Model book for more details on persisting state). If the learning activity has defined that the run-time data be persisted between learner attempts, then the LMS should only create and initialize a set of run-time data when the first learner session on the SCO associated with that activity begins. The persistence of state has no effect on an activity associated with an Asset.

2.1.2. Launching Content Objects

As described in the Content Aggregation Model (see the SCORM Content Aggregation book), the SCORM Content Model is made up of three components:

- Assets
- SCOs
- Content Aggregations

The SCORM Content Aggregation Model 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 (see the SCORM Version 1.3 Sequencing and Navigation 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 experienced 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 URL defined by the content object's launch location, defined in the content package (see Figure 2.1.2a), 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>
```

Figure 2.1.2a: Href used for Launching

The LMS is responsible to determine the appropriate fully qualified URL. The SCORM Content Aggregation Model defines, using the IMS Content Packaging Specification, 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

See the SCORM Content Aggregation Model 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 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 is only required to launch and track one SCO at a time (per learner) and that only one SCO is active at a time (from the standpoint of the LMS). In other words, the LMS launches and tracks one SCO at a time (per learner). The launched SCO can itself implement an API Instance for subordinate SCOs that it may launch and track. The LMS is not responsible for "knowing" of these subordinate SCOs. If this is the case, the "master" SCO that was launched by the LMS is responsible for all cleanup (e.g., closing of any windows that were opened to host the subordinate SCO) upon termination of itself.

The LMS must launch the SCO in a browser window that is a dependent window (i.e., "popup" 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 (see Launching Content Objects). Typically, content objects are taken away in response to a learner or system triggered navigation event (see the SCORM Version 1.3 Sequencing and Navigation 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 LMS's 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:

    a. 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.

    Note: 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, in order to present the appropriate implementation-defined user interface to the user.

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.

    Note: 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.

This page intentionally left blank.

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 the 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 AICC has submitted their work to the IEEE Learning Technology Standards Committee (LTSC) with the emphasis to standardize various pieces of the CMI Guidelines for Interoperability. The SCORM describes the IEEE P1484.11.2 Draft Standard for Learning Technology - ECMAScript Application Programming Interface for Content to Runtime Services Communication [2] as used in the 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 the 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 draft standard relates to a SCO (because in the SCORM, SCOs are the content objects that communicate to an LMS using the API).

The use of a common API fulfills many of the 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's provided API Instance communicates with the server-side component of the LMS is outside the scope of the SCORM. This back-channel communication can be implemented anyway the LMS vendor likes.

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

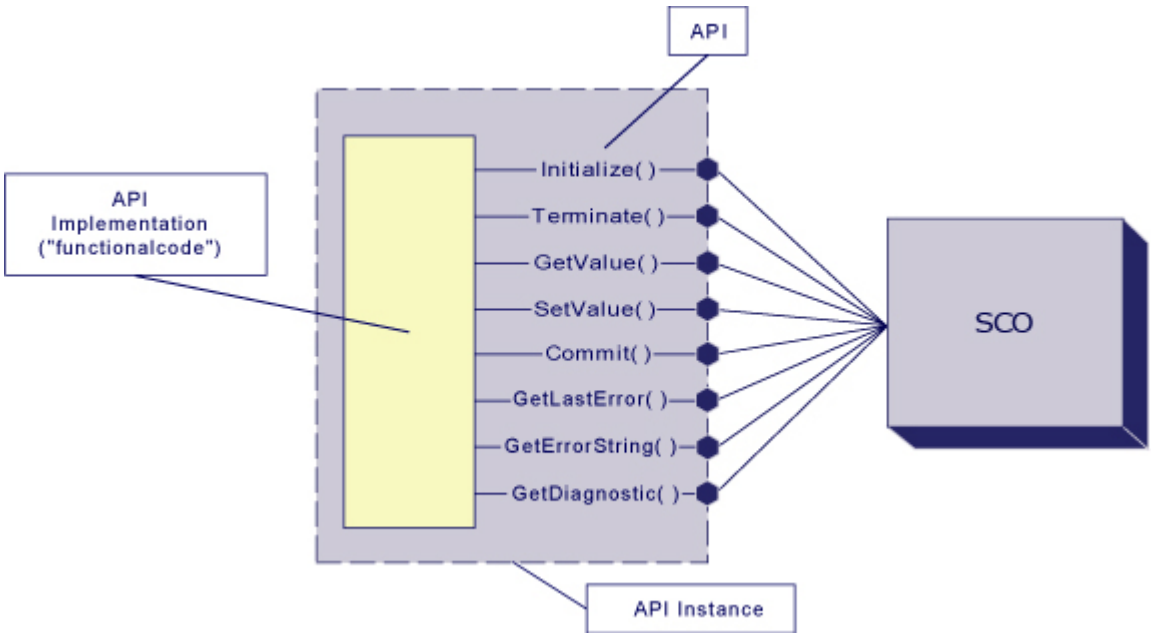

Figure 3.1.1a: API, API Instance, API Implementation

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 the internals of an API Implementation are implemented does not matter to SCO developers, if 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. The following table (Table:3.1.1b) defines these categories :

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.

Table: 3.1.1b: API Methods

3.1.2. API Methods and Syntax

The use of a common API fulfills many of the 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 provided that a SCO can find the API Instance in a consistent manner otherwise content developers would have to 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 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.
- All data passed as parameters shall be represented as a characterstring. Note that several examples throughout the 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 can then exchange (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 is 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. Note that 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

Session methods are used to initiate and terminate data communication between a single launched instance of a SCO 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 set off 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 `Terminiate()` 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 set off 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

The data-transfer methods are used by a SCO to direct the storage and retrieval of data that is 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 within a data model.

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 there was an error condition that 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 within a data model 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 set off 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.

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 set off 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 an 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 256 characters (including null terminator).
- The 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 256 characters (including null terminator). 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 256 characters (including null terminator). If the parameter is unknown by the LMS, an empty characterstring (“”) shall be returned.

3.1.6. Communication Session State Model

The IEEE draft 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` and `Terminated`. Note that 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.

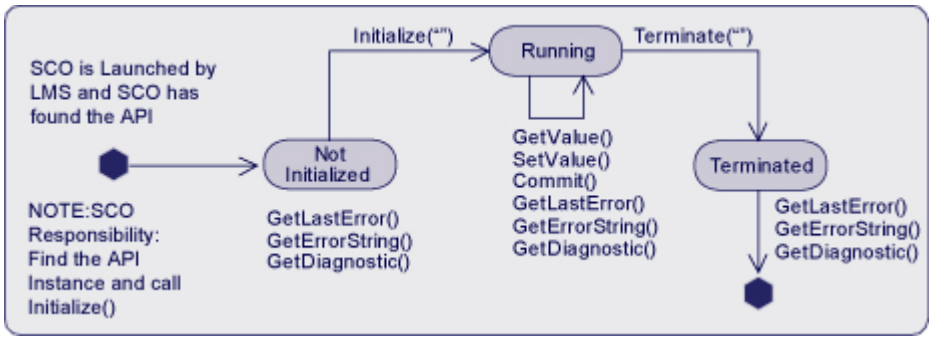

Figure 3.1.6a: Conceptual API Instance State Transitions

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.

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:

- `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 draft 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. The 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. The 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 draft has defined the following categories and numeric ranges for the various error codes:

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

Table 3.1.7a: Error Code Categories and Ranges

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 then 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 `Termination("")` method after a

previous `Termination("")` 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, long-term 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, long-term 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 draft 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 a run-time service. At this time, the IEEE draft 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 draft 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 IEEE draft standard as being used for situations that may arise during a `GetValue()` or `SetValue()` request, however, the SCORM does not define any situations for use of this error code during the processing of `GetValue()` requests. For `SetValue()` requests, some data model categories have certain requirements that certain 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 shall set the 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), the SCORM defines extension error conditions to cover error scenarios that may occur within a SCORM environment. There is no mechanism, defined by IEEE draft standard, to permit extension error codes to be set by an API Instance. In the following sections, the 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 Element Cannot Have Children Error Condition

The Element Cannot Have Children error condition indicates that a SCO attempted to retrieve a list of supported data model elements (children) for a data model element (see the SCORM Run-Time Environment Data Model for information regarding the data model elements). The Element Cannot Have Children error condition shall 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 cannot have any children (see 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 cannot have children.

3.1.7.6.2 Element Cannot Have Count Error Condition

The 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 an array (see the SCORM Run-Time Environment Data Model for elements that are arrays and array handling). The 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 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._count");`

For this example the API Instance shall set the error code to "301" and return an empty characterstring (""). The `learner_name` element does not have a count. The `learner_name` is not an 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 is not an array and does not have a count.

3.1.7.6.3 Collection Element Set Out of Order

The Collection Element 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 *4.2.1.3 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 (Refer to Section *4.2.2.2 Keywords*).

The Collection Element Set Out of Order error condition shall 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 t he array.

3.1.8. API General Application Rules

The following general API application rules shall be followed in order to achieve interoperability:

- The function names are all case sensitive, and must always be expressed exactly as shown and described in this document.
- The function parameters or arguments are case sensitive. All SCORM supported data model (SCORM Run-Time Environment Data Model and SCORM Navigation Data Model) parameters shall be represented in lower case.
- Each call to an API function, other than the Support methods, sets the error code.
- All parameters passed between a SCO and the API Instance are treated as ECMAScript strings and shall be compatible with the data types and formats described by the data models that use the API for communication.

3.2. LMS Responsibilities

The SCORM requires the LMS to provide an instance of the API as defined by the IEEE draft standard and the SCORM. The API Instance shields the SCO from the particular implementation details. The 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

The 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 corresponding the API Instance. To provide for an interoperable means to locate the API Instance, the LMSs 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.

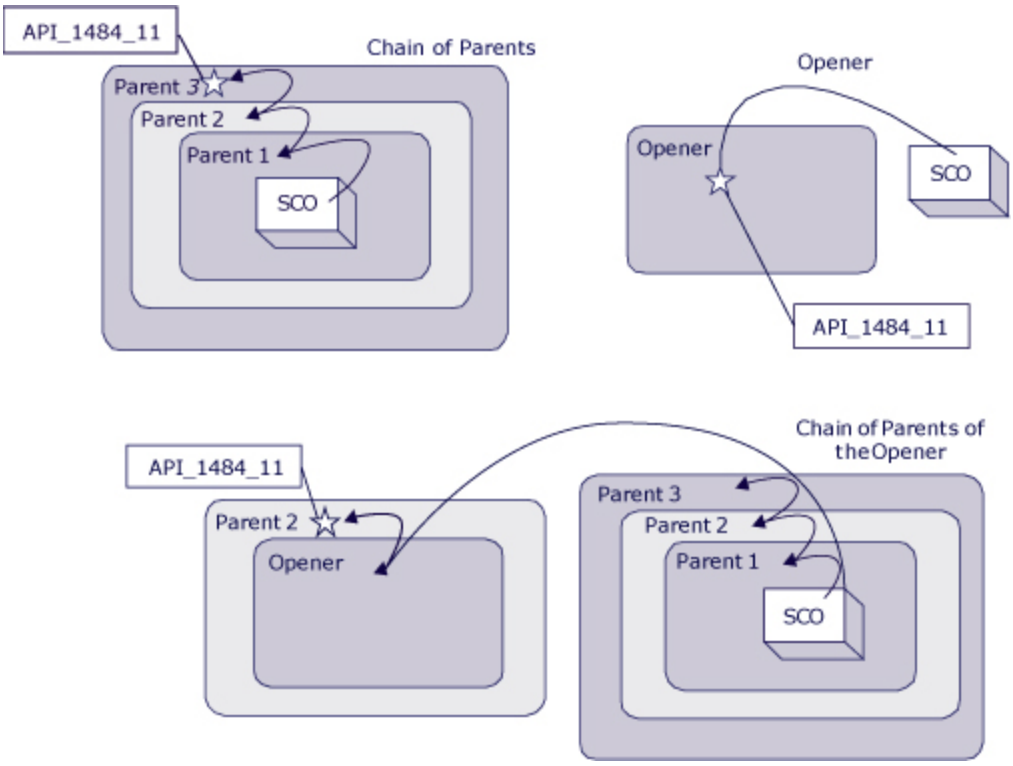

Figure 3.2.1a: Permitted location of API Implementation

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, the 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 the SCORM may introduce other communication protocols that support the fundamental requirements defined by the IEEE draft standard.

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 search. If the API Instance was 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.

3.3.1. Finding the API Instance

In order for a SCO to begin tracking a learners 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 consistently find the API Instance in a consistent manner from one LMS to another, the IEEE draft 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 draft 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`.

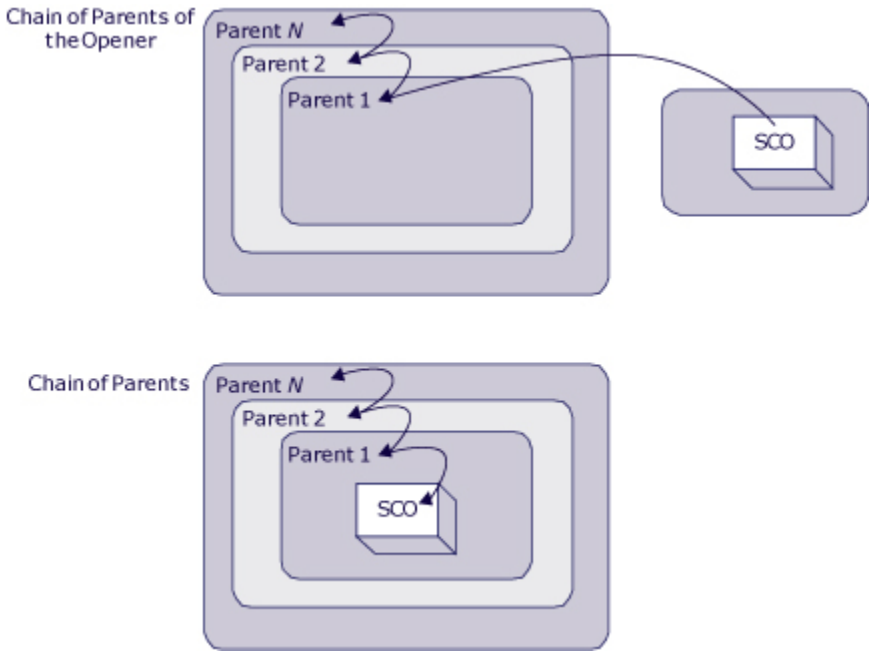

Figure 3.3.1a: Illustration of finding the API

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

The IEEE draft 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 = "";

function ScanForAPI(win)
{
   while ((win.API_1484_11 == null) && (win.parent != null)
           && (win.parent != win))
   {
      nFindAPITries++;
      if (nFindAPITries > maxTries)
      {
         alert("Error in finding API instance -- too deeply nested.");
         return null;
      }
      win = win.parent;
   }
   return win.API_1484_11;
}

function GetAPI()
```

```
{
   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;
      }
   }
}
```

Figure 3.3.1b: Example ECMAScript for finding the API Implementation (from IEEE draft standard)

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 intra-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 cache)

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 they are 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 encounter, SCO developers may want to adhere to the following recommendations:

- Invoke `Commit("")` whenever something significant happens that you want to be sure to have recorded, even if the unexpected happens 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 probably does no harm in a reasonable run-time service implementation, since there would be nothing new to commit during the `Terminate("")` function call.

- Do call `Terminate("")` prior to the SCO being unloaded. Keep track of whether you have successfully done it, so you don't invoke `Terminate("")` again; however, a properly built run-time service won't mind and will be able to handle the call. It should just ignore the second call and set the error status accordingly. If the client browser is Microsoft Internet Explorer, do 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 you have to rely on `onunload`. 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 things like 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 important data has been saved and committed before unload occurs, then they should be safe.

This page intentionally left blank.

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 sharable content objects (SCOs) can be tracked by different learning management system (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.

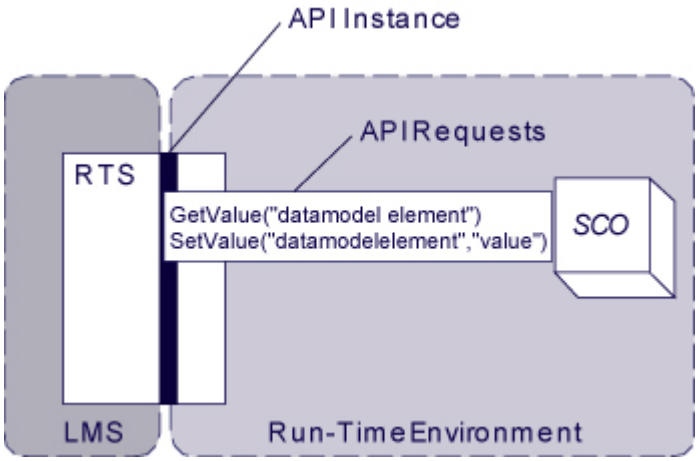

Figure 4.1.1a: Using the data model with the API

The SCORM Run-Time Environment Data Model is based on the P1484.11.1 Draft Standard for Learning Technology - Data Model for Content Object Communication [1] standard produced by the IEEE LTSC Computer Managed Instruction (CMI). P1484.11.1 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 the SCORM) to an LMS. This set of data includes but is not limited to information dealing with the learner, interactions the learner had with the SCO, objective information, mastery and completion statuses. 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 Version 1.3, the SCORM Run-Time Environment Data Model was based on the AICC CMI001 Guideline for Interoperability [7]. Since the release of the SCORM Version 1.2, AICC has submitted CMI001 to the IEEE for standardization. The SCORM Version 1.3 introduces the changes to the data model as defined by the IEEE P1484.11.1 Draft Standard for Learning Technology Data Model for Content Object Communication [1]. Since the IEEE draft standard purely defines elements and their data types, the 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 P1484.11.1 draft standard.

4.1.1. SCORM Run-Time Data Model Basics

4.1.1.1 Data Model Elements

To identify the data model, all of the names of the elements described in the SCORM start with "`cmi`". This signals to LMSs that these elements are part of the IEEE CMI working group's P1484.11.1 Data Model for Content Object Communication [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 the SCORM are required to be implemented and their behaviors supported by an LMS.

All data 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 browser and versions of browsers.

4.1.1.2 Data Model Effects on Sequencing

SCORM Sequencing (see the SCORM Sequencing and Navigation 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 only a limited ability to affect sequencing; only the fact that the Asset is launched is tracked. 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. The SCORM does not mandate a specific method or timing for how and when a SCO's run-time data is used for sequencing; just that the most recent information be used when required by a sequencing

evaluation. Specific LMS requirements for this data-mapping is provided in a per-element basis in the tables, which follow (for more information see the *Impacts on Sequencing* 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.

4.1.1.3 Handling Collections

All collections are defined in the SCORM as arrays. Several data model categories are represented as arrays of data:

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

These categories exist with the intention that SCOs may track multiple comments, objectives and/or interactions. Because of the current dot-notation binding of the SCORM Run-Time Environment Data Model, these categories contain an integer value representing the index (or location) in the list. The index is just that, an index. Indexes 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 categories.

The Objectives and Interactions categories contain an element (ID) 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. Index numbering starts at zero (0). All new collection elements shall be added sequentially. Indexes 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 can be used to determine the current number of 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 categories, besides `cmi.comments_from_lms`, can have their elements overwritten by the SCO. Overwriting or appending is a decision that is made by the SCO developer during the creation of the SCO.

Elements in a collection are referred to using a dot-number notation (represented by .n). For instance, the value of the status element in the first objective in a SCO would be referred to as `"cmi.objective.0.status", and` status element in the fourth objective would be referred to as `"cmi.objective.3.status"`.

4.1.1.4 Smallest Permitted Maximum Support

There is many cases when describing data model element requirements where there are smallest permitted maximums (SPMs) defined. The two cases are for character string lengths and the number of elements contained in collections (arrays, bags, etc.). The SPM is defined as the minimum number of entries (in a collection) or characters (length of character strings) 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. The 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). 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. The SCORM is silent on handling of collections that contain more than the SPM number of entries (e.g., an implementation is free not accept new entries). Implementations should be aware of the consequences if the number of entries exceeds the defined SPM.

4.1.1.5 Keywords

There is a set of keywords that are defined in the binding of the information model. These keywords allow for administrative duties for the data model elements. The three keywords defined for the binding are: `_version`, `_count` and `_children`.

`_version`: The `_version` keyword is used to determine the version of the data model supported by the LMS. The associated `_version` for the SCORM Version 1.3 shall be "`SCORMv1.3`". This value can be requested by a SCO to determine the version of the SCORM supported by the LMS. For example, the SCO could be authored to support multiple versions of the SCORM Data Model if so desired.

`_count`: The `_count` keyword is used to determine the number of elements currently in a collection. The count is the total number of elements in the collection, not the index number of the last position in the collection. The value can be requested by the SCO to determine the next index position that is free to be used for storing information.

`_children`: The `_children` keyword is used to determine all of the elements in a parent element that are supported by the LMS. Note that conforming LMSs will support all data model elements.

4.2. SCORM Run-Time Environment Data Model

4.2.1. Comments from Learner

There may be times where the 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 the 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 the 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 aggregation. These comments then may be used by the designer to evaluate the current design and structure of the content.

The `comments_from_learner` 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 `comments_from_learner` contains freeform text generated by the learner. The value of this data element is intended to provide feedback about the SCO from a specific learner. Using this data element for other purposes may adversely affect interoperability.

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

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 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the Comments From Learner parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements (order of elements is not important): "comment,location,date_time" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored. **LMS Behavior Requirements:** • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 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 _count keyword is used to describe the current number of learner comments that are currently being stored by the LMS. The total number of entries currently being persisted by the LMS shall be returned. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The characterstring representing the number of learner comments that the LMS is currently persisting. **LMS Behavior Requirements:** • This 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 element is implemented by an LMS as read-only. The SCO is only permitted to retrieve the value stored by the LMS. **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. ○ If no comments have been set by the SCO, 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_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 comment data element shall describe comments or annotations associated with a SCO [1]. The characterstring element represents a localized characterstring. **Data Element Implementation Requirements:** • **Data Type:** con_ob_com_localized_string (SPM: 4096) • **Value Space:** A characterstring (defined by ISO-10646-1) with localization information • **Format:** For elements with the con_ob_com_localized_string data type, information is needed to distinguish which language the characterstring is representing. In order to represent this information the following format is required for all con_ob_com_localized_string data types: The characterstring is required to have the following syntax (phrases in quotes are optional): "{lang=<*langcode*>}"<*actual characterstring*> Example: "{lang=en}The content presented an excellent point dealing with …" The *{lang=<langcode>}* shall represent the delimiter that indicates the language of the characterstring to follow. The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4]. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as read/write. • The value is supplied by the learner. The LMS shall not make any assumption on an initial value for this 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 element is implemented by an LMS as read/write. If the SCO desires the use of this element, the SCO can retrieve and store this data model element. • During a SetValue() request, the SCO should be aware the special delimiter is optional. If the delimiter is not provided as 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 special 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* element and set the error code to "0" – No error. The characterstring returned shall adhere to the requirements identified in the Data 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 "403" – Data model element value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.comments_from_learner.n.comment* and 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". o If the supplied value of the SetValue() does not meet the requirements of the Data 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.comments_from_learner.n._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:** • If the element is implemented, the SCO has the responsibility to make sure *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 (4096) that shall be implemented by the LMS. **Example:** • GetValue("cmi.comments_from_learner.0.comment") • SetValue("cmi.comments_from_learner.0.comment","Some comments about the SCO")
cmi.comments_from_learner.n.location	The location data element indicates the point in the SCO to which the comment applies. This data element is implementation-defined by each SCO. If no location is provided, the comment is applicable to the entire SCO (as a whole) [1]. **Data Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 255) • **Value Space:** ISO-10646-1 • **Format:** The format of this element is defined and controlled by the SCO. **LMS Behavior Requirements:**

	• The 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 comment has been set by the SCO, then the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, the SCO can retrieve and store the location of the comment. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_learner.n.location* element and set the error code to "0" – No error. The characterstring returned shall adhere to the requirements identified in the Data 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 "403" – Data model value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.comments_from_learner.n.location* and 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.location* 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 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.comments_from_learner.n._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 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 (255) 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.date_time	The date_time data element indicates the date and time at which the comment was made or changed. Implementation shall support, minimally, time periods in the range of January 1, 1970 through January 1, 2038 [1]. **Data Element Implementation Requirements:** • **Data Type:** time(second,10,2) • **Value Space:** The data type denotes that the value for time is a number expressed as a read data type with values that are accurate to one hundredth of a second. The number of seconds in the time value is the number of seconds since 00:00 on January 1, 1970 [1]. • **Format:** YYYY[-MM[-DD[Thh[:mm[:ss[.s[TZD]]]]]]] where YYYY: A four-digit year (>-0001) 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) TZD: Time zone designator ("Z" for UTC or +hh:mm or –hh:mm) 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: • "2003" • "2003-07-25T03:00:00" • "2003T14:35:34.3" **LMS Behavior Requirements:** • The 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 comment being set the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires to use this element, the SCO can retrieve and store the date_time (timestamp) of the comment. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_learner.n.date_time* element and set the error code to "0" – No error. The characterstring returned shall adhere to the requirements identified in the Data 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 "403" – Data model value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.comments_from_learner.n.date_time* and 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.date_time* 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 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.comments_from_learner.n._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.comments_from_learner.0.data_time") • SetValue("cmi.comments_from_learner.0.date_time","2003T03:00:00")

4.2.2. Comments from LMS

The `comments_from_lms` data model element contain 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. The 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 the 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 `comments_from_lms` data element contains comments and annotations intended to be made available to the learner.

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

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 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the *comments_from_lms* parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements (order of elements is not important): "comment,location,date_time" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored. **LMS Behavior Requirements:** • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:**

	• **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 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 *_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 persisted by the LMS shall be returned. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The characterstring representing the number of comments that the LMS is currently persisting. **LMS Behavior Requirements:** • This 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 element is implemented by an LMS as read-only. The SCO is only permitted to retrieve the value stored by the LMS. **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. ○ 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 comment data element shall describe comments or annotations associated with a SCO [1]. The characterstring element represents the localized characterstring. **Data Element Implementation Requirements:** • **Data Type:** con_ob_com_localized_string (SPM: 4096) • **Value Space:** A characterstring (defined by ISO-10646-1) with localization information • **Format:** For elements with the

	con_ob_com_localized_string data type, information is needed to distinguish which language the characterstring is representing. In order to represent this information the following format is required for all con_ob_com_localized_string data types: The characterstring is required to have the following syntax (phrases in quotes are optional): "{lang=<*langcode*>}"<*actual characterstring*> Example: "{lang=en}The content presented an excellent point dealing with …" The *{lang=<langcode>}* shall represent the delimiter that indicates the language of the characterstring to follow. The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4]. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as read-only. • How this element is initialized is outside the scope of the 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. **SCO Behavior Requirements:** • The element is implemented by an LMS as read-only. If the SCO desires the use of this element, the SCO can only retrieve this data model element. • During a GetValue() request, the SCO should be aware that the special 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.* element and set the error code to "0" – No error. The characterstring returned shall adhere to the requirements identified in the Data 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 "403" – Data model element value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.comments_from_lms.n.comment* and 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():** 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 element based on the request. **Example:** • GetValue(“cmi.comments_from_lms.0.comment”)
cmi.comments_from_lms.n.location	The location data element indicates the point in the SCO to which the comment applies. This data element is implementation-defined by each SCO. If no location is provided, the comment is applicable to the entire SCO (as a whole) [1]. **Data Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 255) • **Value Space:** ISO-10646-1 • **Format:** The format of this element is defined and controlled by the SCO developer. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read-only. • The value is controlled and supplied by the SCO. If a GetValue() request is made before the comment being set the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The element is implemented by an LMS as read-only. If the SCO desires the use of this element, the SCO can retrieve the location of the comment. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_lms.n.location* element and set the error code to “0” – No error. The characterstring returned shall adhere to the requirements identified in the Data 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 “403” – Data model element value not initialized and return an empty characterstring (“”). o If the SCO attempts to retrieve the *cmi.comments_from_lms.n.location* and 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():** 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 element based on the request. **Example:** • GetValue("cmi.comments_from_lms.0.location")
cmi.comments_from_lms.n.date_time	The date_time data element indicates the date and time at which the comment was made or changed. Implementation shall support, minimally, time periods in the range of January 1, 1970 through January 1, 2038 [1]. **Data Element Implementation Requirements:** • **Data Type:** time(second,10,2) • **Value Space:** The data type denotes that the value for time is a number expressed as a read data type with values that are accurate to one hundredth of a second. The number of seconds in the time value is the number of seconds since 00:00 on January 1, 1970 [1]. • **Format:** YYYY[-MM[-DD[Thh[:mm[:ss[.s[TZD]]]]]]] Where: **YYYY**: A four-digit year (>-0001) **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) **TZD**: Time zone designator ("Z" for UTC or +hh:mm or –hh:mm) 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: • "2003" • "2003-07-25T03:00:00" • "2003T14:35:34.3" **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read-only. • If a GetValue() request is made prior to the comment being set the LMS shall behave according to the API Implementation Requirements below. **SCO Behavior Requirements:** • The element is implemented by an LMS as read-only. If the SCO desires the use of this element, the SCO can retrieve the date_time (timestamp) of the comment. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.comments_from_lms.n.date_time* element and set the error code to "`0`" – No error. The characterstring returned shall adhere to the requirements identified in the Data 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 "403" – Data model element value not initialized and return an empty characterstring ("").
	○ If the SCO attempts to retrieve the *cmi.comments_from_lms.n.date_time* and 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():** If the SCO invokes a SetValue() request to set the *cmi.comments_from_lms.n.date_time,* 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 element based on the request.
	Example:
	• GetValue("cmi.comments_from_lms.0.date_time")

4.2.3. Completion Status

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

Dot-Notation Binding	Details
cmi.completion_status	**Data Element Implementation Requirements:** • **Data Type:** state (completed, incomplete, not_attempted, unknown) • **Value Space:** The IEEE draft defines four state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "completed": The learner has experienced enough of the SCO to consider it complete [1]. How completion is determined is controlled and managed by the SCO. o "incomplete": The learner has not experienced enough of the SCO to consider it 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]. With the introduction of IMS Simple Sequencing, the LMS is now responsible for attempt tracking and is responsible for determining an attempt. Since this is the case, the SCO shall not set the completion_status to "not attempted". This value is not used in SCORM Version 1.3. 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 element is mandatory and shall be implemented by an LMS as read/write. • Normally the SCO will report its own completion_status to the LMS, however there is no requirement in the SCORM that mandates a SCO to set completion_status. If the SCO does not set the completion_status, then the LMS shall treat the completion_status as "unknown". • The default completion_status, if not set by the SCO, shall be "unknown". • Impacts on Sequencing, (1) If the SCO or LMS (through the above process) sets completion_status, of the SCO to "unknown", the Attempt Progress Status for the learning activity associated with the SCO shall be false. (2) If the SCO or LMS (through the above process) sets 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. (3) If the SCO or LMS (through the above process) sets 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. **SCO Behavior Requirements:** • The element is implemented by the LMS as read/write. The SCO, if desired, shall be permitted to retrieve (GetValue()) and/or store (

	SetValue()) the data model element • The SCO shall not set completion_status to "not attempted". • The SCO should be aware that setting the completion_status will affect the learning activity associated with the SCO, therefore possibly affecting sequencing. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated completion_status currently maintained 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 Element Implementation Requirements. o Until some determination factor is present, the default value of the *cmi.completion_status* is "unknown". • **SetValue():** If the SCO invokes a request to set the 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 element based on the request. **Additional Information:** • Since the determination of completion_status is controlled and managed by the SCO, the LMS cannot imply that the SCO is completed in any way. If no completion_status is reported by the SCO, then the LMS can only rely on the fact that the completion_status is "unknown". • 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. **Example:** • GetValue("cmi.completion_status") • SetValue("cmi.completion_status","incomplete")

4.2.4. Credit

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

Dot-Notation Binding	Details
cmi.credit	**Data Element Implementation Requirements:** • **Data Type:** state (credit, no_credit) • **Value Space:** The IEEE draft defines two state values. The 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 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 effects the determination of success status (Refer to Section *4.2.14.1 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 effect the determination of success status or score shall not be interpreted by the LMS and shall not effect 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 element is mandatory and shall be implemented by an LMS as read only. • The 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 the 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 element is implemented by the LMS as read only. The SCO, if desired, shall be permitted to retrieve this data. 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 data element values to the LMS (via a SetValue() call). • The SCO is not permitted to request to set this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.credit* currently maintained 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 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 maintained 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 element based on the request. **Additional Information:**

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

4.2.5. Entry

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

Dot-Notation Binding	Details
cmi.entry	**Data Element Implementation Requirements:** • **Data Type:** state (ab_initio, resume, _nil_) [1] • **Value Space:** The IEEE draft defines three state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "ab-initio": Indicates that the LMS asserts that the learner has never accessed the SCO [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, and (2) upon exiting, the *cmi.exit* data element had the value of "suspend" [1]. o "" (empty characterstring): Indicates all other conditions. The LMS has no knowledge of previous access or does not wish to indicate a specific entry condition [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 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"), 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), 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 element is implemented by the LMS as read only. The SCO, if desired, shall be permitted to retrieve this data. What the SCO does with this data is totally up to the discretion of the SCO. • The SCO is not permitted to request to set this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.entry* value currently maintained 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 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 element based on the request. **Additional Information:** • 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 suspend_data available. An entry value of “suspend” indicates that the SCO is accessing run-time data for the current learner attempt as set from the previous learner session on the SCO. An entry value of “” (empty string) indicates that the state of the SCO has been persisted to span learner attempts. **Example:** • GetValue(“cmi.entry”)

4.2.6. Exit

The `exit` data 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.

Dot-Notation Binding	Details
cmi.exit	**Data Element Implementation Requirements:** • **Data Type:** state (timeout, suspend, logout, _nil_) • **Value Space:** The IEEE draft defines four state values. The 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]. o "" empty characterstring: The SCO exited normally [1]. The default value. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as write-only. • There is no initial value for this data model element. This value is completely controlled by the SCO. The SCO is responsible for setting this value. 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. • Impacts on Sequencing: (1) If the SCO sets 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. (2) If the SCO sets exit to "suspend", the LMS shall set the Activity is Suspended value of the learning activity associated with the SCO to true. (3) If the SCO sets exit to "logout", the LMS shall process an "Suspend All" navigation request when the SCO is taken away, instead of any pending (from the learner or LMS) navigation request. **SCO Behavior Requirements:** • The element is implemented by an LMS as write-only. If the SCO desires the use of this element, then the SCO can only make requests to store a value for this 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 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. o 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 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 element shall be implemented as read-only, an LMS is responsible for managing this value (the SCO does not directly effect 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 differently. 1. If the SCO set the *cmi.exit* to "time-out", it is assumed that the learner attempt is ending. The LMS shall set the *cmi.entry* to "ab-initio" on the next learner attempt on the timed-out SCO. In addition, the LMS shall end the attempt on the content aggregation 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 it at the point of exit later. 3. If the SCO set the *cmi.exit* to "logout", then the LMS shall set the *cmi.entry* to "resume" In addition, the LMS shall suspend the attempt on the content aggregation for the learner. This value does not indicate to log the learner out of the LMS and force the learner to reauthenticate in any manner. This behavior shall be exhibited when a Terminate("") request is received from the SCO that set the *cmi.exit* to "logout" or the learner navigates away. 4. If the SCO set the *cmi.exit* to "" (empty characterstring), then the LMS shall set the *cmi.entry* to "" (empty characterstring). 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. Note: 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").

4.2.7. Interactions

The interactions data model element defines a set 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 4.2.7a. As defined by the IEEE draft standard an LMS is required to support (i.e., store) at least 100 sets of interaction data [1]. An LMS can elect to provide support for more than 100, however the requirement is to support at least 100 sets of interaction data.

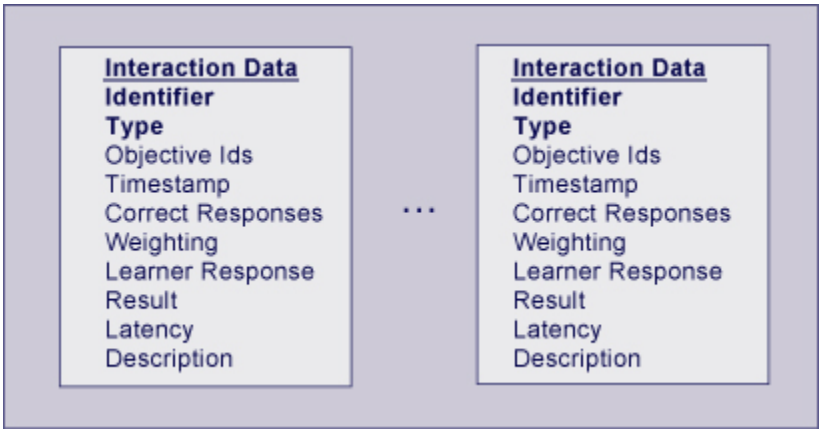

Figure 4.2.7a: Interactions and Interaction Data

There are two important pieces of data that are required to be in the set of interaction data, an identifier (*cmi.interactions.n.id*) and the type of interaction (*cmi.interactions.n.type*). These two pieces of information are what set off the interaction data from other interaction data found in the set of interactions (considered the dependencies of the interaction data). The identifier uniquely identifies one interaction from another. The type uniquely identifies the type of interaction (true-false, matching, etc.).

The Interaction data model element can be use 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 learners 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.

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 set of 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 100. SCO developers need to understand the smallest permitted maximum of 100 indicates that the LMS is only required to support 100 sets of interaction data.

As with any data element stored in arrays the index position (n) is not what sets the uniqueness of the data being stored. The identifier of the interaction should be used to uniquely identify each record. 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). It is recommended that SCOs be built, if using the interaction data 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 his highly recommended that the SCO be built to search throughout the interaction data looking for given identifiers, prior to updating (status scheme described above) 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.

Dot-Notation Binding	Details
cmi.interactions	Represents a set of learner responses that can be passed from a SCO to the LMS [1].
cmi.interactions._children	The *cmi.interactions._children* data model element represents a listing of supported data model elements. This 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the Interaction parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements (order of elements is not important): "id, type,objectives, timestamp, weighting, learner_response, result, latency, description" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored.

	LMS Behavior Requirements: • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 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 _count keyword is used to describe the current number of interactions being stored by the LMS. The total number of entries currently being persisted by the LMS shall be returned. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The characterstring representing the number of interactions that the LMS is currently persisting. **LMS Behavior Requirements:** • This 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 element is implemented by an LMS as read-only. The SCO is only permitted to retrieve the value stored by the LMS. **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 is 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.initeractions._count")

cmi.interactions.n.id	The *id* data element is a label for the interaction. This label shall be unique at least within the scope of the SCO [1]. **Data Element Implementation Requirements:** • **Data Type:** con_ob_com_object_identifier • **Value Space:** A characterstring (SPM: 4096) that represents a valid URI as per RFC 2396 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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 **LMS Behavior Requirements:** • This 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 element is implemented by the LMS as read/write. The SCO is permitted to retrieve and store the interactions identifier. • 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 for each interaction needing tracked by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated interactions identifier currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to retrieve the interactions identifier and the index (n) is not available (i.e., has not been set by the SCO previously), then the LMS shall return an empty characterstring ("") and set the error code to "403" Data model element value not initialized. • **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 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 element based on the request . o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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.id") • SetValue("cmi.interactions.0.id","obj1") **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 interaction's identifier is a required field that shall be set by the SCO if interaction data is will be requested to be tracked by the SCO.
cmi.interactions.n.type	The *type* data element indicates which category of interaction is recorded and how the interaction response should be interpreted [1]. **Data 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 draft defines ten state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "true-false": The interaction has two possible responses [1]. The 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 predefined 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 a string of characters [1]. o "long-fill-in": The interaction requires the learner to supply a response in the form of a long string of characters. 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]. o "likert": The interaction asks the learner to select from a discrete set of choices on a scale [1]. o "numeric": The interaction requires a numeric response from the learner. The response is a simple number with an optional decimal point [1]. o "other": Any other type of interaction not defined by the IEEE draft 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 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 element is implemented by the LMS as read/write. The SCO is permitted to retrieve and store the interactions type. • 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 needing tracked by the SCO. The *cmi.interactions.n.type* shall be set prior to any other interactions data (other than the *cmi.interactions.n.id*). If the *cmi.interactions.n.type* is not set prior to any other interaction data (other than *cmi.interaction.n.id*), then a data model dependency is not being met. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated interactions type currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to retrieve the interactions type and the index (n) is not available (i.e., has not been set by the SCO previously), then the LMS shall return an empty characterstring ("") and set the error code to "403" Data model element value not initialized. • **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 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 element based on the request . o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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.type") • SetValue("cmi.interactions.0.type","true-false")
cmi.interactions.n.objectives	The *cmi.interactions.n.objectives* indicates those objectives that are related to the interactions data. The data model element stores objective identifiers that the interaction data relates to.
cmi.interactions.n.objectives._count	The _count keyword is used to describe the current number of objectives (i.e., objective identifiers) being stored by the LMS. The total number of entries currently being persisted by the LMS shall be returned.

	Data Element Implementation Requirements: • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The characterstring representing the number of objective identifiers that the LMS is currently persisting. **LMS Behavior Requirements:** • This 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 element is implemented by an LMS as read-only. The SCO is only permitted to retrieve the value stored by the LMS. **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.n.id	The *cmi.interactions.n.objectives.n.id* data 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.n.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 Element Implementation Requirements:** • **Data Type:** con_ob_com_object_identifier • **Value Space:** A characterstring (SPM: 4096) that represents a valid URI as per RFC 2396 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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:objective-id-0001

LMS Behavior Requirements:

- This 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 element is implemented by the LMS as read/write. The SCO is permitted to retrieve and store the interactions identifier.
- 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 set the *cmi.interactions.n.id* and *cmi.interactions.n.type*.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated objective identifier currently maintained 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 Element Implementation Requirements.
    - If the SCO invokes a request to retrieve the objective identifier and the index (n) is not available (i.e., has not been set by the SCO previously), then the LMS shall return an empty characterstring ("") and set the error code to "`403`" Data model element value not initialized.
- **SetValue():** The LMS shall set the *cmi.interactions.n.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 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 element based on the request .
    - The data model binding for collections are 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 (for the interaction) being persisted (can be determined by requesting the *cmi.interactions.n.objectives._count*) or the current number of interactions being persisted (can be determined by requesting the *cmi.interactions.n._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* and *cmi.interactions.n.type*

	are not set prior to the request to set the *cmi.interactions.n.objectives.n.id*, then the LMS shall set the error code to "`408`" – Data model dependency not met, return "`false`" and not change the current state of the element. **Example:** • GetValue("cmi.interactions.0.objectives.0.id") • SetValue("cmi.interactions.0.id"," urn:ADL:objective-id-0001")
cmi.interactions.n.timestamp	The *cmi.interactions.n.timestamp* data element is the time at which the interaction was first available for response to the learner [1]. The value of the timestamp is actual wall-clock time. If several interactions are presented at the same time, they have he same timestamp value. If an interaction was never available for response, such 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 Element Implementation Requirements:** • **Data Type:** time(second,10,2) • **Value Space:** The data type denotes that the value for time is a number expressed as a read data type with values that are accurate to one hundredth of a second. The number of seconds in the time value is the number of seconds since 00:00 on January 1, 1970 [1]. • **Format:** YYYY[-MM[-DD[Thh[:mm[:ss[.s[TZD]]]]]]] where YYYY: A four-digit year (>-0001) 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) TZD: Time zone designator ("Z" for UTC or +hh:mm or –hh:mm) 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: • "2003" • "2003-07-25T03:00:00" • "2003T14:35:34.3" **LMS Behavior Requirements:** • This 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 element is implemented by the LMS as read/write. The SCO is permitted to retrieve and store the interactions timestamp.

	• 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 set the *cmi.interactions.n.id* and *cmi.interactions.n.type*. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated interaction timestamp currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to retrieve the interaction's timestamp and the index (n) is not available (i.e., has not been set by the SCO previously), then the LMS shall return an empty characterstring ("") and set the error code to "403" Data model element value not initialized. • **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". o If the supplied value of the SetValue() does not meet the requirements of the Data 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 element based on the request . o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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.timestamp*, then the LMS shall set the error code to "408" – Data model dependency not met, return "false" and not change the current state of the element. **Example:** • GetValue("cmi.interactions.0.timestamp") • SetValue("cmi.interactions.0.timestamp"," 2003-07-25T03:00:00")
cmi.interactions.n.correct_responses	The *cmi.interactions.n.correct_responses* data element indicates the correct response to the interaction (as determined by the SCO). The data element shall have one of ten possible variants. Each variant depends on the type (i.e., *cmi.interactions.n.type*) of interaction.
cmi.interactions.n.correct_responses._count	The _count keyword is used to describe the current number of correct responses being stored by the LMS. The total number of entries currently being persisted by the LMS shall be returned. **Data Element Implementation Requirements:** • **Data Type:** characterstring

	• **Value Space:** ISO-10646-1 • **Format:** The characterstring representing the number of correct responses that the LMS is currently persisting. **LMS Behavior Requirements:** • This 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 element is implemented by an LMS as read-only. The SCO is only permitted to retrieve the value stored by the LMS. **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 is no correct responses for the interaction data currently being stored. If no correct responses 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.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 element defines the correct responses to the interaction. The format of the pattern value depends on the type (*cmi.interactions.n.type*) of interaction. The *cmi.interactions.n.correct_responses.n.pattern* is an array of correct responses. Depending on the type (*cmi.interactions.n.type*) of interaction, the amount of correct responses required to be supported varies. Refer to Section *4.2.11.1 Correct Responses Data Element Specifics* for more details on each type. **Data Element Implementation Requirements:** • The data type for the pattern varies depending on the type (*cmi.interactions.n.type*) of interaction. *Section 4.2.11.1 Correct Responses Data Element Specifics* defines the data element implementation requirements and format for each type of interaction. **LMS Behavior Requirements:** • The 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 (See API Implementation Requirements). **SCO Behavior Requirements:**

	• The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store the correct responses. The data element requirements and format for the correct responses are described in detail in *Section 4.2.11.1 Correct Responses Data Element Specifics*. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.correct_responses.n.pattern* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.interactions.n.correct_responses.n.pattern* prior to the value being 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.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 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 4.2.11.1 Correct Responses Data 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._count*) or correct response patterns being persisted (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*. 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.correct_responses.n.pattern*, then the LMS shall set the error code to "408" – Data model dependency not met, return "false" and not change the current state of the element. **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 element is the weight used by the SCO to compute a total score. The interaction weights typically are used to explain the effect of an interaction on a total 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 the SCORM.

Data Element Implementation Requirements:

- **Data Type:** real (10,7)
- **Value Space:** A real number with values that is accurate to seven significant decimal figures.

LMS Behavior Requirements:

- The 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 (See API Implementation Requirements).

SCO Behavior Requirements:

- The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store the weighting value for an interaction.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated *cmi.interactions.n.weighting* currently maintained 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 Element Implementation Requirements.
  - If the SCO invokes a request to get the *cmi.interactions.n.weighting* prior to the value being 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`".
  - 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 element based on the request.
  - The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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* and *cmi.interactions.n.type* are not 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 met, return "`false`" and not change the current state of the 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 element consists of the data generated by a learner when responding to an interaction or a description of the learner's action [1]. The learners 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 *4.2.11.2 Learner Response Data Element Specifics* for more details on each type. **Data Element Implementation Requirements:** • The data type for the value varies depending on the type (*cmi.interactions.n.type*) of interaction. *Section 4.2.11.2 Learner Response Data Element Specifics* defines the data element implementation requirements and format for each type of interaction. **LMS Behavior Requirements:** • The 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store the learner response. The data element requirements and format for the learner response are described in detail in *Section 4.2.11.2 Learner Response Data Element Specifics*. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.learner_response* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.interactions.n.learner_response* prior to the value being 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 4.2.11.2 Learner Response Data 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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 met, return "`false`" and not change the current state of the 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 element is a judgment of the correctness of the learner response [1]. **Data Element Implementation Requirements:** • **Data Type:** state (correct, wrong, unanticipated, neutral, real (10,7)) • **Value Space:** The IEEE draft defines five state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "correct": The learner response was correct [1]. o "wrong": The learner response was incorrect [1]. o "unanticipated": The learner response was not expected by the SCO [1]. o "neutral": The learner response was neither correct nor incorrect [1]. o real (10,7): A real number • **Format:** The format of the data model value shall be one of the restricted tokens listed above ("correct", "wrong", "unanticipated", "neutral") or a real number with values that is accurate to seven significant decimal figures real. **LMS Behavior Requirements:** • The 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store the result of the learner response for an interaction. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.result* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the

	cmi.interactions.n.result prior to the value being 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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.result*, then the LMS shall set the error code to "408" – Data model dependency not met, return "false" and not change the current state of the 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 first available for response to the learner 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 Element Implementation Requirements:** • **Data Type:** timeinterval (seconds,10,2) - a time interval with resolution to 0.01 seconds • **Format:** P[yY][mM][dD][T[hH][mM][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) The character literals designators "P","Y","M","D","T","H","M","S" shall appear if the corresponding non-zero value is present. Example: o P1Y3M2D – A period of 1 year, 3 months and 2 days o PT4H56M – A period of time of 4 hours and 56 minutes **LMS Behavior Requirements:**

	• The 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store the latency for an interaction. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.interactions.n.latency* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.interactions.n.latency* prior to the value being 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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.latency*, then the LMS shall set the error code to "408" – Data model dependency not met, return "false" and not change the current state of the 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 element is a brief informative description of the interaction. **Data Element Implementation Requirements:** • **Data Type:** con_ob_com_localized_string (SPM: 255) • **Value Space:** A characterstring that represents a localized

characterstring.

- **Format:** For elements with the con_ob_com_localized_string data type, information is needed to distinguish which language the characterstring is representing. In order to represent this information the following format is required for all con_ob_com_localized_string data types:

The characterstring is required to have the following syntax (phrases in quotes are optional):

"{lang=<*langcode*>}"<*actual characterstring*>

Example:

"{lang=en}Which of the following are red?"

The *{lang=<langcode>}* shall represent the delimiter that indicates the language of the characterstring to follow. The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4].

LMS Behavior Requirements:

- The 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 (See API Implementation Requirements).

SCO Behavior Requirements:

- The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store a short description for an interaction.
- During a SetValue() request, the SCO should be aware the special 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 special 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 special 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 maintained 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 Element Implementation Requirements.
  - If the SCO invokes a request to get the *cmi.interactions.n.description* prior to the value being 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". o If the SCO tries to set the *cmi.interactions.n.descritpion* 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 element based on the request. o The data model binding for collections are 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 persisted (can be determined by requesting the *cmi.interactions.n._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.description*, then the LMS shall set the error code to "408" – Data model dependency not met, return "false" and not change the current state of the element. **Example:** • GetValue("cmi.interactions.0.description") • SetValue("cmi.interactions.0.description","Which of the following are red?") - default language of en is used.

4.2.7.1 Correct Responses Data Element Specifics

The *cmi.interactions.n.correct_responses.n.pattern* 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 the pattern.

Interaction Type	Characterstring Pattern
true_false	The IEEE draft defines the true_false data value as: true_false: state(true, false) **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The IEEE draft defines two state values. The 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. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern", "true")
multiple_choice	The IEEE draft defines the multiple choice data value as: multiple_choice:

bag of set of con_ob_com_short_identifier
// bag SPM: 10, set SPM: 36

This indicates that the characterstring value for multiple choice correct responses needs to be a representation of a bag of a set of identifiers (where the con_ob_com_short_identifier defines the format of the identifiers). The bag contains one or more sets, any of which satisfies the requirement for a correct response. Each set contains one or more con_ob_com_short_identifiers, all of which are required for a correct response. Each of the con_ob_com_short_identifiers represents an expected choice.

The use of the of the index position (n) for the pattern (*cmi.interactions.n.correct_responses.**n**.pattern*) represents the bag. The LMS shall maintain the array of at least 10 (required SPM) patterns for the correct response. The LMS may extend the ability to store more, however, this is implementation specific. The LMS is only responsible for managing the SPM of 10.

The set is represented by the format of the characterstring that is stored in each of the correct_responses array positions. The special reserved token "[,]"shall be used as the delimiter for separating con_ob_com_short_identifiers in the set. The characterstring that is built shall support the SPM of 36 con_ob_com_short_identifiers. This indicates that a LMS only has to support the SPM of 36 con_ob_com_short_identifiers. 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 con_ob_com_short_identifiers.

Data Element Implementation Requirements:

- **Data Type:** characterstring
- **Value Space:** A characterstring that represents sets of con_ob_com_short_identifiers separated by a special reserved token ("[,]"). Each con_ob_com_short_identifier shall be a valid URI as per RFC 2396 [6]. It is recommended that the URI be a URN as per RFC 2141 [3].
  A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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].
- **Format:** The format of the characterstring shall be:
  - con_ob_com_short_identifier"[,]"con_ob_com_short_identifier

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

- The set shall contain at least one con_ob_com_short_identifier. If the set contains more than one con_ob_com_short_identifiers (interaction has multiple correct answers – all of which are required) each shall be separated by the special reserved token "[,]".
- If the interaction contains multiple sets of correct responses, each set shall stored in a separate position in the pattern array (*n*.pattern).
- During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the correct_response. How the LMS and or SCO handles and processes the correct_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.

Example:

- SetValue("cmi.interactions.0.correct_responses.0.pattern","urn:ADL:choice1[,] urn:ADL:choice2[,]urn:ADL:choice3")
- SetValue("cmi.interactions.0.correct_responses.1.pattern","urn:ADL:choice1[,]ur n:ADL:choice2")

The above SetValue() examples indicates that the correct response for the interaction stored at position 0, is either

- "urn:ADL:choice1", "urn:ADL:choice2" and "urn:ADL:choice3" or,

	• "urn:ADL:choice1" and "urn:ADL:choice2"

fill_in	The IEEE draft defines the fill_in data value as:
	
	fill_in:
	bag of record
	{
	case_matters:
	boolean,
	match_text:
	con_ob_com_localized_string(255),
	// 255 parameter value is a SPM
	}// bag SPM: 5
	
	Data Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Format:** The format of the characterstring shall be the following:
	o {case_matters=<boolean>}{lang=<langcode>}<fill-in correct response>
	
	The following requirements shall be adhered to in building the characterstring:
	
	• The `{case_matters=<boolean>}` indicates whether or not the case matters for judgment of correct responses. The `{case_matters=<boolean>}` is a special reserved delimiter that shall be the first set of characters found in the characterstring. The `<boolean>` value shall be either "`true`" or "`false`". The default value for `case_matters` is "`false`". If case does not matter, the characterstring can opt to not include this special reserved delimiter.
	• The `{lang=<langcode>}` indicates the language of the characterstring. The `{lang=<langcode>}` is a special reserved delimiter that shall be either:
	o The first set of characters in the characterstring (if the `{case_matters=true}` special reserved delimiter is not used) or,
	o The second set of characters in the characterstring (if the `{case_matters=true}` special reserved delimiter is not used).
	• The default langcode shall be "en". The `{lang=<langcode>}` is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4].
	• If the interaction has multiple correct answers (e.g., car and automobile are acceptable), then each of these correct responses shall be stored in a separate position in the pattern array (*n*.pattern).
	
	• During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the correct_response. How the LMS and or SCO handles and processes the correct_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.
	
	Example:
	
	• SetValue("cmi.interactions.0.correct_responses.0.pattern", "{case_matters=true}{lang=en}car")
	• SetValue("cmi.interactions.0.correct_responses.0.pattern", "{case_matters=true}{lang=en}automobile"
	
	Indicates that the correct response is "car" or "automobile". The correct response has to be all lowercase and the language of the characterstring is English.
	
	• SetValue("cmi.interactions.0.correct_responses.0.pattern","{lang=en}car")
	
	Indicates that the correct response is "car". The case does not matter, however, the language of the characterstring is English.
	
	• SetValue("cmi.interactions.0.correct_responses.1.pattern","car")
	
	Indicates that the correct response is "car". Since neither of the special reserved delimiters are not used the default value for these reserved delimiters are used (case

	matters is false and language is English.

long_fill_in	The IEEE draft defines the long_fill_in data value as:
	
	fill_in:
	bag of record
	{
	case_matters:
	boolean,
	match_text:
	con_ob_com_localized_string(4096),
	// 4096 parameter value is a SPM
	}// bag SPM: 5
	
	Data Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Format:** The format of the characterstring shall be the following:
	o {case_matters=<boolean>}{lang=<langcode>}<long-fill-in correct response>
	
	The following requirements shall be adhered to in building the characterstring:
	
	• The {case_matters=<*boolean*>} indicates whether or not the case matters for judgment of correct responses. The {case_matters=<*boolean*>} is a special reserved delimiter that shall be the first set of characters found in the characterstring. The <*boolean*> value shall be either "true" or "false". The default value for case_matters is "false". If case does not matter, the characterstring can opt to not include this special reserved delimiter.
	• The {lang=<*langcode*>} indicates the language of the characterstring. The {lang=<*langcode*>} is a special reserved delimiter that shall be either:
	o The first set of characters in the characterstring (if the {case_matters=true} special reserved delimiter is not used) or,
	o The second set of characters in the characterstring (if the {case_matters=true} special reserved delimiter is not used)
	• The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4].
	• If the interaction has multiple correct answers (e.g., The Gettysburg Address or the Emancipation Proclamation are acceptable), then each of these correct responses shall be stored in a separate position in the pattern array (*n*.pattern).
	
	• During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the correct_response. How the LMS and or SCO handles and processes the correct_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.
	
	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…")
	Indicates that the correct response is the Gettysburg Address where case matters (as identified by the <long-fill-in correct response> value). The language of the characterstring is English.
	
	• SetValue("cmi.interactions.0.correct_responses.0.pattern","{lang=en}Four score an seven years ago…")
	Indicates that the correct response is the Gettysburg Address where case does not matter. Since the {case matters=<*boolean*>} special reserved delimiter was

	not used, the default value of "`false`" is used to indicate that case does not matter. The language of the characterstring is English. • SetValue("cmi.interactions.0.correct_responses.0.pattern","car") Indicates that the correct response is "car". Since neither of the special reserved delimiters are not used the default value for these reserved delimiters are used (case matters is false and language is English.
likert	The IEEE draft defines the likert data value as: likert: con_ob_com_short_identifier The likert is a short identifier that matches a choice on a scale. Although a correct response for likert interactions can be specified, likert interactions typically do not have correct responses [1]. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** The characterstring represents the con_ob_com_short_identifier. The con_ob_com_short_identifier shall be a valid URI as per RFC 2396 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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:** • SetValue("cmi.interactions.0.correct_responses.0.pattern","a") Indicates that the correct response for the likert interaction is "a". The SCO is responsible for understanding the identifier value. • SetValue("cmi.interactions.1.correct_responses.0.pattern","urn:ADL:likert_1") Indicates that the correct response for the likert interaction is "urn:ADL:likert_1". The SCO is responsible for understanding the identifier value.

matching	The IEEE draft defines the matching data value as:
	matching:
	bag of bag of record
	{
	source:
	con_ob_com_short_identifier,
	target:
	con_ob_com_short_identifier,
	} // outer bag SPM: 5, inner bag SPM: 36
	The matching data type represents a bag of bag of records. The outer bag contains 1 or more inner bags, any of which satisfies the requirement for a correct response. Each inner bag consists of one or more records, all of which are required for a correct response. Each of the records is a pair of con_ob_com_short_identifiers representing an expected matching input [1].
	The use of the of the index position (n) for the pattern (*cmi.interactions.n.correct_responses.**n**.pattern*) represents the outer bag. The LMS shall maintain the array of at least 5 (required SPM) patterns for the correct response. The LMS may extend the ability to store more, however, this is implementation specific. The LMS is only responsible for managing the SPM of 5.
	The inner bag is represented by the format of the characterstring that is stored in each of the pattern array positions. The special reserved token "[,]"shall be used as the delimiter for

separating con_ob_com_short_identifiers (source/target pairs) in the inner bag. The characterstring that is built shall support the SPM of 36 con_ob_com_short_identifiers. This indicates that a LMS only has to support the SPM of 36 con_ob_com_short_identifiers. 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 con_ob_com_short_identifiers.

Data Element Implementation Requirements:

- **Data Type:** characterstring
- **Value Space:** A characterstring that represents collection of matching pairs representing correct responses.
- **Format:** The characterstring shall have the following format:
  - `<target>` "[.]"`<source>`

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

- The `<target>` is represented as a `con_ob_com_short_identifier`
- The `<source>` is represented as a `con_ob_com_short_identifier`
- The "[.]" is the special reserved token to delimit the <target> and <source>.
- The "[,]" is the special reserved token to delimit multiple <target>/<source> pairs.
- If the interaction has multiple correct answers (e.g., "1[.]a[,]2[.]b[,]3[.]c" and "1[.]b[,]2[.]a[,]3[.]c" are acceptable), then each of these correct responses shall be stored in a separate position in the pattern array (*n*.pattern).
- At least one <target>/<source> pair is required, other pairs are optional.
- During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the correct_response. How the LMS and or SCO handles and processes the correct_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.

Example:

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

Indicates that the correct response for the matching interaction is:

Source	Target
1	a
2	c
3	b

The SCO is responsible for understanding the meaning of the short identifier values.

performance

The IEEE draft defines the performance data value as:

```
performance:
  bag of record
  (
    order_matters:
      boolean,
    answers:
      array(0..254) of record
      // array dimension is an SPM,
      (
        step_name :
          con_ob_com_short_identifier,
        step_answer :
          choice
          (
            state( literal, numeric)
          )
          of
          (
            literal :
```

```
            characterstring (iso-10646-1),
            // SPM 255
          numeric :
            record
            (
              min :
                real (10,7),
              max :
                real (10,7),
            )
        )
    )
) // bag SPM: 5
```

The performance correct response is a bag of records. The bag contains 1 or more records, any of which satisfies the requirement for a correct response.

Each record consists of a flag and an array. The array represents a set of inputs for a correct response. The flag indicates whether the order of the inputs matters for a correct response. If the order matters, the learner must provide each input in the exact order as defined by the array. If the order does not matter, the learner may provide the inputs in any order. The default value (if order_matters is not specified) is "`true`" (order matters)

Each input consists of a name and either a single literal value or a numeric range. If an input is expressed as a literal value, the interaction implementation determines how to use the value to evaluate the corresponding response. If an input is expressed as a numeric range, the learner must provide an input within that range.

Data Element Implementation Requirements:

- **Data Type:** characterstring
- **Format:** The characterstring shall have the following format:
  - `{order_matters=<boolean>}<step_name>"[.]"<step_answer>`

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

- The `{order_matters=<boolean>}` is a special reserved delimiter that indicates whether the order matters. The delimiter is optional. If the delimiter is omitted, then the default value of "true" shall be utilized. No white space shall appear before this tag.
- The `<step_name>` shall be a valid con_ob_com_short_identifier.
- The special delimiter between the `<step_name>` and `<step_answer>` shall be the special reserved token of"[.]".
- The separator between <step_name> and <step_answer> pairs shall be the special reserved token of "[,]". This delimiter can be easily discarded during any parsing routine.
- The `<step_name>` may be omitted if there is no step name but only a `<step_answer>`. In this case, the special reserved token ("[.]") shall still be present before the `<step_answer>`.
- The `<step_answer>` may also be omitted. In this case, the special reserved token ("[.]") shall still be present after the `<step_name>`.
- If the `<step_answer>` consists entirely of a single space character, the `<step_answer>` shall still be preceded by the special reserved token ("[.]").
- If the interaction has multiple correct answers, then each of these correct responses shall be stored in a separate position in the pattern array (*n*.pattern).
- During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the correct_response. How the LMS and or SCO handles and processes the correct_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.

Example:

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

	"{order_matters=true}step_1[.]remove safety[,]step_2[.]fire weapon") • SetValue("cmi.interactions.0.correct_responses.0.pattern","First step[.]inspect wound[,]Second step[.]clean wound[,]Third step[.]apply bandage")
sequencing	The IEEE draft defines the sequencing data value as: sequencing: bag of array (0..35) of con_ob_com_short_identifier, // bag SPM: 5, array dimension is an SPM The sequencing correct response is a bag of arrays of con_ob_com_short_identifiers. The use of the index position (n) for the pattern (*cmi.interactions.n.correct_responses.**n**.pattern*) represents the bag. The bag contains 1 or more arrays, any of which satisfies the requirement for a correct response. The LMS shall maintain the array of at least 5 (required SPM) patterns for the correct response. The LMS may extend the ability to store more, however, this is implementation specific. The LMS is only responsible for managing the SPM of 5. Each array represents a sequence of con_ob_com_short_identifiers for a correct response. The sequence of con_ob_com_short_identifiers are represented as one characterstring. Each short identifier shall be separated by the following special reserved token "[,]". **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Format:** The characterstring shall have the following format: o <sequence_value>"[,]"<sequence_value> The following requirements shall be adhered to in building the characterstring: • The <sequence_value>is represented as a con_ob_com_short_identifier • If numerous correct responses are acceptable, then each of these correct responses shall be stored in a separate position in the pattern array (*n*.pattern). • During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the correct_response. How the LMS and or SCO handles and processes the correct_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **Example:** • SetValue("cmi.interactions.0.correct_responses.0.pattern","a[,]b[,]c[:]b[,]c[,]a") • SetValue("cmi.interaction.0.correct_response.1.pattern","urn:ADL:buildHouse/buyMaterials[,]urn:ADL:buildHouse/buildFoundation[,]urn:ADL:buildHouse/buildFirstFloor[,] urn:ADL:buildHouse/buildSecondFloor") In the first example, the correct response is the sequence of a then b then c, or b then c then a. The SCO is responsible for understanding and managing the values used for the con_ob_com_short_identifiers.
numeric	The IEEE draft defines the numeric data value 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 a range for the correct response. If the

	numbers are the same, then a single number is specified as the correct response [1]. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Format:** The characterstring shall have the following format: o <minimum>"[:]"<maximum> The following requirements shall be adhered to in building the characterstring: • The <minimum> and <maximum> can be optionally used. However at least one shall be present. • **<minimum>[:]<maximum>** Indicates that the correct response is greater than the minimum value supplied and less than the maximum value supplied. If the <minimum> and the <maximum> are identical numbers, then the correct response is that number. • **[:]<maximum>** Indicates that there is no lower bound for the correct response, only an upper bound. • **<minimum>[:]** Indicates that there is no upper bound for the correct response, only a lower bound. Examples: • SetValue("cmi.interactions.0.correct_response.0.pattern","4[:]10") • SetValue("cmi.interactions.0.correct_response.0.pattern", "[:]10") • SetValue("cmi.interactions.0.correct_response.0.pattern", "4[:]") • SetValue("cmi.interactions.0.correct_response.0.pattern", "3.14159[:]3.14159")
other	The IEEE draft defines the other data value as: other: characterstring (iso-10646-1) // SPM 4096 The other correct response is used to support other types of interactions not defined by the standard. The value of the string is left to the implementer of the interaction.

4.2.7.2 Learner Response Data 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 the pattern.

Interaction Type	Characterstring Pattern
true_false	The IEEE draft defines the true_false data value as: true_false: state(true, false) The learner_response is a state that contains the values of "true" or "false". **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The IEEE draft defines two state values. The 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")

multiple_choice	The IEEE draft defines the multiple choice data value as: multiple_choice: bag of con_ob_com_short_identifier A bag of con_ob_com_short_identifiers. If the interaction allowed only one choice, the bag contains only one short identifier. If the interaction allowed more than one choice or more than one combination of choices, the combination of choices made by the learner is represented by the con_ob_com_short_identifiers in the bag. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** The characterstring represents the bag of con_ob_com_short_identifiers. Each con_ob_com_short_identifier (representing the learner's response) shall be delimited by the special reserved token "[,]". Each con_ob_com_short_identifier shall have a SPM of 255 characters. • **Format:** The con_ob_com_short_identifiers shall be represented as a URI. A URI must conform to the syntax defined in RFC 2396. It is recommended that the URI be a URN. A URN is a special case of a URI. 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]. • During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the learner_response. How the LMS and or SCO handles and processes the learner_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **Example:** • SetValue("cmi.interactions.0.learner_response", "urn:ADL:choice1[,]urn:ADL:choice2[,]urn:ADL:choice3") The above SetValue() examples indicates that the learner response for the interaction stored at position 0, where the "[,]" acts as the delimiter for items in the set, is: "urn:ADL:choice1,urn:ADL:choice2,urn:ADL:choice3"

fill_in	The IEEE draft defines the fill_in data value as:
	
	fill_in:
	{
	con_ob_com_localized_string(255),
	// 255 parameter value is a SPM
	}
	
	The learner response for the fill_in interaction type is a localized characterstring.
	
	Data Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Format:** The format of the characterstring shall be the following:
	o {lang=<langcode>}<fill-in learner response>
	
	The following requirements shall be adhered to in building the characterstring:
	• The {lang=<*langcode*>} indicates the language of the characterstring. The {lang=<*langcode*>} is a special reserved delimiter that shall be the first set of characters in the characterstring
	• The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4].
	• During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware

	of the format of the learner_response. How the LMS and or SCO handles and processes the learner_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **Example:** • SetValue("cmi.interactions.0.learner_response", "{lang=en}car") Indicates that the learner response is "car". The language of the characterstring is English. • SetValue("cmi.interactions.0.learner_response","{lang=en}car") Indicates that the learner response is "car". The language of the characterstring is English. • SetValue("cmi.interactions.0.learner_response","car") Indicates that the learner response is "car". Since the language was not indicated in the characterstring the default value for the reserved delimiter is used (language is English.

long_fill_in	The IEEE draft defines the long_fill_in data value as:
	
	fill_in:
	{
	con_ob_com_localized_string(4096),
	// 4096 parameter value is a SPM
	}
	
	The learner response for the long_fill_in interaction type is a localized characterstring.
	
	Data Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Format:** The format of the characterstring shall be the following:
	o {lang=<langcode>}<long-fill-in learner response>
	
	The following requirements shall be adhered to in building the characterstring:
	
	• The {lang=<*langcode*>} indicates the language of the characterstring. The {lang=<*langcode*>} is a special reserved delimiter that shall be the first set of characters in the characterstring
	• The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4].
	
	• During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the learner_response. How the LMS and or SCO handles and processes the learner_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner.
	
	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…")
	Indicates that the learner response is the Gettysburg Address. The language of the characterstring is English.
	
	• SetValue("cmi.interactions.0.learner_response","{lang=en}Four score and seven years ago…")
	Indicates that the learner response is the Gettysburg Address. The language of the

	characterstring is English. • SetValue("cmi.interactions.0.learner_response","Four score and seven years ago…") Indicates that the learner response is the Gettysburg Address. Since the language was not indicated in the characterstring the default value for the reserved delimiter is used (language is English).
likert	The IEEE draft defines the likert data value as: likert: con_ob_com_short_identifier The likert is a con_ob_com_short_identifier that matches a choice on a scale. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** A characterstring represents the con_ob_com_short_identifier. The con_ob_com_short_identifier shall be a valid URI as per RFC 2396 [6]. It is recommended that the URI be a URN as per RFC 2141 [3]. • **Format:** A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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:** • SetValue("cmi.interactions.0.learner_response", "a") Indicates that the learner response for the likert interaction is "a". The SCO is responsible for understanding the identifier value. • SetValue("cmi.interactions.1.learner_response"," urn:ADL:likert_1") Indicates that the learner response for the likert interaction is "urn:ADL:likert_1". The SCO is responsible for understanding the identifier value.

matching	The IEEE draft defines the matching data value as:
	matching:
	bag of record
	{
	source:
	con_ob_com_short_identifier,
	target:
	con_ob_com_short_identifier,
	} // bag SPM: 36
	The learner response for the matching interaction type is a bag that contains zero or more pairs of con_ob_com_short_identifiers. Each pair represents a match made by the learner [1]. The "[,]" is the special reserved delimiter for separating pairs of con_ob_com_short_identifiers in the bag. The characterstring that is built shall support the SPM of 36 con_ob_com_short_identifiers. This indicates that a LMS only has to support the SPM of 36 con_ob_com_short_identifiers. 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 con_ob_com_short_identifiers.
	Data Element Implementation Requirements:
	• **Data Type:** characterstring
	• **Format:** The characterstring shall have the following format:
	o <target>"[.]"<source>
	The following requirements shall be adhered to in building the characterstring:

	• The target is represented as a con_ob_com_short_identifier • The source is represented as a con_ob_com_short_identifier • The special reserved token "[.]" shall be used to separate the target from the source. • If more than one <target> <source> pairs are provided as the learner's response to the interaction. The different sets of pairs shall be delimited by the special reserved token ("[,]"). • During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the learner_response. How the LMS and or SCO handles and processes the learner_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **Example:** • SetValue("cmi.interactions.0.learner_response", "1[.]a[,]2[.]c[,]3[.]b") Indicates that the learner response for the matching interaction is: **Source** / **Target** 1 / A 2 / C 3 / B The SCO is responsible for understanding the meaning of the short identifier values.
performance	The IEEE draft defines the performance data value as: performance: array(0..254) of record (step_name : con_ob_com_short_identifier, step_answer : choice (state(literal, numeric)) of (literal : characterstring (iso-10646-1), // SPM 255 numeric : real (10,7),)) The learner response for the performance interaction type is an array of responses in the order they were provided by the learner in response to the interaction. Each response consists of a step name and either a single literal value or a number. The step names and types of the responses shall match those provided in the *cmi.interactions.n.correct_response* for the interaction, but the response in the learner_response may be in a different order [1]. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Format:** The characterstring shall have the following format: o <step_name>"[.]"<step_answer> The following requirements shall be adhered to in building the characterstring: • The <step_name> shall be a valid con_ob_com_short_identifier. • The special delimiter between the <step_name> and <step_answer> shall be the special reserved token of "[.]".

	• The separator between <step_name> and <step_answer> pairs shall be the special reserved token of "[,]". This delimiter can be easily discarded during any parsing routine. • The <step_name> may be omitted if there is no step name but only a <step_answer>. In this case, the special reserved token ("[.]") shall still be present before the <step_answer>. • The <step_answer> may also be omitted. In this case, the special reserved token ("[.]") shall still be present after the <step_name>. • If the <step_answer> consists entirely of a single space character, the <step_answer> shall still be preceded by the special reserved token ("[.]"). • During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the learner_response. How the LMS and or SCO handles and processes the learner_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **Example:** • SetValue("cmi.interactions.0.learner_response", "step_1[.]remove safety[,]step_2[.]fire weapon") • SetValue("cmi.interactions.0.learner_response","First step[.]inspect wound[,]Second step[.]clean wound[,]Third step[.]apply bandage")
sequencing	The IEEE draft defines the sequencing data value as: sequencing: array (0..35) of con_ob_com_short_identifier, // array dimension is an SPM The learner response for the sequencing interaction type is an array of short identifiers. The sequence determined by the learner is represented by the order of the array. Each short identifier identifies one element that was available to be sequenced. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Format:** The characterstring shall have the following format: o <sequence_value>"[,]"<sequence_value> The following requirements shall be adhered to in building the characterstring: • The <sequence_value>is represented as a con_ob_com_short_identifier • The <sequence_value> values shall separated by the special reserved token "[,]" • During SetValue() and GetValue() processing, LMSs and SCOs, need to be aware of the format of the learner_response. How the LMS and or SCO handles and processes the learner_response is outside the scope of the SCORM. For example, the SCO may need to parse the characterstring returned from the LMS in order to display the value to a learner. **Example:** • SetValue("cmi.interactions.0.learner_response", "a[,]b[,]c") • SetValue("cmi.interaction.0.learner_response","b[,]c[,]a") This first example indicates that the learner's response to the interaction was the sequence of a then b then c. The second example indicates that the learner's response to the interaction was the sequence of b then c then a. The SCO is responsible for understanding and managing the values used for the con_ob_com_short_identifiers.
numeric	The IEEE draft defines the numeric data value as: numeric:

	real (10,7) The numeric learner response is represented as a number with an optional decimal point. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Format:** The real number shall be represented as a characterstring. Examples: • SetValue("cmi.interactions.0.learner_response","4") • SetValue("cmi.interactions.0.learner_response", "3.14159")
other	The IEEE draft defines the other data value as: other: characterstring (iso-10646-1) // SPM 4096 The other learner response is used to support other types of interactions not defined by the standard. The value of the string is left to the implementer of the interaction.

4.2.8. 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 some sort of parameters to the SCO prior to the launch. The `cmi.launch_data` element provides the designer to supply this information.

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

The allowable values for this data 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. This is information that is needed for every use of the SCO, in a certain context.

Dot-Notation Binding	Details
cmi.launch_data	**Data Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 4096) • **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()). **LMS Behavior Requirements:** • The 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 the 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 Item for which the SCO 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 (See API Implementation requirements). **SCO Behavior Requirements:** • The element is implemented by the LMS as read-only. If the SCO desires to use this element, the GetValue() request can be invoked to retrieve the desired information. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.launch_data* element and set the error code to "0" - No error. The characterstring returned shall adhere to the requirements identified in the Data Element Implementation Requirements. o If there is no defined <adlcp:datafromlms> for the Item in which the SCO 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 element based on the request. **Additional Behavior Requirements:** • The SCO developer is responsible for defining the 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 Item that references the SCO. The LMS is responsible for initializing the *cmi.launch_data* for the referenced SCO with the value defined in the element. The launch_data is permitted to be any valid representation of a characterstring (as defined by the above Data Element Implementation Requirements). The LMS shall at least provide the smallest permitted maximum of 4096 characters. A characterstring that is greater than 4096 characters is not guaranteed to be persisted 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")

4.2.9. Learner_id

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

Dot-Notation Binding	Details
cmi.learner_id	**Data Element Implementation Requirements:** • **Data Type:** con_ob_com_object_identifier • **Value Space:** A characterstring (SPM: 4096) that represents a valid Universal Resource Identifier (URI) as per RFC 2396 [6]. It is recommended that the URI be a Universal Resource Name (URN) as per RFC 2141 [3]. • **Format:** A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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]. **LMS Behavior Requirements:** • This 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 the SCORM (e.g., this is typically handled via a learner registration system within the LMS). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated learner identifier currently maintained 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 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 element based on the request. **Example:** • GetValue("cmi.learner_id")

4.2.10. Learner_name

The `learner_name` data element is the name provided for the learner by the LMS [1]. How the `learner_name` is assigned or created is outside the scope of the SCORM. The `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 `learner_name`, however there is no restriction on how this process is accomplished.

Dot-Notation Binding	Details
cmi.learner_name	**Data Element Implementation Requirements:** • **Data Type:** con_ob_com_localized_string (SPM: 255) • **Value Space:** A characterstring that represents a localized characterstring. • **Format:** For elements with the con_ob_com_localized_string data type, information is needed to distinguish which language the characterstring is representing. In order to represent this information the following format is required for all con_ob_com_localized_string data types: The characterstring is required to have the following syntax (phrases in quotes are optional): "{lang=<*langcode*>}"<*actual characterstring*> Example: "{lang=en}Joe Student" The *{lang=<langcode>}* shall represent the delimiter that indicates the language of the characterstring to follow. The default langcode shall be "en". The *{lang=<langcode>}* is optional. If not supplied the default language of the characterstring is "en". The <langcode> shall be conformant to ISO-646 [4]. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as read-only. • The LMS is responsible for initializing *cmi.learner_name*. How this is function is performed is currently outside the scope of the SCORM. This function is typically handled by various mechanisms, some of which are described above. **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. • During a GetValue() request, the SCO should be aware that the special 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 maintained 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 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 element based on the request. **Additional Information:** • How the LMS or SCO interprets and processes the special reserved delimiter

	for the characterstring is outside the scope of the SCORM. The SCO may be built to parse out the language information prior to utilizing the characterstring returned by the LMS. **Example:** • GetValue("cmi.learner_name")

4.2.11. Learner_preference

The information defined by this set of elements defines a set of learner preferences for the learner's use of the SCO during the life cycle of a learner's attempt to satisfy the requirements of the SCO. It is important to note that this information is only available for the given learner attempt at the SCO. If a new attempt is underway any learner preferences are lost from the previous learner attempt . The values are initialized back to default values. All learner preferences would have to be set again for each new learner attempt. How this is done is outside the scope of the SCORM.

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 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the Learner Preference parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements (order of elements is not important): "audio,language,speed, text" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored. **LMS Behavior Requirements:** • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 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	The audio data element is a multiplier value that specifies an intended change in perceived audio 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 attenuation of 10 decibels [1]. **Data Element Implementation Requirements:** • **Data Type:** real(10,7), range (0..*) • **Value Space:** A real number greater than 0. **LMS Behavior Requirements:** • This element is mandatory and shall be implemented by an LMS as read/write. • The SCO is responsible for controlling and managing the associated value. The LMS shall only be responsible for storage and retrieval of the audio value. If the value is requested to be retrieved prior to the value being set, then the LMS shall behave as defined in the API Implementation Requirements. **SCO Behavior Requirements:** • This element is implemented by the LMS as read/write. The SCO has the responsibility for setting this value initially based on the learner and the learning experience. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_preference.audio* value currently being maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.learner_preference.audio* prior to the value being 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.learner_preference.audio* 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* 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 element based on the request. o If the SCO tries to set the *cmi.learner_preference.audio* 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 element based on the request. **Example:** • GetValue("cmi.learner_preference.auido") • SetValue("cmi.learner_preference.audio","3") **Additional Behavior Requirements:** The SCO is responsible for setting the audio value based on a learner's preference. How this value is

	collected, determined or applied to the SCO is outside the scope of the SCORM.
cmi.learner_preference.language	The language data element is the learner's preferred language for SCOs with multilingual capability [1]. **Data Element Implementation Requirements:** • **Data Type:** characterstring (SPM 255) • **Value Space:** iso-646 • **Format:** langcode ("-" subcode)* where langcode is required and multiple, optional, hyphen-prefixed subcodes may follow. The following rules apply to "langcode" [1]: o 2-letter codes are defined by ISO 639 o 3-letter codes are defined by ISO 639-2 o The value "i" is reserved for registrations defined by IANA o The value "x" is reserved for private use The following rules apply to the first "subcode" [1]: o 2-letter subcodes are ISO 3166-1 alpha-2 country codes o Subcodes from 3 to 8 letters are registered with IANA. The default language shall be "en". **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read/write. • The SCO is responsible for controlling and managing the associated value. The LMS shall only be responsible for storage and retrieval of the language value. If the value is requested to be retrieved prior to the value being set, then the LMS shall behave as defined in the API Implementation Requirements. **SCO Behavior Requirements:** • This element is implemented by the LMS as read/write. The SCO has the responsibility for setting this value initially based on the learner and the learning experience. • During a SetValue() request, the SCO should be aware the special 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 special 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 present, the SCO can assume the default language of "en". **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_preference.language* value currently being maintained 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 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, then the LMS shall set the error code to "0" – No error and return the default value of "en".

	• **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 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 element based on the request. **Example:** • GetValue("cmi.learner_preference.language") • SetValue("cmi.learner_preference.language","fr-CA") **Additional Behavior Requirements:** The SCO is responsible for setting the language value based on a learner's preference. How this value is collected, determined or applied to the SCO is outside the scope of the SCORM.

cmi.learner_preference.speed	The speed data element is the learner's preferred relative pace of content delivery. The value is a multiplier of the default pace. For example, 2 is twice as fast as the default delivery pace and 0.5 is one half the default pace. The default value shall be 1 [1].
	Data Element Implementation Requirements:
	• **Data Type:** real(10,7), range (0..*)
	• **Value Space:** A real number greater than 0.
	LMS Behavior Requirements:
	• The element is mandatory and shall be implemented by the LMS as read/write.
	• The SCO is responsible for controlling and managing the associated value. The LMS shall only be responsible for storage and retrieval of the speed value. If the value is requested to be retrieved prior to the value being set, then the LMS shall return the default value of 1.
	SCO Behavior Requirements:
	• This element is implemented by the LMS as read/write. The SCO has the responsibility for setting this value initially based on the learner and the learning experience.
	API Implementation Requirements:
	• **GetValue():** The LMS shall return the associated *cmi.learner_preference.speed* value currently being maintained 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 Element Implementation Requirements.
	  o If the SCO invokes a request to get the *cmi.learner_preference.speed* prior to the value being set by the SCO, then the LMS shall return the default value of "1" and set the error code to "0" – No error and return an 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.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 element based on the request. o If the SCO tries to set the *cmi.learner_preference.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 element based on the request. **Example:** • GetValue("cmi.learner_preference.speed") • SetValue("cmi.learner_preference.speed","0.5") **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 the SCORM.
cmi.learner_preference.text	The text data element specifies whether captioning text is displayed with or in the place of audio [1]. **Data Element Implementation Requirements:** • **Data Type:** state (-1,0,1) • **Value Space:** The IEEE draft defines three state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "-1": Captioning is off and text corresponding to audio is not displayed [1]. o "0": The current default captioning setting [1]. o "1": Captioning is on and text corresponding to audio is displayed [1]. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read/write. • The SCO is responsible for controlling and managing the associated value. The LMS shall only be responsible for storage and retrieval of the text value. If the value is requested to be retrieved prior to the value being set, then the LMS shall return the default value of 0. **SCO Behavior Requirements:** • This element is implemented by the LMS as read/write. The SCO has the responsibility for setting this value initially based on the learner and the learning experience. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.learner_preference.text* value currently being maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.learner_preference.text* prior to the value being set by the SCO, then the LMS shall return the default value of "0" and set the error code to "0" – No error and return an empty characterstring (""). • **SetValue():** If the SCO invokes a request to set the *cmi.learner_preference.text* 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 element based on the request. **Example:** • GetValue("cmi.learner_preference.text") • SetValue("cmi.learner_preference.text","-1") **Additional Behavior Requirements:** How this element is initialized is outside the scope of the SCORM. The value may be initialized in a number of ways. For example: • SCO may be built to set the initial text 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 the SCORM.

4.2.12. Location

The `location` data 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 element 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 location is only understood by the SCO setting the value. If the SCO communicates a location, this element may be used to indicate a "bookmark" or "checkpoint" within the SCO. This element may be used as a starting point upon re-entry into the SCO after a suspended learner session. In this case, the location element corresponds to the SCO exit point the last time the learner experienced the SCO. The behavior and use of this element are managed by the SCO.

Dot-Notation Binding	Details
cmi.location	**Data Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 1000) • **Value Space:** ISO-10646-1:2000 [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()). **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as read/write. • This 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 element is required to be implemented by the LMS as read/write. The SCO, if desired, shall be permitted to retrieve (GetValue()) and/or store (SetValue()) 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 10646-1:2000. 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 maintained 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 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 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`". o If the SCO invokes a SetValue() request to set the *cmi.location* and the supplied value does not meet the Data 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 element based on the request. **Example:** • GetValue("cmi.location") • SetValue("cmi.location","chkPt1.p3.f5")

4.2.13. Max_time_allowed

The `max_time_allowed` data element is the amount of accumulated time the learner is allowed in a learner session with the SCO [1]. As described above (`cmi.session_time`), the SCO developer determines what makes up the learner session.

Dot-Notation Binding	Details
cmi.max_time_allowed	**Data Element Implementation Requirements:** • **Data Type:** timeinterval (seconds,10,2) - a time interval with resolution to 0.01 seconds • **Format:** P[yY][mM][dD][T[hH][mM][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) The character literal designators "P","Y","M","D","T","H","M","S" shall appear if the corresponding non-zero value is present. Example: o P1Y3M2D – A period of 1 year, 3 months and 2 days o PT4H56M – A period of time of 4 hours and 56 minutes **LMS Behavior Requirements:** • This 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 associated with the content aggregation. The LMS shall use the <imsss:attemptAbsoluteDurationLimit> element, if defined for the Item referencing the SCO Resource in the manifest, to initialize this value. **SCO Behavior Requirements:** • This element is implemented by the LMS as read-only. The element is optionally used by the SCO, If desired, the SCO is only permitted to retrieve the value for this element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.max_time_allowed* element and set the error code to "0" – No error. The characterstring returned shall adhere to the requirements identified in the Data Element Implementation Requirements. o If there is no maximum time allowed defined in the manifest and the SCO requests the value for this 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 element based on the request. **Example:**

	• GetValue(“cmi.max_time_allowed”)

4.2.14. Mode

The `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 SCOs 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. Designers may enable SCOs to behave in virtually an unlimited amount of ways.

Dot-Notation Binding	Details
cmi.mode	**Data Element Implementation Requirements:** • **Data Type:** state (browse, normal, review) [1] • **Value Space:** The IEEE draft defines three state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "browse": The SCO is presented without the intent of recording status (i.e., passed, completed, failed) [1]. o "normal": The SCO is presented with the intent of recording status (i.e., passed, completed, failed) [1]. This is the default value if no mechanism is in place to identify the mode. o "review": The SCO has a recorded status (i.e., passed, completed, failed) and is presented without the intent of updating its recorded status [1]. **LMS Behavior Requirements:** • This 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 aggregation or a way of reviewing a content aggregation, then this is one mechanism for initializing the mode in which the content (SCO) in the content aggregation should be viewed. The "normal" mode shall be the default mode for all SCOs. **SCO Behavior Requirements:** • The element is implemented by an LMS as read-only. If the SCO desires the use of this element, then the SCO can only retrieve the value for this element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.mode* currently maintained 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 element based on the request. **Example:** • GetValue("cmi.mode")

4.2.14.1 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:

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

Table 4.2.14.1a: Mode and Credit Values

If credit is "`no-credit`" then any information (success_status, interactions, etc.) communicated by the SCO is informative. Data is used to make sequencing decisions, other than that the LMS is free to do what it wants with the data.

4.2.15. Objectives

Instructional designers may wish to associate learning or other types of objectives with a learning activity and its associated content object. The SCORM does not define what an objective is or place requirements on its use. However, the 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 (with the exception of *Persist State*). 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)

During the learner's interaction with the SCO, the SCO is permitted to update status information/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 SCORM Meta-data. The Purpose, Taxonpath and Description elements can also be used to describe and identify the SCO's objective(s). However, applying Meta-data to a SCO is for informative purposes only; currently, there is no requirement defined for an LMS to process the Mata-data associated with s SCO and there is no behavior defined for an LMS to interoperably utilize the Meta-data associated with a SCO (Refer to the SCORM Content Aggregation Model for more details).

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

4.2.15.1 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 objective status information is an indexed list (Refer to Section *4.2.1.3 Handling Collections*). When setting (`SetValue()`) objective status information the SCO is responsible for inserting objective status information in sequential order.

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"

Figure 4.2.8.1a: Scenarios for Storing Collection Data

To avoid the incorrect scenario shown above, in Figure 4.2.8.1a, 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 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.

4.2.15.2 Utilizing Objective Status Information for Sequencing

Instructional designers may wish to use objective status information to make conditional sequencing decisions; this desire is explicitly represented in the sequencing information associated with a learning activity (see the SCORM Content Aggregation Model book). As a SCO references sets of objective status information through identifiers, so does the sequencing information associated with a learning activity. The SCORM defines how identifiers are used to relate objective tracking information obtained at run-time, during the learner experience with the SCO, to the objectives defined on the activity for sequencing purposes.

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 run-time data model Objectives data model element for the SCO associated with that learning activity. When a SCO is launched for a new learner attempt, the LMS will initialize a set of run-time objectives (*cmi.objectives.n.xxx*) for the SCO with the objective status information managed for sequencing the SCO's associated learning activity. During the learner experience, the SCO may modify the status of these sets of

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

Dot-Notation Binding	Details
cmi.objectives._children	The *cmi.objectives._children* data model element represents a listing of supported data model elements. This 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the Objectives parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements (order of elements is not important): "id,score,success_status,completion_status" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored. **LMS Behavior Requirements:** • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 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 _count keyword is used to describe the current number of objectives being stored by the LMS. The total number of entries currently being persisted by the LMS shall be returned. The LMS is responsible for supporting the smallest permitted maximum of 100 objectives. **Data Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 • **Format:** The characterstring representing the number of objectives that the LMS is currently persisting. **LMS Behavior Requirements:** • This 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 element is implemented by an LMS as read-only. The SCO is only permitted to retrieve the value stored by the LMS. **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 id data element is an identifier for an objective and shall be unique at least within the scope of the SCO. The id data element shall contain a valid value if either the score or status data elements described below is implemented [1]. If a SCO is requesting to store objective information the SCO is required to set the identifier first (unless it was initialized by another means), prior to any other objective information. **Data Element Implementation Requirements:** • **Data Type:** con_ob_com_object_identifier • **Value Space:** A characterstring (SPM: 4096) that represents a valid Universal Resource Identifier (URI) as per RFC 2396 [6]. It is recommended that the URI be a Universal Resource Name (URN) as per RFC 2141 [3]. • **Format:** A URI must conform to the syntax defined in RFC 2396. A URN is a special case of a URI. 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]. **LMS Behavior Requirements:** • This element is mandatory and shall be implemented by an LMS as read/write. • If the <imsss:objectives> are defined for <imscp:item> elements in the content package manifest, then the LMS is responsible for initializing the objective status information for the SCO referenced by the <item>. Each objective defined (<imsss:primaryObjective> or <imsss:objective>) has a required objectiveID attribute. This 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 needs to be initialized. The LMS is also responsible for initializing status and score for the objective information data if the that information is available to the LMS (for more information see the SCORM Sequencing and Navigation book – Global objectives). **SCO Behavior Requirements:** • The element is read/write. The SCO is permitted to retrieve and store objective identifiers for the SCO.

	• 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. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated objectives identifier currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to retrieve the objective identifier and the index (n) is not available or has not been set by the SCO previously, then the LMS shall return an empty characterstring ("") and set the error code to "408" Data model dependency not established. • **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". o If the supplied value of the SetValue() does not meet the requirements of the Data 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 element based on the request . o Since the *cmi.objectives.n.id* is required to be set first prior to any other objective information, if the SCO sets objective information (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 element based on the request. o Collection 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*. **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	The score data element is the score achieved by the learner for the objective [1]. This data indicates the performance of the learner on the objective during a learner session with the SCO as determined by the SCO.
cmi.objectives.n.score._children	The *cmi.objectives.n.score._children* data model element represents a listing of supported data model elements. This 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the Score parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements (order of elements is not important): "scaled,raw,min,max" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored. **LMS Behavior Requirements:** • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 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 scaled data element is a number that reflects the performance of the learner for the objective. The value of the data element is scaled to fit the range –1 to 1 inclusive [1]. **Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. The value shall be in the range of -1.0 to 1.0, inclusive. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read/write. • The SCO is responsible for determining the scaled score. 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 (See API Implementation Requirements). • Impacts on Sequencing: (1) 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. (2) 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 element is implemented by an LMS as read/write. If the SCO desires the use of the element, the SCO can use this element to retrieve and store the scaled score. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.scaled* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.objectives.n.score.scaled* prior to the value being 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". o 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 element based on the request. o 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 out of range, return "false". The LMS shall not alter the state of the element based on the request. 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.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 element based on the request. o The data model binding for collections are 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 persisted, 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 Information: • If there is sequencing information applied to the learning activity associated with the SCO that relies on measure, the SCO must ensure score 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 measure for the appropriate objective (based on objective IDs) of the learning activity associated with the SCO when processing sequencing information. **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 raw data 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 Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. **LMS Behavior Requirements:** • The 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 element is implemented by an LMS as read/write. If the SCO desires the use of this element, the SCO can use this element to retrieve and store objectives raw scores. 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 maintained 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 Element Implementation Requirements. o The data model binding for collections are 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 "403" – Data model element value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.objectives.n.score.raw* and 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.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 element based on the request. o The data model binding for collections are 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 persisted, 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 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 min data element is the minimum value, for the objective, in the range for the raw score [1]. **Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. **LMS Behavior Requirements:** • The 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, the SCO can use this element to retrieve and store minimum scores. The minimum score is determined by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.min* currently maintained 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 Element Implementation Requirements. o The data model binding for collections are 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 "403" – Data model element value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.objectives.n.score.min* and 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.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.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 element based on the request. o The data model binding for collections are 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 persisted, 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 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 max data element is the maximum value, for the objective, in the range for the raw score [1]. **Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. **LMS Behavior Requirements:** • The 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, the SCO can use this element to retrieve and store maximum scores. The max score

	is determined by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.objectives.n.score.max* currently maintained 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 Element Implementation Requirements. o The data model binding for collections are 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 "403" – Data model element value not initialized and return an empty characterstring (""). o If the SCO attempts to retrieve the *cmi.objectives.n.score.max* and 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.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". o 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 element based on the request. o The data model binding for collections are 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 persisted, 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.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 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 success_status data element indicates whether the learner has passed the objective [1]. How the SCO determines the success_status for the objective is outside the scope of the 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.

Data Element Implementation Requirements:

- **Data Type:** state (passed, failed, unknown)
- **Value Space:** The IEEE draft defines three state values. The SCORM binds these state values to the following restricted vocabulary tokens:
  - "passed": The learner has passed the SCO [1]. Indicates that the necessary number of objectives was mastered or a necessary score was achieved.
  - "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.
  - "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 element is mandatory and shall be implemented by an LMS as read/write.
- Normally the SCO will report its own success_status to the LMS, however there is no requirement in the SCORM that mandates a SCO to set success_status. If 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.
- Impacts on Sequencing:
  (1) If the SCO sets 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.
  (2) If the SCO sets 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.
  (3) If the SCO sets 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 element is optionally used by a SCO. The SCO can read/write this data model element if needed.

API Implementation Requirements:

- **GetValue():** The LMS shall return the objectives associated success_status currently maintained 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 Element Implementation Requirements.
  - Until the SCO sets the success_status, the default value of the element shall be "unknown".
- **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 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 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 element based on the request. ○ The data model binding for collections are 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 persisted, 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 element based on the request. **Additional Information:** • 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. **Example:** • GetValue("cmi.objectives.n.success_status") • SetValue("cmi.objectives.n.success_status","passed")
cmi.objectives.n.completion_status	The completion_status data element indicates whether the learner has completed the associated objective [1]. How the SCO determines the completion_status for the objective is outside the scope of the SCORM. For example, the SCO could base this decision on a number of interactions associated with the objective being completed. **Data Element Implementation Requirements:** • **Data Type:** state (completed, incomplete, not_attempted, unknown) • **Value Space:** The IEEE draft defines four state values. The SCORM binds these state values to the following restricted vocabulary tokens: ○ "completed": The learner has experienced enough of the objective for it to be considered complete [1]. How completion is determined is controlled and managed by the SCO. ○ "incomplete": The learner has started the objective but did not finish [1]. How completion is determined is controlled and managed by the SCO. ○ "not attempted": The learner has not accessed the objective, or the learner previously has accessed the objective but has experienced so little of it that the objective is considered it to be not attempted [1].

	The SCO is responsible for determining whether or not the objective was attempted or not. 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 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 the SCORM that mandates a SCO to set completion_status. If the SCO does not set the completion_status, then the LMS shall treat the completion_status as "unknown". **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, the SCO can retrieve and store the completion_status of the objectives. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated completion_status currently maintained 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 Element Implementation Requirements. o Until the SCO sets the completion_status, the default value of the element shall be "unknown". • **SetValue():** If the data dependency has been met (the identifier for the objective has been previously set for the given index *n* and the supplied value meets the Data Element Implementation Requirements) the LMS shall set the *cmi.objectives.n.completion_status* data model element to the supplied value, set the error code to "0" – No error and return "true". o If the SCO invokes a request to set the 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 element based on the request. o The data model binding for collections are 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 persisted, 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.completion_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 element based on the request. **Additional Information:**

	• Since the determination of completion_status is controlled and managed by the SCO, the LMS cannot imply that the SCO is completed in any way. If no completion_status is reported by the SCO, then the LMS can only rely on the fact that the completion_status is "unknown". If there are sequencing rules built that rely on completion status the SCO will have to ensure that data is sent to the LMS (SetValue()), if not the completion status as used by LMS sequencing engines will be considered "unknown". **Example:** • GetValue("cmi.objectives.0.completion_status") • SetValue("cmi.objectives.0.completion_status","incomplete")

4.2.16. Scaled_passing_score

The scaled_passing_score data element is the scaled passing score required to master the SCO. The value of the data element is scaled to fit the range -1 to 1 inclusive [1]. Indicates the passing scaled score for a SCO.

Dot-Notation Binding	Details
cmi.scaled_passing_score	**Data Element Implementation Requirements:** • **Data Type:** real(10,7) range (-1 .. 1) • **Value Space:** -1.0 <= scaled_passing_score <= 1.0 **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read-only. • The LMS is responsible for initializing this element based using the IMS Simple Sequencing namespace element <imsss:minNormalizedMeasure> element associated with an <imsss:primaryObjective> element for the <item> element that references a SCO resource. If the value is not provided in the manifest, then the LMS shall not make any assumptions of a scaled passing score. **SCO Behavior Requirements:** • The element is implemented by an LMS as read-only, If desired, the SCO is only permitted to retrieve the value for this element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.scaled_passing_score* element and set the error code to "0" – No error. The characterstring returned shall adhere to the requirements identified in the Data Element Implementation Requirements. o If there is no scaled passing score defined in the manifest and the SCO requests the value for this 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.scaled_passing_score,* 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 element based on the request. **Example:** • GetValue("cmi.scaled_passing_score")

4.2.17. Score

The score data element is the learner's score for the SCO [1]. The score data model element is broken into four sub-elements:

- Scaled: The scaled data element is a number that reflects the performance of the learner. The value of the data element is scaled to fit the range –1.0 to 1.0 inclusive [1].
- Raw: The raw data element is a number that reflects the performance of the learner relative to the range bounded by the values of min and max [1].
- Minimum: The min data element is the minimum value in the range for the raw score [1].
- Maximum: The max data element is the maximum value in the range for the raw score [1].

Dot-Notation Binding	Details
cmi.score._children	The *cmi.score._children* data model element represents a listing of supported data model elements. This 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 Element Implementation Requirements:** • **Data Type:** characterstring • **Value Space:** ISO-10646-1 [5] • **Format:** A comma-separated list of all of the elements in the Score parent element that are supported by the LMS. Since all elements are required to be supported by the LMS, the characterstring shall represent the following elements: "scaled,min,max,raw" The order of the elements is not important. The element names shall be considered reserved tokens and all whitespace is ignored. **LMS Behavior Requirements:** • This 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 (See Data Element Implementation Requirements above). **SCO Behavior Requirements:** • This element is required to be implemented by an LMS as read-only. The SCO only has the ability to retrieve this value (GetValue()). • The SCO is not permitted to invoke the SetValue() request for this data model element. **API Implementation Requirements:** • **GetValue():** The LMS shall return a comma-separated list of data model elements supported by the LMS (See Data 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 Element Implementation Requirements. • **SetValue():** If the SCO invokes a SetValue() request to set the *cmi.score_children*, then the LMS shall set the error code to "`404`" – Data model element is read only and return "`false`". **Example:**

	• GetValue("cmi.score._children")
cmi.score.scaled	**Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. The value shall be in the range of –1.0 to 1.0, inclusive. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read/write. • The SCO is responsible for determining the scaled score. The LMS cannot make any judgments of the scaled score unless reported 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 (See API Implementation Requirements). • Impacts on Sequencing: (1) If the SCO does not set *cmi.score.scaled*, the Objective Measure Status for the primary objective of the learning activity associated with the SCO shall be false. (2) If the SCO sets *cmi.score.scaled*, the Objective Measure Status for the primary objective of the learning activity associated with the SCO shall be true, and the Objective Normalized Measure for the primary objective of the learning activity associated with the SCO shall equal the value of score.scaled. **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of the element, then the SCO can use this element to retrieve and store the scaled score. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.score.scaled* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.score.scaled* prior to the value being 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.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". o If the SCO tries to set the *cmi.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 element based on the request. o If the SCO tries to set the *cmi.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 out of range, return "false". The LMS shall not alter the state of the element based on the request. **Additional Information:** • If there is sequencing information applied to the learning activity associated with the SCO that relies on measure, the SCO must ensure score 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 measure for the primary objective of the learning activity associated with the SCO when processing sequencing information. **Example:** • GetValue("cmi.score.scaled")

	• SetValue("cmi.score.scaled","0.750033") • SetValue("cmi.score.scaled","0.75")
cmi.score.raw	**Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. **LMS Behavior Requirements:** • The 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 raw 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store raw scores. The raw score may be determined and calculated in any manner that makes sense to the SCO. For instance, it could reflect the percentage of objectives complete, it could be the raw score on a multiple choice test or it could indicate the number of correct first responses to embedded questions in a SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.score.raw* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.score.raw* prior to the value being 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.score.raw* 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.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 element based on the request. **Example:** • GetValue("cmi.score.raw") • SetValue("cmi.score.raw","75.0033") • SetValue("cmi.score.raw","0.75")
cmi.score.max	**Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. **LMS Behavior Requirements:** • The 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 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires

	the use of this element, then the SCO can use this element to retrieve and store maximum scores. The max score is determined by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.score.max* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.score.max* prior to the value being 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.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". o If the SCO tries to set the *cmi.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 element based on the request. **Example:** • GetValue("cmi.score.max") • SetValue("cmi.score.max","1.0") • SetValue("cmi.score.max","500")
cmi.score.min	**Data Element Implementation Requirements:** • **Data Type:** real (10,7) • **Value Space:** A real number with values that is accurate to seven significant decimal figures. **LMS Behavior Requirements:** • The 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 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 (See API Implementation Requirements). **SCO Behavior Requirements:** • The element is implemented by an LMS as read/write. If the SCO desires the use of this element, then the SCO can use this element to retrieve and store minimum scores. The max score is determined by the SCO. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.score.min* currently maintained 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 Element Implementation Requirements. o If the SCO invokes a request to get the *cmi.score.min* prior to the value being 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.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.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 element based on the request. **Example:** • GetValue("cmi.score.min")

	• SetValue("cmi.score.min","1.0") • SetValue("cmi.score.min","500")

4.2.18. Session_time

The session_time data element is the amount of time that the learner has spent in the current learner session for this SCO. If no learner session is in progress, then the session time is the time for the last learner session for this SCO [1].

If the SCO is going to track session_time, the SCO determines the value and meaning of session_time. Examples:

1. The SCO may not start recording session time until after it has initialized a media segment.
2. If the learner takes a break in the learner session, then the break time may or may not be included in the reported session_time [1].

The SCO should be responsible for tracking any and all times. This includes implementations of internal time clocks for recording such times.

Dot-Notation Binding	Details
cmi.session_time	**Data Element Implementation Requirements:** • **Data Type:** timeinterval (seconds,10,2) - a time interval with resolution to 0.01 seconds • **Format:** 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) The character literal designators "P","Y","M","D","T","H","M","S" shall appear if the corresponding non-zero value is present. Example: o P1Y3M2D – A period of 1 year, 3 months and 2 days o PT4H56M – A period of time of 4 hours and 56 minutes **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as write-only. • Since this element is implemented by the LMS as write-only, the LMS is not responsible for initializing this data model element. It is the responsibility of the SCO to manage this value. The LMS is only responsible for accepting a SetValue() call to this element and perform the accumulation with *cmi.total_time*. • Since a SCO is not required to set a value for this element (not required to keep track of the session time), an LMS shall keep track of session time from the time the LMS launches the SCO. If the SCO reports a different session time, then the LMS shall use the session time as reported by the SCO instead of the session time as measure by the LMS. **SCO Behavior Requirements:** • The element is implemented by the LMS as write-only. The element is optionally used by the SCO. A SCO shall not attempt to request (GetValue()) this data model element. A SCO is permitted, in a single communication session, to perform multiple sets of session time (*cmi.session_time*). SCO developers should be aware of the fact that when the SCO issues the Terminate("") or the user navigates away, the LMS shall

	take the last *cmi.session_time* that the SCO set (if there was one set) and accumulate this time to the *cmi.total_time*. **API Implementation Requirements:** • **GetValue():** If the SCO invokes a request to get the *cmi.session_time,* 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.session_time* to the supplied value. The supplied value shall meet the Data Element Implementation Requirements defined above. If the supplied value is correctly formulated, then the LMS shall set the value, return "true" and set the error code to "0" - No error. o If the supplied value does not meet the Data Element Implementation Requirements identified 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 element based on the request. **Additional Behavior Requirements:** • This value is used to keep track of the time spent in a SCO for a learner session. The LMS shall use this value in determining the *cmi.total_time*. A SCO is able, in a single learner session, to perform multiple sets (SetValue()) of the *cmi.session_time*. When the SCO issues the Terminate("") or the user navigates away, the LMS shall take the last *cmi.session_time* that the SCO set (if there was a set) and accumulate this time to the *cmi.total_time*. Upon subsequent launch of the SCO in the same learner attempt, and a GetValue() call for *cmi.total_time*, the LMS shall return the accumulated time. LMS's shall not accumulate the multiple session times sent via the SetValue() request. If multiple calls to SetValue() for *cmi.session_time* are made, then the LMS shall internally keep track of the various session times and only use the last value set when persisting the *cmi.session_time*. **Example:** • SetValue("cmi.session_time","PT1H5M")

4.2.19. Success_status

The success_status data element indicates whether the learner has passed the SCO [1]. How the SCO determines its success_status is outside the scope of the SCORM. The SCO could base this decision on a certain percentage of interactions being passed, a certain percentage of objectives being met, a total score for a test or quiz compared against a mastery score, etc. This value indicates the overall success status for the SCO as determined by the SCO developer.

Dot-Notation Binding	Details
cmi.success_status	**Data Element Implementation Requirements:** • **Data Type:** state (passed, failed, unknown) • **Value Space:** The IEEE draft defines three state values. The 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 element is mandatory and shall be implemented by an LMS as read/write. • Normally the SCO will report its own success_status to the LMS, however there is no requirement in the SCORM that mandates a SCO to set success_status. Since this is the case, the LMS shall adhere to the following success_status determination rules: (1) If the SCO does not set success_status and no mastery score (*cmi.scaled_passing_score*) information is available, then not enough significant information is available. The success_status shall be "unknown". (2) If the SCO sets success_status and no mastery score (*cmi.scaled_passing_score*) information is available, then the LMS shall rely only on the value set by the SCO. (3) If the SCO sets success status (*cmi.success_status*) and a scaled score (*cmi.score.scaled*); and mastery score (*cmi.scaled_passing_score*) information is available, then the LMS shall override the value set by the SCO. The value used shall be determined by comparing the scaled score with the mastery score. If the scaled score is greater than or equal to the mastery score, then the LMS shall set the success_status to "passed". If the scaled score is less than the mastery score, then the LMS shall set the success_status to "failed". If no scaled score is set by the SCO, then the LMS should rely on the success_status provided by the SCO. (4) If there is mastery score (*cmi.scaled_passing_score*) information provided by the SCO developer and the SCO does not set the success_status, then the LMS shall determine the status depending on the learner's score (*cmi.score.scaled*) compared to the mastery score (*cmi.scaled_passing_score*). If no scaled score is provided by the SCO, then the LMS shall not alter the *success_status*. • Impacts on Sequencing:

(1) If the SCO or LMS (through the above process) sets success_status, of the SCO to "unknown", the Objective Progress Status for the primary objective of the learning activity associated with the SCO shall be false.
(2) If the SCO or LMS (through the above process) sets success_status, of the SCO to "passed", the Objective Progress Status for the primary objective of the learning activity associated with the SCO shall be true, and the Objective Satisfied Status for the primary objective of the learning activity associated with the SCO shall be true.
(3) If the SCO or LMS (through the above process) sets success_status, of the SCO to "failed", the Objective Progress Status for the primary objective of the learning activity associated with the SCO shall be true, and the Objective Satisfied Status for the primary objective of the learning activity associated with the SCO shall be false.

SCO Behavior Requirements:

- The element is implemented by the LMS as read/write. The SCO, if desired, shall be permitted to retrieve (GetValue()) and/or store (SetValue()) the data model element.
- The SCO should be aware of the success_status determination rules that shall be adhered to by an LMS. These rules outline how and when an LMS can override a success_status reported by the SCO.
- The SCO should be aware that setting the success_status will affect the learning activity associated with the SCO, therefore possibly affecting sequencing.

API Implementation Requirements:

- **GetValue():** The LMS shall return the associated success_status currently maintained 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 Element Implementation Requirements.
  - Until some determination factor is present, the default value of the *cmi.success_status* is "unknown".
- **SetValue():** If the SCO invokes a request to set the success_status 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 element based on the request.

Additional Information:

- If there is sequencing information applied to the learning activity associated with the SCO that relies on success status, the SCO must ensure success 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 primary objective of the learning activity associated with the SCO when processing sequencing information.

Example:

- GetValue("cmi.success_status")
- SetValue("cmi.success_status","passed")

4.2.20. Suspend_Data

During a learning experience, the learner or SCO may wish to suspend the learner attempt on the SCO and resume the learner attempt later. It is the SCO's responsibility to provide some mechanism for the learner to suspend the current learner attempt. If the learner attempt on the SCO is suspended, the state of the SCO's run-time data will persist until the next learner session on the SCO (if the `cmi.exit` is set to "`suspend`"). The suspend_data data model element provides additional space to store and retrieve suspend data between learner sessions; suspend data may be used by the SCO to resume the learner attempt.

Note, the use of `suspend_data` is closely related to `cmi.exit`. If `cmi.exit` is not set to "`suspend`" prior to a learner session on the SCO ending, the learner attempt on the SCO also ends. In this case, with the exception of SCO's associated learning activity identified with *Persist State*, the state of suspend_data (and all other data model elements) will not be available to the SCO if the SCO is launched in a future learner attempt.

The suspend_data data element provides information that may be created by a SCO as a result of a learner accessing or interacting with that SCO. The format of the content of this data element is unspecified [1].

The intent is for the SCO to store data for later use in the current learner session or a subsequent learner session between the SCO and the same learner. The LMS shall not interpret or change this data [1].

This element can typically be used to store information that the SCO may need upon resumption from a suspended state, for which the lesson_location cannot be utilized. The data could also be used later on in the same learner session (no suspended state).

Dot-Notation Binding	Details
cmi.suspend_data	**Data Element Implementation Requirements:** • **Data Type:** characterstring (SPM: 4096) • **Value Space:** ISO-10646-1 • **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 to by the SCO (GetValue()). **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by the LMS as read/write. • The LMS is not responsible for the initialization of this data. The LMS is only responsible for storing and retrieve the data for the SCO. If the SCO requests the value before any suspend_data has been set, then the LMS shall behave according to the API Implementation Requirement behaviors defined below. **SCO Behavior Requirements:** • The element is implemented by the LMS as read/write. If the SCO desires the use of this element, the SCO can retrieve and store suspend_data dealing with the learner's attempt. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.suspend_data* and set the error code to "0" - No error. The characterstring returned shall adhere to the requirements identified in the Data Element Implementation Requirements.

	○ If the SCO has not set the value prior to the get request, 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.suspend_data* to the data supplied, set the error code to "0" - No error and return "true". The value must be a valid characterstring as defined b the Data Element Implementation Requirements. ○ If the supplied value is not a valid characterstring according to the Data Elements Implementation Requirements 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 element based on the request. **Additional Behavior Requirements:** • The SCO is responsible for the information to be stored by the LMS. The data sent to the LMS is permitted to be any valid representation of a characterstring. The LMS shall at least provide the smallest permitted maximum of 4096 characters. A characterstring that is greater than 4096 characters is not guaranteed to be persisted in its entirety by an LMS. The intent is for the SCO to store data for later use in the current learner session or subsequent learner sessions between the SCO and the same learner. Since the SCO is responsible for the format of the *cmi.suspend_data*, the LMS shall not attempt to interpret or change this data in any way. • The LMS shall provide this data as set in a previous learning session only if the SCO as set the value of *cmi.exit* to "suspend" (a suspended learner session) in the previous learner session. In other words, the LMS shall keep the *cmi.suspend_data* available during the same learner session, regardless of the value of *cmi.exit*. However, if a SCO calls Finish("") without having set *cmi.exit* to "suspend", the LMS may discard the value of the *cmi.suspend_data* at its convenience and shall not give this data back to the SCO in a later session. This behavior also applies to any other data that was set during the previous session. **Example:** • SetValue("cmi.suspend_data","<data><intID>1001</intID><ans>A</ans></data>") • SetValue("cmi.suspend_data","A1;B2;C11-3")

4.2.21. Time_limit_action

The `time_limit_action` data element indicates what the SCO should do when `max_time_allowed` is exceeded [1].

Dot-Notation Binding	Details
cmi.time_limit_action	**Data Element Implementation Requirements:** • **Data Type:** state (exit_message, continue_message, exit_no_message, continue_no_message) • **Value Space:** The IEEE draft defines four state values. The SCORM binds these state values to the following restricted vocabulary tokens: o "exit,message": The learner is forced out of the SCO. The SCO shall provide a message to the learner indicating that the maximum time allowed to complete the SCO was exceeded [1]. o "continue,message": The learner is allowed to continue in the SCO. The SCO shall provide a message to the learner indicating that the maximum time allowed in the SCO was exceeded [1]. o "exit,no message": The learner is forced out of the SCO with no message [1]. o "continue,no message": Although the learner has exceeded the maximum time allowed, the learner is given no message and should not be forced out of the SCO [1]. **LMS Behavior Requirements:** • The element is mandatory and shall be implemented by an LMS as read-only. The data model element is initialized using the ADL Content Packaging namespace element <adlcp:timelimitaction>. This element shall only be placed on an <item> element that references a SCO resource, found in a Content Package manifest. **SCO Behavior Requirements:** • This element is implemented by the LMS as read-only. The element is optionally used by the SCO, If desired, the SCO is only permitted to retrieve the value for this element. **API Implementation Requirements:** • **GetValue():** The LMS shall return the value stored for the *cmi.time_limit_action* element and set the error code to "`0`" – No error. The characterstring returned shall adhere to the requirements identified in the Data Element Implementation Requirements. o If there is no time limit action defined in the manifest and the SCO requests the value for this 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.time_limit_action,* 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 element based on the request. **Example:** • GetValue("cmi.time_limit_action")

4.2.22. Total_time

The value of the total_time data element is the sum of all the learner's session times (`cmi.session_time`) accumulated in the current learner attempt prior to the current learner session [1]. This data model element is used to track the total time spent in all of the learner's sessions for a given learner attempt (Refer to Section *2.1.1 Run-Time Environment Temporal Model* for more details on learner attempts and learner sessions).

Dot-Notation Binding	Details
cmi.total_time	**Data Element Implementation Requirements:** • **Data Type:** timeinterval (seconds,10,2) - a time interval with resolution to 0.01 seconds • **Format:** P[yY][mM][dD][T[hH][mM][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) The character literals designators "P","Y","M","D","T","H","M","S" shall appear if the corresponding non-zero value is present. Example: o P1Y3M2D – A period of 1 year, 3 months and 2 days o PT4H56M – A period of time of 4 hours and 56 minutes **LMS Behavior Requirements:** • This element is mandatory and shall be implemented by the LMS as read-only. • Since this element is implemented by the LMS as read-only, it is the responsibility of the LMS to manage this data. Since this value is the accumulated session times (*cmi.session_time*), the LMS cannot determine this value until the SCO sets session times. If the SCO requests the value before any session times have been set, then the LMS shall behave according to the API Implementation Requirement behaviors defined below. **SCO Behavior Requirements:** • The element is implemented by an LMS as read-only. If the SCO desires the use of this element, then the SCO can only retrieve the value of total_time. **API Implementation Requirements:** • **GetValue():** The LMS shall return the associated *cmi.total_time* currently being maintained 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 Element Implementation Requirements. o If the SCO has not set any session times (*cmi.session_time*), then the LMS cannot determine the total_time value. In these cases, 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 request to set the *cmi.total_time,* 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 element based on the request. **Additional Behavior Requirements:** • A SCO is permitted, in a single communication session, to perform multiple sets of session time (*cmi.session_time*). When the SCO issues the Terminate("") or the user navigates away, the LMS shall take the last

	cmi.session_time that the SCO set (if there was one set) and accumulate this time to the *cmi.total_time*. **Example:** • GetValue("cmi.total_time")

APPENDIX A
ACRONYM LISTING

This page intentionally left blank.

Acronym Listing

ADL	Advanced Distributed Learning
AICC	Aviation Industry CBT Committee
API	Application Programming Interface
ARIADNE	Alliance of Remote Instructional Authoring & Distribution Networks for Europe
CMI	Computer Managed Instructions
DOM	Document Object Model
DTD	Document Type Definition
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
LMS	Learning Management System
LOM	Learning Objects Metadata
LTSC	Learning Technology Standards Committee
RTS	Run-Time Service
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
URI	Universal Resource Identifier
URL	Universal Resource Locator
URN	Universal Resource Name
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

This page intentionally left blank.

APPENDIX B
REFERENCES

This page intentionally left blank.

References

1. IEEE P1484.11.1 Draft 1/WD 13 Draft Standard for Learning Technology – Data Model for Content Object Communication. July 1, 2003.
   Available at: http://ltsc.ieee.org/
2. IEEE P1484.11.2 Draft 4 Standard for Learning Technology – ECMAScript Application Programming Interface for Content to Runtime Services Communication. July XX,2003
   Available at: http://ltsc.ieee.org/
3. IETF RFC 2141: 1997, URN Syntax.
   Available at: http://www.ietf.org/
4. ISO/IEC 646:1991, Information technology – ISO 7-bit coded character set for information interchange.
5. ISO/IEC 10646-1:2000, Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane.
6. IETF RFC 2396:1998, Universal Resource Identifiers (URI): Generic Syntax.
   Available at: http://www.ietf.org/
7. AICC/CMI001 Guidelines for Interoperability Version 3.5. April 02, 2001
   Available at: http://www.aicc.org/
8. W3C, Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Working Draft 26 February 2003. (See: http://www.w3.org/DOM/)
9. ISO/IEC 16262:1998, Information technology—ECMAScript language specification
10. The SCORM Version 1.3 Overview
    Available at: http://www.adlnet.org/
11. The SCORM Version 1.3 Sequencing and Navigation
    Available at: http://www.adlnet.org/
12. ISO/IEC 11404:1996, Information technology – Programming languages, their environments and system software interfaces – Language-independent datatypes
13. The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: *The Unicode Standard, Version 4.0* (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1)

APPENDIX C
DOCUMENT REVISION HISTORY

This page intentionally left blank.

Document Revision History

SCORM Version	Release Date	Description of Change
1.3 Working Draft 1	20-Oct-2003	Initial Draft • Changes due to IEEE P1484.11.1 Draft 1/WD 13 Draft Standard for Learning Technology – Data Model for Content Object Communication • Changes due to IEEE P1484.11.2 Draft 4 Standard for Learning Technology – ECMAScript Application Programming Interface for Content to Runtime Services Communication • Changes reflecting IMS Simple Sequencing Version 1.0 • Changes reflecting IMS Content Packaging Version 1.1.3