



INSTITUTE FOR DEFENSE ANALYSES

**Sharable Courseware Object
Reference Model (SCORM)
Version 1.0**

Philip Dodds, Editor

July 2000

Approved for public release;
distribution unlimited.

IDA Document D-2461

Log: H 00-001936

This work was conducted under contract DASW01 98 C 0067, Task AK-2-1800, for the Office of the Under Secretary of Defense (Personnel and Readiness) [OUSD(P&R)]. The publication of this IDA document does not imply Department of Defense endorsement of factual accuracy or opinion, nor should the contents be construed as reflecting the official position of that Agency.

© 2000, 2003 Institute for Defense Analyses, 4850 Mark Center Drive, Alexandria, Virginia 22311-1882 • (703) 845-2000.

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (NOV 95).

INSTITUTE FOR DEFENSE ANALYSES

IDA Document D-2461

**Sharable Courseware Object
Reference Model (SCORM)
Version 1.0**

Philip Dodds, Editor

PREFACE

This work was performed by the Advanced Distributed Learning (ADL) Co-Laboratory under a task entitled “ADL Co-Laboratory.” It is an essential component of the ADL initiative being undertaken by the Department of Defense (DoD) in cooperation and coordination with the Office of Science Technology Policy (OSTP). This work was performed for the Under Secretary of Defense for Personnel and Readiness (USD(P&R)) and the Office of Naval Research (ONR).



Sharable Courseware Object Reference Model

SCORM

Version 1.0
January 31, 2000

ADL Course Structure (Editable)

- ADL Course Root
 - globalProperties
 - block: Creating Standards 101
 - block: Background
 - block: Standards Defined
 - block: Standards Groups
 - block: Process
 - au: Herding Kittens
 - block: Consensus Building
 - au: About Consensus Building
 - block: Identifying Stakeholders
 - block: Stakeholder Interests
 - au: Stakeholder Agendas
 - au: Stakeholder Contributions
 - au: Impediments to Progress
 - block: Determining Common Ground

Information Window

```

<title>Creating Standards 101</title>
<description>A short course on how to create standards</description>
<labels>
  <curricular>Course</curricular>
</labels>
</identification>
<completionReq>
  <![CDATA[A24]]>
</completionReq>
<block id="B1">
  <identification>
    <title>Background</title>
    <labels>
      <curricular>Unit</curricular>
    </labels>
  </identification>
  <au id="A1">
    <identification>

```

```

public class API extends Applet {

    public void init()
    { . . . }
    public String LMSInitialize(String param)
    { . . . }
    . . .
    public String LMSGetValue(String element)
    { . . . }
    public void LMSSetValue(String element,
                           String value)
    { . . . }
    public void LMSCcommit()
    { . . . }
    public String LMSGetLastError()
    { . . . }
    public String LMSGetErrorString( String
                                   errorCode)
    { . . . }
}

```

EXTENSION

- STRING *
 - EXTENSION
- LANGSTRINGTYPE
 - * LANGSTRING ~
 - ? EXTENSION
- RECORD ~
 - ? METAMETADATA
 - ? GENERAL
 - ? LIFECYCLE
 - ? TECHNICAL
 - ? EDUCATIONAL
 - ? RIGHTS
 - * RELATION
 - * ANNOTATION
 - * CLASSIFICATION
 - ? EXTENSION

www.adlnet.org

contact: secretariat@adlnet.org

Editor: Philip Dodds

Partial List of Contributors:

ADL

Ron Ball
Richard Burke
Dexter Fletcher
Alan Hoberney
Paul Jesukiewicz

AICC (www.aicc.org)

Jack Hyde
Bill McDonald
Anne Montgomery

IEEE (ltsc.ieee.org)

Mike Fore
Wayne Hodgins

IMS (www.imsproject.org)

Thor Anderson
Steve Griffin
Tom Wason

(At Large)

Chris Moffatt
Claude Ostyn
Chantal Paquin
Dan Rehak
Tom Rhodes
Tyde Richards
Roger St. Pierre

...and many others

ADL Sharable Courseware Object Reference Model

Version 1.0

January 31, 2000

1. Overview.....	11
1.1 Status of the Sharable Courseware Object Reference Model (SCORM)	11
1.2 Accelerating the Process	11
1.3 Specification Evolution and Example Code.....	12
2. Goal.....	13
2.1 Rationale.....	13
2.1.1 The Need for Competency.....	13
2.1.2 The Value of Tailored Instruction	13
2.1.3 The Effectiveness of Technology-Based Instruction	14
2.1.4 Promoting the Use of Technology-Based Instruction.....	15
2.2 The Need for a Reference Model.....	16
2.3 Reference Model Criteria	16
3. SCO Reference Model.....	19
3.1 Defining a “Learning Management System” (LMS)	19
3.2 Overview of SCO Reference Model (Narrative)	20
3.3 High-Level Requirements and SCORM Scope	21
3.4 Web-based Design Assumption	22
4. Definitions	23
4.1 Sharable Courseware Object (SCO).....	24
4.2 Sharable Courseware Object Reference Model (SCORM)	24
4.3 Course Structure Format (CSF) [1].....	24
4.3.1 External Course Metadata [1a].....	24
4.3.2 Assignment Hierarchy [1b]	24
4.3.3 Objectives [1c]	24
4.3.4 Assignment Hierarchy Metadata [1e].....	24
4.4 Content [2].....	24
4.4.1 Content Metadata [2a]	25
4.5 Raw Media [3]	25
4.5.1 Raw Media Metadata [3a]	25
4.6 Runtime Environment [4].....	25
4.6.1 Content Launch Protocol [4a]	25
4.6.2 Content Application Program Interface (API) [4b]	25
4.6.3 Content Data Model [4c]	25
5. XML CSF.....	27

5.1	Overview	27
5.2	Scope	27
5.3	Approach	27
5.4	Course “Packaging”	28
5.5	Course Interchange Overview	28
5.5.1	Source/Model/Location Structure.....	28
5.5.2	Course Sequencing Using Prerequisites and Completion Requirements	29
5.5.3	Unique IDs and ID References, and Aliases	30
5.5.4	Identification Model	31
	Title and Description.....	31
	Developer and Curricular Labels	32
5.5.5	CSF Extension Mechanism.....	32
5.6	Course Information—globalProperties	33
5.6.1	Global Properties— <i>externalMetadata</i>	33
5.6.2	Global Properties— <i>curricularTaxonomy</i>	33
5.6.3	Global Properties— <i>extensions</i>	34
5.7	Course Structure Hierarchy—block.....	34
5.7.1	Block— <i>externalMetadata</i>	36
5.7.2	Block— <i>objectivesRef</i>	36
5.7.3	Block— <i>identification</i>	36
5.7.4	Block— <i>prerequisites</i>	37
5.7.5	Block— <i>completionReq</i>	37
5.7.6	Block— <i>extensions</i>	37
5.7.7	Block— <i>au/block</i>	37
5.7.8	Block— <i>blockAlias</i>	38
5.8	Assignable Unit—au	38
5.8.1	Assignable Unit— <i>externalMetadata</i>	38
5.8.2	Assignable Unit— <i>objectivesRef</i>	38
5.8.3	Assignable Unit— <i>identification</i>	38
5.8.4	Assignable Unit— <i>prerequisites</i>	38
5.8.5	Assignable Unit— <i>completionReq</i>	39
5.8.6	Assignable Unit— <i>timeLimit</i>	40
5.8.7	Assignable Unit— <i>launch</i>	40
5.8.8	Assignable Unit— <i>masteryScore</i>	40
5.8.9	Assignable Unit— <i>extensions</i>	40
5.8.10	Assignable Unit— <i>auAlias</i>	40
5.9	Objectives	41
5.9.1	Objective— <i>externalMetadata</i>	41
5.9.2	Objective— <i>assignmentRef</i>	42
5.9.3	Objective— <i>identification</i>	42
5.9.4	Objective— <i>prerequisites</i>	42
5.9.5	Objective— <i>completionReq</i>	42
5.9.6	Objective— <i>extensions</i>	42
5.9.7	Objective— <i>objectiveAlias</i>	42
5.10	CSF Element Definitions/Descriptions.....	42
5.11	Examples.....	44
5.11.1	Simplest Example of CSF Record.....	44
5.11.2	Simple Example Pointing To Content.....	45
5.11.3	Example of Course With One Block	45
5.11.4	Example of Course With Blocks Within Blocks.....	45

5.11.5	Multi-Tiered Example (Seven Levels)	46
5.12	Conformance Testing.....	47
5.13	Sample Course Mappings.....	48
6.	Runtime Environment.....	49
6.1	Overview	49
6.2	SCORM and the AICC API Specification.....	50
6.3	API Adapter	50
6.4	Content Launch.....	52
6.5	Application Program Interface (API).....	53
6.6	Further Defining Content as “Assignable Units”	56
6.6.1	Content as a “Lesson”	56
6.6.2	Content as an “Assignable Unit”	56
6.6.3	SCORM “Assignable Unit” Definition	57
6.6.4	Defining a “Sharable Courseware Object” as an Assignable Unit	57
6.6.5	Small “Sharable Courseware Objects”	57
6.6.6	Content Sequencing Control.....	58
6.6.7	Toward Adaptive and Intelligent Tutoring.....	58
6.7	“CMI” Data Model	59
6.7.1	“CMI” LMS to Content (AU) Data Model	60
6.7.2	“CMI” Content (AU) to LMS Data Model	63
6.7.3	Student Data Collection	65
6.7.4	“CMI” Data Types and Controlled Vocabularies	67
6.8	Conformance Testing.....	68
7.	Metadata	69
7.1	Overview	69
7.2	Definitions of SCORM Metadata Elements	69
7.2.1	Raw Media Metadata	69
7.2.2	Content Metadata.....	70
7.2.3	External Course Metadata.....	70
7.2.4	SCO Structure Format (Assignment Hierarchy) Metadata	70
7.3	SCORM Metadata Mapping	71
7.4	Stand-Alone XML Metadata Records	85
7.5	XML Schema, Namespaces, and Extensibility	86
7.6	Conformance Testing.....	86
7.7	XML Examples.....	86
7.7.1	Empty Raw Media XML Metadata Record	86
7.7.2	Empty Content XML Metadata Record	89
7.7.3	External Course Metadata XML Record.....	92
8.	Sample SCORM Code	93
8.1	Sample LMS and Content	93
8.1.1	LMS Server	94
8.1.2	LMS Client.....	95

8.1.3	AICC CMI Data Model	97
8.1.4	Sample Course	98
8.1.5	Mapping Example Code to the SCORM.....	99
8.1.6	Structure of Sample LMS Application.....	99
8.1.7	Flow of Sample LMS Application.....	100
8.1.8	API Wrapper JavaScript Code Fragment	101
8.1.9	Course Structure Format XML Fragment	103
8.1.10	Course Metadata XML Example	104
8.1.11	Assignable Unit (Content) Metadata XML Example.....	106
8.1.12	Raw Media Metadata XML Example.....	108
8.2	CSF Browser/Editor.....	109
8.2.1	Overview of CSF Browser/Editor	109
9.	Acronym List	111
	Appendix A – Supporting Documents.....	113
A.1	SCORM Course Structure Format DTD.....	113
	Appendix B – AICC API Specification	119
	Appendix B: API-based CMI Communication.....	120
	Appendix C – IEEE Learning Object Metadata Draft 3.....	155
	IEEE Learning Technology Standards Committee (LTSC)	156
	Learning Object Metadata	156
	Working Draft Document 3 (approved 1999-11-27).....	156
	Contents	156
1	Background Information.....	157
1.1	Creation of this document	157
1.2	Disclaimers.....	157
1.3	Submitting comments and corrections	157
1.4	Revisions	157
1.5	Acknowledgements.....	158
2	Introduction	158
2.1	Scope	159
2.2	Purpose	159
3	Overview of the Metadata Structure.....	160
3.1	Basic metadata structure	160
3.2	Data elements	160
3.3	List values	161
3.4	Vocabularies.....	161
3.5	Minmax values	162
3.6	Character sets	162
3.7	Derived schemes	162
3.8	Indexation.....	163
3.9	Representation.....	163
4	Conformance.....	163
5	BaseScheme.....	164
6	LangStringType	180

7 DateType	181
8 VocabularyType.....	181
9 References to Other Standards	182
9.1 Complete Dublin Core Mapping.....	182
9.2 Miscellaneous	183
<i>Appendix D – IMS Learning Resource Metadata XML Binding Specification.....</i>	<i>185</i>
<i>Appendix E – Reference Material</i>	<i>207</i>
AICC Learning Model Definitions	207
Army Learning Structure Definitions	208
Air Force Shared Content Object Model	209
Marine Corps Learning Object Taxonomy	210
ADL Learning Taxonomy Mapping	211
<i>Appendix F – Document Change Summary.....</i>	<i>213</i>

1. Overview

The Department of Defense (DoD) established the Advanced Distributed Learning (ADL) Initiative to develop a DoD-wide strategy for using learning and information technologies to modernize education and training. To leverage existing practices, promote the use of technology-based learning, and provide a sound economic basis for investment, the ADL initiative has defined high-level requirements (e.g., content reusability, accessibility, durability, and interoperability) for learning content.

This document attempts to define a reference model for sharable courseware “objects”(SCOs) that meet ADL high-level requirements. It should be possible to map existing learning models and practices to this reference model so that common interfaces and data can be defined and standardized across courseware management systems and development tools.

1.1 Status of the Sharable Courseware Object Reference Model (SCORM)

The release of this document completes the initial drafting and review of early-stage Web-based learning specifications. With this release, the ADL Initiative’s SCORM enters a test and evaluation phase, during which researchers and early adopters are encouraged and expected to develop trial implementations based on these specifications. During this testing phase, corrections, clarifications, and improvements—based on the trial implementations—will be gathered and redistributed for review.

In parallel with the testing phase, which is expected to take 4-to-6 months, the ADL Collaboration Laboratory (ADL Co-Lab) plans to release example implementations, addenda to this document, and, at the end of the phase, a suite of conformance-test software. These products will permit content and tool developers to verify that their work products conform to the SCORM specification and are reusable, interoperable, accessible, and durable.

It is critically important for readers to understand that the specifications contained or referenced herein are largely untested in practice. Although example implementations supporting these specifications are available and several organizations have implemented portions of the SCORM, these specifications have not been deployed as a system or as a fully supported product.

1.2 Accelerating the Process

The purpose of the ADL Initiative, among other things, is to accelerate the development and adoption of technical specifications that promote sharable content and systems. Thus, this release is designed to encourage rapid trial implementations, incorporate the findings of those participating, and share results with as wide an audience as possible—all in “Internet time.”

We expect some organizations to adopt elements of the SCORM specifications immediately to take early advantage of the benefits of a common approach. These organizations will necessarily accept the risk that some aspects of the specifications may change as experience through implementations is gained. Of course, those who accept the risk early on stand the likelihood of being farther up the learning curve than those who wait, and early adopters can influence the resolution of ambiguities that may exist.

1.3 Specification Evolution and Example Code

The release of SCORM 1.0 indicates the belief that the specifications are as stable as they can be before being tested through trial implementations. The release of Version 1.0 includes a suite of examples that implement various parts of the specifications. These examples are provided, without cost or restriction, to accelerate more sophisticated implementations. Those who review or use the code examples are encouraged—but not required—to provide to the ADL Initiative feedback and/or other code examples that can be shared with others. In this way, the specifications will become more complete and accurate, and test-development software will become robust.

Through this process, the ADL Initiative will provide interim releases (e.g., Versions 1.1, 1.2) that clarify, amplify, and generally improve the usability of these specifications. When this process has stabilized in roughly 4 to 6 months and conformance software has been completed and tested by the ADL Co-Lab, a “final” version, probably “2.0,” will be released.

We hope and expect that the changes made during this test phase will not significantly change the intent or technical approach of Version 1.0. Changes that impede implementation or are determined to be outright oversights will be incorporated into Version 1.1. New topics that expand the overall scope of the SCORM will be added if they are judged to be appropriate and stable. Other, more comprehensive changes will be included in later versions of the SCORM.

2. Goal

A key ADL requirement for learning content is the ability to reuse instructional components in multiple applications and environments regardless of the tools used to create them. This requires, among other things, that content be separated from context-specific runtime constraints so that it can be incorporated into other applications. For reuse to be possible, content must also have common interfaces and data. This document attempts to specify a reference model that abstracts runtime constraints and defines a common interface and data scheme for reusable content.

2.1 Rationale

2.1.1 The Need for Competency

Government, industry, and academia are experiencing a revolution of unprecedented proportions in science and technology. This revolution and the advances it presents pose significant challenges and opportunities. Organizations must adopt these advances and leverage them if they are to compete successfully in the 21st century; however, infusing technology in routine operations increases the demand for people who can deploy, operate, and maintain it competently. Despite the increasing presence of technology, competent human performance remains as essential as ever, and its ready availability is a matter of the first importance in all sectors of the economy.

Fortunately, technology also provides the means to meet the challenges it presents. As new instructional technologies emerge, they provide opportunities for universally accessible and effective life-long learning. These technologies extend learning beyond the confines of traditional classrooms to encompass homes, workplaces, and community resources, such as museums and libraries. They extend beyond the traditional school-age population to support a nation of life-long learners.

These considerations have led to a vision that guides the work of the ADL Initiative:

To ensure all Americans access, anytime and anyplace, to high-quality education and training tailored to their individual learning and workplace needs.

2.1.2 The Value of Tailored Instruction

Empirical studies have raised national interest in employing education and training technologies that are based on the increasing power, accessibility, and affordability of computer and networking technologies. These studies suggest that the promise of training technologies [e.g., computer-based instruction (CBI), interactive multimedia instruction, and intelligent tutoring systems] keys on their ability to tailor instruction to the needs of individuals. In contrast to classroom learning, these approaches enable the pace, sequence, content, and method of instruction to better fit each student's learning style, objectives, and goals.

These technologies allow individually tailored training to be delivered anytime, anywhere, and to anyone who needs it. Such accessibility pays off for the individual who wishes to advance knowledge, skill, and career opportunities and for the organization that depends on the individual's growing competencies to compete successfully in the global marketplace.

Research has supported the intuitive appeal of technology-based instruction. The speed with which individuals can progress through instruction has been found to vary by factors of three to five (or even as much as seven), even in classes of carefully selected students.¹ On average, a student in classroom instruction asks about 0.1 questions an hour.² In individual tutoring, students may ask or be required to answer as many as 120 questions an hour. The achievement of individually tutored students may exceed that of classroom students by as much as two standard deviations—an improvement roughly equivalent to raising the performance of 50th percentile students to that of 98th percentile students.³

The dilemma presented by individually tailored instruction is that it combines an instructional imperative with an economic impossibility. With few exceptions, one instructor for every student—despite its advantages—is not affordable. The promise of instructional technology is that it can provide most of the advantages of individualized instruction at affordable cost while maintaining consistent, measurable, high-quality content.

2.1.3 The Effectiveness of Technology-Based Instruction

Studies have shown that technology-based instruction may significantly reduce the costs (by 30–60 percent) of achieving a wide range of instructional objectives. These studies have also found that it either reduces the time to achieve given instructional objectives by about 30 percent or that it increases student skills and knowledge by about 30 percent, depending on whether achievement or time is held constant.

The value of these capabilities in reducing direct training costs is obvious. The savings accrued through improved management of indirect costs, such as productivity and time away from a job site, are more difficult to quantify and capture but are equally significant when determining the full return on investments in instructional technologies.

For instance, reducing by 30 percent the time to train just 40 percent of all DoD students in specialized skill training—which excludes other categories such as recruit training, pilot training, unit training, and field exercises—could potentially save the DoD over \$500 million annually.

¹ Gettinger, M. (1984). Individual differences in time needed for learning: A review of the literature. *Educational Psychologist*, 19, 15–29.

² Graesser, A. C., & Person, N. K. (1994). Question asking during tutoring. *American Educational Research Journal*, 31, 104–137.

³ Bloom, B.S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13, 4–16.

Given these potential cost savings, a reasonable question to ask is whether training effectiveness is being lost to achieve them. Figure 2-1 shows results aggregated from empirical comparisons of technology-based training with conventional classroom instruction. As this figure shows, 233 such studies of conventional CBI averaged an improvement in learning of about 0.39 standard deviations. Adding multimedia capabilities also adds effectiveness, raising the improvement to 0.50 standard deviations. Intelligent tutoring systems intended to emulate more directly one teacher interacting with one student and allowing either the student or the computer to ask questions increases the improvement to 0.84 standard deviations. Some recent assessments of intelligent tutoring systems yielded improvements averaging about 1.05 standard deviations. We have not yet met the 2.00 standard-deviation challenge, but the trends are promising.

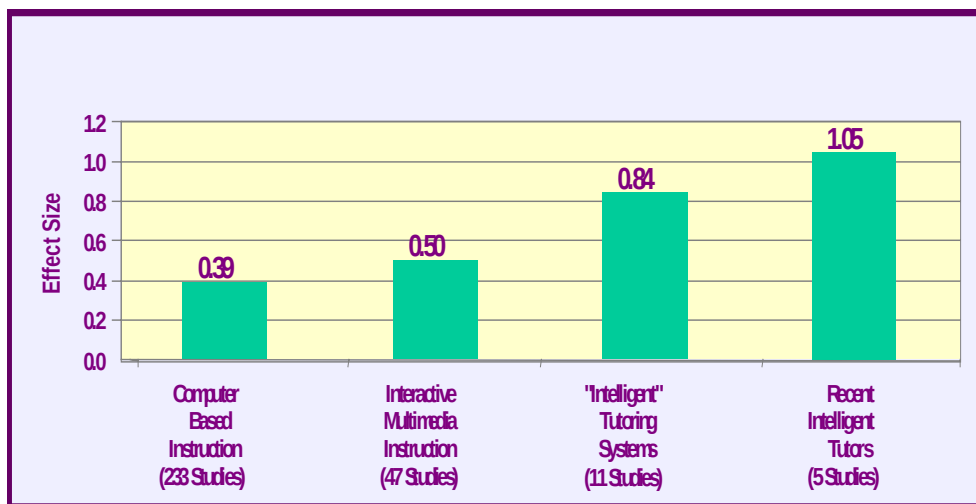


Figure 2-1. Some Effect Sizes for Technology-Based Instruction

2.1.4 Promoting the Use of Technology-Based Instruction

There is, then, evidence that technology-based instruction can lower training costs and, at the same time, increase instructional effectiveness for a variety of training objectives and programs. Yet, its use is only beginning. For instance, data collected suggest that less than 5 percent of DoD training programs routinely use interactive training technologies. Technology insertion, as is often the case with new applications, may depend on issues that are more structural and organizational than technological. Accounting categories, local incentives, personnel policies, and training procedures must be changed to make the best use of these new training capabilities.

Despite these difficulties, the benefits of technology-based instruction are increasingly recognized, and initiatives are being undertaken to increase its use. Primary among these is the ADL Initiative. The aim of this initiative is to increase the efficiency of investments in technology-based instruction through the development of Web-available, “sharable courseware objects,” or SCOs. These SCOs would be reusable in the

development of technology-based instruction, portable across different presentation platforms, accessible through the use of metadata standards for identifying and locating them, and durable across different versions of operating systems, browsers, and other supporting systems software.

This DoD initiative is being undertaken in cooperation with the Military Departments and the Office of Science and Technology Policy (OSTP), which plans to extend the ADL approach across all Federal training programs. The private sector involved in producing technology-based instructional systems and solutions and academia are also expected to adopt the ADL approach.

2.2 The Need for a Reference Model

Successful implementation of the ADL initiative will require guidelines that are shared and observed by organizations that have a stake in the development and use of instructional technology materials. The ultimate form and status of these guidelines remain to be determined. They may be international or national standards, agreed upon practices, recommendations, or de facto practices.

If these guidelines are to be articulated and implemented successfully, they must be based on a common “reference model.” This model will not replace the detailed models of instructional system design or practice that have been devised and adopted by specific organizations, such as those of instructional developers, instructional tool developers, or customers associated with particular industries or the Armed Forces. Instead, the purpose of the reference model is to describe an approach for developing instructional material in sufficient detail to permit guidelines for the production of SCOs to be articulated clearly and implemented.

2.3 Reference Model Criteria

There are three primary criteria for such an SCO reference model. First, as stated previously, the criteria must fully support articulation of guidelines that can be understood and implemented for the production of SCOs. Second, the criteria must be adopted, understood, and used as much as possible by a wide variety of stakeholders, such as courseware and courseware tool developers and their customers. Third, the criteria must permit mapping of any stakeholder’s specific model for instructional systems design and development into itself. Stakeholders must be able to see how the reference model they hold in common reflects their own model of instructional system design.

Applications of information technology have been shown to increase the effectiveness and the efficiency of training; however, up-front investment is required to develop and convert training materials for technology-based presentation. These investment costs can be reduced by an estimated 50–80 percent through the use of SCOs that are:

- Durable—do not require modification as versions of system software change
- Interoperable—operate across a wide variety of hardware, operating systems, and Web browsers

- Accessible—can be indexed and found as needed
- Reusable—can be modified and used by many different development tools.

Procedures for developing such courseware objects are within the state-of-the-art, but they must be articulated, accepted, and widely used as guidelines by developers and their customers. These goals can only be achieved through collaborative development. Collaboration will also increase the number, quality, and per unit value of courseware objects made available. Such collaboration requires agreement upon a common reference model.

3. SCO Reference Model

This section provides a high-level overview of the scope and purpose of the SCORM. Subsequent sections define technical details for implementing each aspect of the model.

3.1 Defining a “Learning Management System” (LMS)

LMS is used as a catch-all term throughout this document to refer to a suite of functionalities designed to deliver, track, report on, and administer learning content, student progress, and student interactions. The term LMS can apply to very simple course management systems or highly complex, enterprise-wide distributed environments. Figure 3-1 shows a broad definition of LMS as a suite of server-side functionalities that controls the delivery and tracking of learning content to a client-side student.

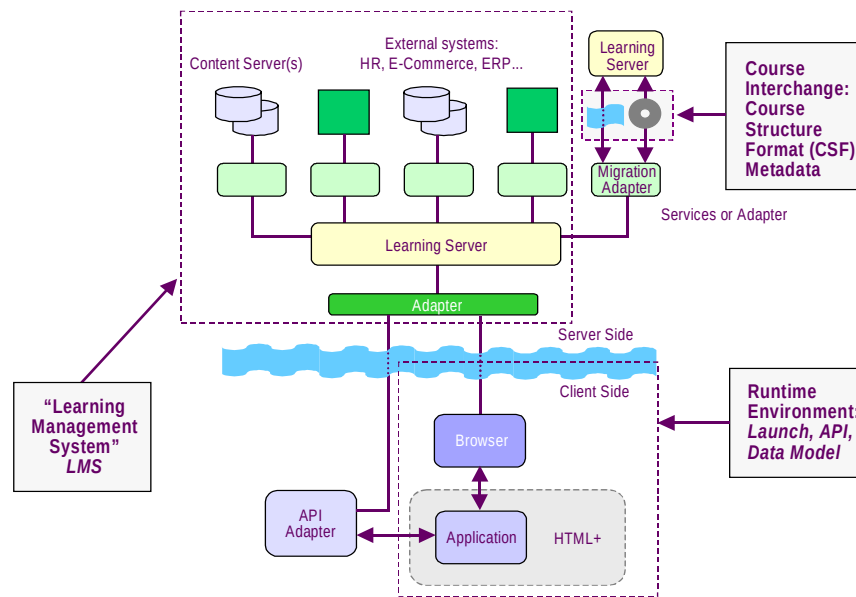


Figure 3.1. An LMS

Note for Figure 3-1: The SCORM does not specify functionality within the LMS. Only Course Interchange, Metadata, and Runtime Environment are “in scope” for this version of the SCORM.

Many participants in the development of learning technology standards now use the term LMS instead of “Computer-Managed Instruction” (CMI). In this way, they can include new functionalities and capabilities that have not historically been associated with CMI systems, such as back-end connections to other information systems, complex tracking and reporting, centralized registration, online collaboration, and adaptive content delivery.

The term LMS is now being used as a “superset” description of many possible capabilities. Within the SCORM context, implementations are expected to vary widely. SCORM focuses on key interface points between content and LMS environments and is silent about the capabilities provided within a particular LMS.

Within the SCORM context, the term LMS implies a server-based environment in which the intelligence resides to control the delivery of learning content to students. In other words, in the SCORM, the LMS has the “smarts” about what to deliver and when, and tracks student progress through the learning content.

Learning content, therefore, does not play a “management” role in the SCORM because that function falls entirely within the LMS. That means that SCORM content does not determine (on its own on the client-side) how to navigate through a course or when a student has completed a section of the course. That is the LMS’s job. This approach frees content from course-specific constraints and permits the development of content that is reusable, sharable, and as context independent as possible.

3.2 Overview of SCO Reference Model (Narrative)

The SCORM defines a Web-based learning “content model.” At its simplest, it is a set of interrelated specifications designed to meet DoD’s high-level requirements for Web-based learning content: reusability, accessibility, durability, and interoperability.

The work of the ADL Initiative in developing the SCORM is also a process of knitting together disparate groups and interests. We hope that this reference model will serve as a bridge from general emerging technologies to commercial implementations.

Several organizations have been working on different, yet highly related aspects of Web-based learning technology. These work areas have coalesced into three major topics: metadata, runtime environment, and course interchange. Although these evolving areas have made great strides recently, they have not yet been “connected” to one another in a meaningful way. In some cases, emerging specifications are quite general, anticipating a wide variety of implementations by various user communities (e.g., metadata). In other cases, the specifications are rooted in earlier CMI practices and require adaptation to Web-based applications.

The SCORM’s purpose is to apply current technology developments to a specific content model and to produce recommendations for consistent implementations by the vendor community. These technology developments come from groups such as the Instructional Management Systems (IMS) Project, the Aviation Industry CBT Committee (AICC), and the Institute of Electrical and Electronics Engineers (IEEE) Learning Technology Standards Committee (LTSC).

The scope of the SCORM is not all inclusive. A host of issues are not addressed by this version of the document. We expect that the scope will be enlarged over time and the reference model will be expanded as experience is gained through implementation and deployment.

This version of the SCORM comprises three major elements:

1. **Course Structure Format.** An Extensible Markup Language (XML)-based representation of a course structure that can be used to define all the course elements, structure, and external references necessary to move a course from one LMS environment to another (see Section 5 of this document).
2. **Runtime Environment.** A definition of runtime environment that includes a specific launch protocol to initiate executable Web-based content, a common content-to-LMS application program interface (API), and a data model defining the data exchanged between an LMS environment and executable content at runtime (see Section 6 of this document).
3. **Metadata.** A mapping and recommended usage of IEEE LTSC metadata elements for each of the following SCORM categories (see Section 7 of this document):
 - a. *Raw media metadata.* A definition of metadata that can be applied to so-called “raw media” assets, such as illustrations, documents, or media streams, that provide descriptive information about the raw media—independent of courseware content. These metadata are used to facilitate reuse and discoverability principally during content creation of such media elements within, for example, a media repository.
 - b. *Content metadata.* A definition of metadata that can be applied to Web-based content “chunks” that provide descriptive information about the content—independent of a particular course. These metadata are used to facilitate reuse and discoverability of such content within, for example, a content repository.
 - c. *External course metadata.* A definition for external metadata that describes a course package for searching (enabling discoverability) within a courseware repository and providing descriptive information about the course.

3.3 High-Level Requirements and SCORM Scope

This SCORM document frequently references the following high-level ADL requirements throughout. The definitions below describe the capabilities that the SCORM expects to enable:

- **Accessibility.** The ability to access instructional components from one remote location and deliver them to many other locations
- **Interoperability.** The ability to use instructional components developed in one location and with one set of tools or platform in another location and with a different set of tools or platform. (**Note:** *There are multiple levels of interoperability.*)
- **Durability.** Instructional components that do not require redesign or recoding to operate when base technology changes
- **Reusability.** The design of instructional components so that they can be incorporated into multiple applications.

These definitions can be restated as:

- The ability of a Web-based LMS to launch “executable” content that is authored using tools from different vendors and to exchange data with that content
- The ability of Web-based LMS products from different vendors to launch the same executable content and exchange data with that content during execution
- The ability of multiple Web-based LMS products/environments to access a common repository of executable content and to launch such content.

During the initial implementation and testing phases, these requirement statements will be used as evaluation criteria.

3.4 Web-based Design Assumption

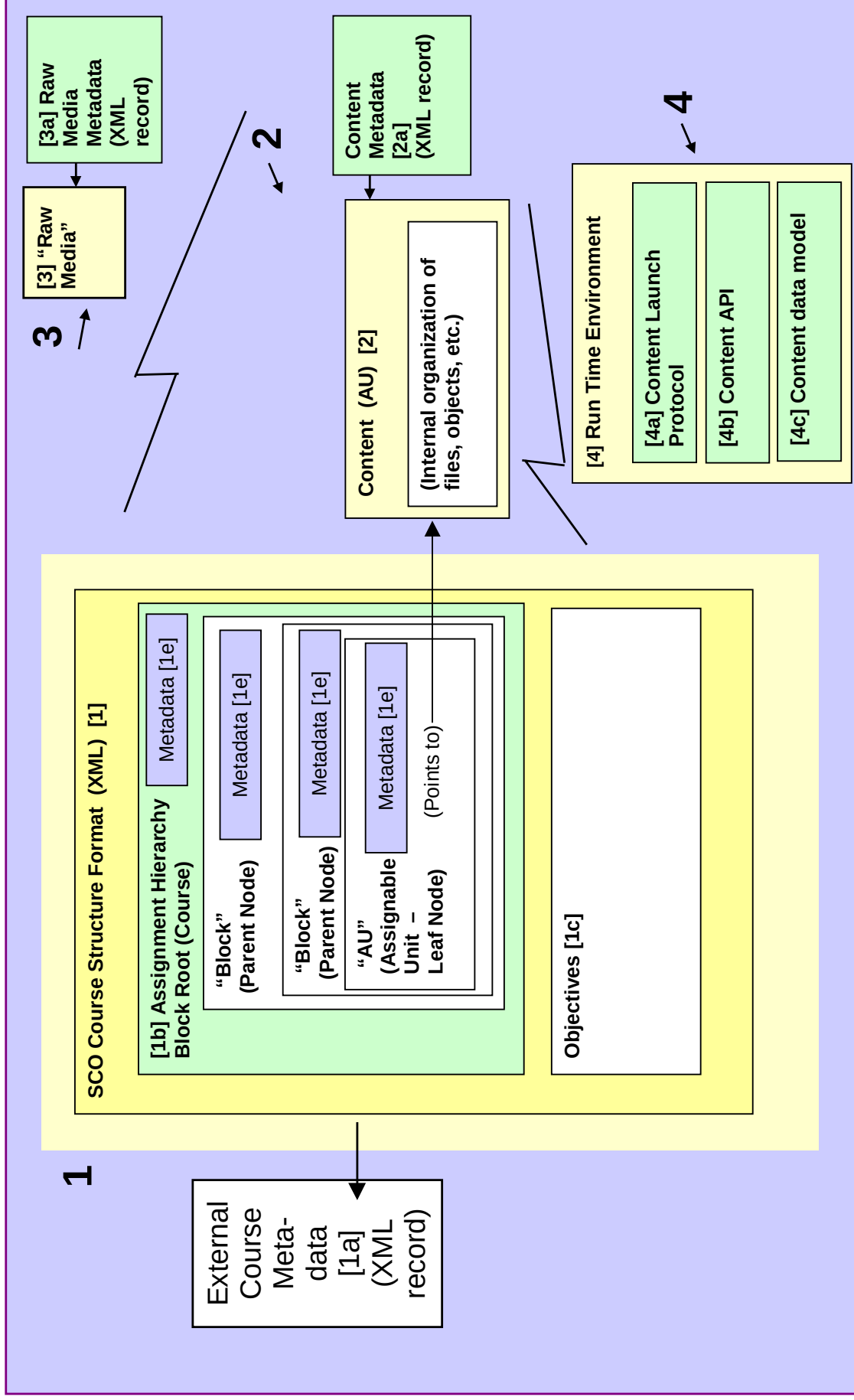
SCORM assumes an Internet, Web-based infrastructure as a basis for its technical implementation. This assumption was made for several reasons:

- Web/Internet technologies and infrastructure are rapidly expanding and provide a mainstream basis for learning technologies.
- Web-based learning technologies standards do not yet exist.
- Web-based content can be delivered using nearly any type of medium (e.g., CD-ROM, stand-alone systems, and/or as-networked environments).

This approach embraces the mainstream transition to common content and delivery formats that is occurring in industry. Computer operating system environments now natively support Web content formats such as Hypertext Markup Language (HTML) and the Joint Photographic Experts Group (JPEG). The trend is toward the use of common content formats that can be used locally, on local Intranets, or over the Internet. The SCORM extends this trend to learning technologies.

4. Definitions

Figure 4-1. ADL SCO Reference Model Diagram



4.1 Sharable Courseware Object (SCO)

An SCO is an interoperable, durable, computer-based course or component of a course packaged with sufficient information to be reusable and accessible.

4.2 Sharable Courseware Object Reference Model (SCORM)

SCORM is a software model that defines the interrelationship of course components, data models, and protocols so that courseware “objects” are sharable across systems that conform with the same model.

4.3 Course Structure Format (CSF) [1]

A CSF defines all the course elements, the course structure, and all external references necessary to represent a course and its intended behavior.

4.3.1 External Course Metadata [1a]

External course metadata is information that can be searched externally, such as the course title, course description, and version.

4.3.2 Assignment Hierarchy [1b]

The assignment hierarchy is a tree structure that defines a hierarchical lesson plan for a course. The ordering of the tree elements defines a default sequence for executing each of the assignments in the course.

4.3.3 Objectives [1c]

An objective is a statement of skills, knowledge, and attitudes to be acquired by the student.

4.3.4 Assignment Hierarchy Metadata [1e]

Assignment hierarchy metadata are metadata that are described with the specific assignments at different levels within the lesson plan hierarchy (e.g., course element metadata within a particular course hierarchy that is context specific to that course hierarchy).

4.4 Content [2]

This is content that runs on a client (i.e., executed within a client-side browser).

4.4.1 Content Metadata [2a]

Content metadata are metadata that describe a [sharable] “chunk” of content. Content metadata are not related to a specific course structure (i.e., context-independent metadata). This is information can be searched externally (e.g., content asset title, description, and version).

4.5 Raw Media [3]

Raw media are media assets (e.g., images, sounds, text, or other presentation documents) that can be incorporated into executable assets (content) during authoring or dynamically at runtime. Media assets have metadata but are not expected to be used in a standalone mode (i.e., outside of content).

4.5.1 Raw Media Metadata [3a]

Raw media metadata are metadata that describe raw media elements in a noncontext-specific manner. This is information that can be searched externally (e.g., media asset title, description, date of creation, and version) and that can be used to create a searchable repository of sharable media elements.

4.6 Runtime Environment [4]

The runtime environment is defined mechanisms for starting (launching) executable content and exchanging data between an LMS and the content.

4.6.1 Content Launch Protocol [4a]

Content launch protocol is the protocol used to launch the executable content and connect it to the API provided by an LMS.

4.6.2 Content Application Program Interface (API) [4b]

The content uses content API to communicate with an LMS

4.6.3 Content Data Model [4c]

The content data model is the definition of the data exchanged between an LMS and the content launched under control of such a system:

- The LMS makes student data available to the content.
- The content passes learner performance data and other tracking information back to the LMS.

5. XML CSF

5.1 Overview

The purpose of the XML CSF is to provide a means for moving a course from one LMS to another. A CSF defines all the course elements, the course structure, and all external references necessary to represent a course and its intended behavior.

This CSF is intended to promote reuse of entire courses and encourage the reuse of course components by exposing all the details of each course element. It is intended to reduce or eliminate the dependency of a course on a particular LMS implementation.

This CSF was co-developed by several organizations, including ADL, AICC, IEEE, and IMS. Many thanks go to Jack Hyde and Bill McDonald for providing the basis of course structure in AICC's CMI specifications (and supporting this effort); Tyde Richards, who constructed the very first XML versions of this structure; Thor Anderson (IMS) for his work in harmonizing the CSF with the IMS metadata efforts; Dan Rehak for keeping us honest; and many others who continue to contribute to this effort on an ongoing basis.

5.2 Scope

This CSF is (only) an intermediate format for representing Web-based courses that are being moved from one LMS to another. It does not define LMS functionality. It is assumed that an LMS may have a private, unique representation for course elements and structure and that the LMS can “export” a CSF file or record that can then be “imported” by another LMS and stored in its local form. The CSF is not intended to require LMS systems to adopt the CSF model or structure internally.

5.3 Approach

The CSF is intended to represent a wide variety of course structures and content “aggregations.” The CSF can represent content structures that range from very small “chunks” of content—as simple as a few lines of HTML or short media clip—to highly interactive learning content that is tracked by an LMS. The CSF is neutral about the complexity of content, the number of hierarchical levels of a particular course (i.e., “granularity”), and the instructional methodology employed to design a course.

The XML CSF is derived from the AICC content model for course structure, properties, and objectives. This model was chosen as a starting point because key components of course representation are defined in the AICC's Semantic Document v3.0 (CMI-Sem30.doc). One objective of this version of the CSF is to map the course structure, properties, and objectives in the AICC-defined tables into an XML format for Web applications. Another objective is to extend the CSF to include additional features, such as referencing external IMS/IEEE metadata records. Thus, this CSF extends the AICC CMI practice to include new capabilities for Web-based content.

5.4 Course “Packaging”

The CSF should not be confused with so called “course packaging.” Packaging is the process of identifying all course files, regardless of type, and then physically bundling all these components together with a manifest (packing slip) for movement from one environment to another. The CSF is simply one of the “files” (albeit a very important one) needed to move a course physically from one place to another. Actual content, metadata records, and raw media must also be “packaged” with a CSF when a course is moved from one place to another. The CSF is, therefore, a “blueprint” for assembling all of the constituent pieces once a course has arrived at its destination.

5.5 Course Interchange Overview

The CSF describes a course using three groups of information (see Figure 5-1). The first group, called *globalProperties*, is the data about the overall course. The second group, called *block*, defines the structure of the course. The third group, *objectives*, defines a separate structure for learning objectives with references to course elements within the assignment structure.

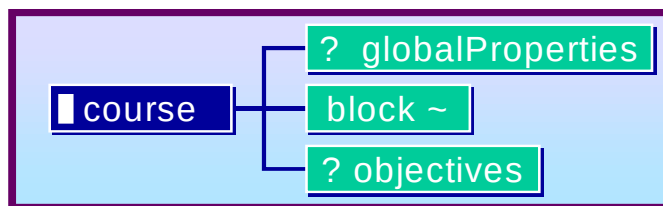


Figure 5-1. CSF High-Level XML Document Type Definition (DTD) Structure

Note for Figure 5-1: No notation = one element required; “?” = zero or one (optional).

5.5.1 Source/Model/Location Structure

Several CSF elements use a common substructure to define externally referenced information or practices. The idea is to provide a way to determine which organizational standards, technical specifications, or other relevant information are helpful to understanding the intent of the course. Although the usage will vary slightly from one element to another, the general intent is that subelement:

- **Source.** Describes the source or originator of a given practice or specification to which this course adheres. Examples could include ADL CSF, AICC CMI, IEEE Learning Object Metadata (LOM), ARMY LEARNING STRUCTURE, and so forth.
- **Model.** Describes a specific data model, defining structure, or specification to which this course adheres. Examples could include ADL SCORM 1.0, AICC CMI 3.0.1, IEEE LOM 3.1, and so forth.

- **Location.** Describes where the controlling specification or referenced material can be found.

5.5.2 Course Sequencing Using Prerequisites and Completion Requirements

The CSF elements *block*, *au*, and *objective* each have subelements called *prerequisites* and *completionReq* (completion requirements). These elements provide a field that can be used to represent algorithmically the sequence of events through a course. These elements mirror certain tracked data elements in the data model described in Section 6 of this document. The data model provides a means for content to report to an LMS when a particular part of a course is “complete” or “incomplete.” An LMS can then evaluate the statements in *prerequisites* and *completionReq* to determine what the student should be delivered next.

The *prerequisites* element defines what other parts of the course must be completed before starting the parent *block*, *au*, or *objective*. Similarly, *completionReq* defines what other course parts must be completed to consider the parent “done.” This allows an LMS to compute multiple paths through a course.

The AICC’s Semantic Document v3.0 (CMI-Sem30.doc) (Section 6) describes in great detail the use of prerequisites and completion requirements. Table 5-1, which is extracted from the AICC document, describes the logic encoding used for these two elements.

Table 5-1. Logic Encoding Used for the *prerequisites* and *completionReq* Elements

And	All elements separated by an & must be complete for the expression to be evaluated as complete. A34 & A36 & A38 Assignable units numbers 34, 36, and 38 must all be complete (Passed or Completed) for the group to be considered complete.
Or	If any of the elements separated by an are passed, the expression is considered true. A34=P A36=P A38=P If any one of the lessons, 34, 36, or 38, are passed, the group is considered complete.
Not	An operator that returns incomplete (false) if the following element or expression is complete, and returns complete (true) if the following element or expression is incomplete (false). Element Identifier: A34 Requirement: ~A35 The student may enter unit A34 as long as unit A35 has not been completed (that is, the status of A35 must be Incomplete, Failed, or Not attempted). If assignable unit A35 is complete, the student may not enter unit A34.
Equals	An operator that returns true when representations on both sides of the symbol have the same values. Element Identifier: A34 Requirement: A33=Passed The student may enter unit A34 if he or she has passed unit A33.

Table 5-1. Logic Encoding Used for the *prerequisites* and *completionReq* Elements (Continued)

Not equals	An operator that returns true when elements on both sides of the symbol have different values. Element Identifier: A34 Requirement: A35<>Passed The student may enter unit A34 as long as he or she has not passed A35. Notice the difference between this expression and the example for the not operator. The equivalent of ~A35 is (A35<>Passed & A35<>Completed)
Set	A list of course elements separated by commas and surrounded by curly brackets -- { }. A set differs from a block, in that the set is defined only for purposes of the prerequisite file. A set has no effect on the structure of the course. {A34, A36, A37, A39} Assignable units A34, A36, A37, and A39 are part of a set.
Separator	The comma is used to separate the members of a set. Each member of the set can be evaluated as a Boolean element – complete or incomplete. {A34, A36, A37, A39} Assignable units A34, A36, A37, and A39 are each separated by a comma in this set.
X*	X is an integer number. This operator means that X or more members of the set that follows must be complete for the expression to be complete (true). Element Identifier: A38 Requirement: 3*{A34, A36, A37, A39} Any three or more of the following units – 34, 36, 37, 39 -- must be complete before the student can enter unit 38.
Evaluate 1st	The expression within the parenthesis () must be evaluated before combining its results with other parts of the logical statement. Parentheses may be nested.⁴ Element Identifier: A39 Requirement: A34 & A35 A36 In this statement, completing A36 all by itself enables the student to enter A39. Element Identifier: A39 Requirement: A34 & (A35 A36) Adding the parenthesis makes it necessary to complete at least two units (A36 all by itself is no longer sufficient) to enter unit A39.

The *prerequisites* and *completionReq* elements require that their values be of XML type CDATA to preserve all of the characters within a sequencing expression. A “type” attribute is also provided to identify the type of algorithmic script being used in the CDATA field. In the following example, the aicc_script “language” defined above is in use, and blocks B1, B2, and assignable unit A1 must be completed before the parent may be entered:

```
<prerequisites type="aicc_script"> <![CDATA[B1&B2&A1]]>
</prerequisites>
```

5.5.3 Unique IDs and ID References, and Aliases

Three of the CSF elements—*block*, *au*, and *objective*—use the XML “ID” and “IDRef” attributes to identify uniquely other elements within the CSF. These three elements are candidate targets for reference elsewhere within the CSF.

⁴Operator precedence is the same as in the C programming language, including the use of parenthesis.

XML requires that these attribute values begin with a letter and may otherwise be composed of letters, digits, hyphens, underscores, and full-stop characters. XML also requires the attribute value to be unique to the XML document (which fits the usage in this course representation). ID attributes may not have fixed default values, and only one attribute per element may be of type ID. It is assumed that the assignment of ID values will be automated within tools and LMS environments to ensure uniqueness.

SCORM has adopted the following usage convention:

Element Type	ID value
Block	B+int
Assignable Unit (AU)	A+int
Objective	O+int

Note: This convention closely follows prior AICC practices except that “O” is used for Objective instead of “J” for “Objective” IDs. This change was made for clarity.

ID References (IDREF) are essentially “pointers” to specific CSF elements that have IDs and must match one of the IDs in the CSF record.

Note: Provision has been made for a “relation” attribute for ID References. This is in recognition of the need to define the nature of a reference for proper interpretation by an LMS. An objective, for example, might have a relation that is “satisfied by” the completion of a block reference. Because no relations models have been defined at this time, however, this attribute is reserved for future use.

Similar to IDs, there are three “aliases” in the content hierarchy: *blockAlias*, *AuAlias*, and *ObjectiveAlias*. These were provided as a shorthand way to provide a second or third (or more) reference to the same course element without having to restate the same information in-line. Similar to ID References, an alias must match an ID within the same XML record. By convention, it is expected that aliases will not be used in a CSF record until after the referenced element has been defined (to avoid the need for forward referencing).

5.5.4 Identification Model

Title and Description

The same Identification subtree structure is used under *blocks*, *aus*, and *objectives*. The most important elements of this model are *title* and *description*. Note that the identification-title is a minimum required element throughout the CSF. The description is optional.

At first glance, *title* and *description* look similar to the same kind of metadata discussed in Section 7 of this document. This is not exactly the case, although the contents of these elements could contain the same or similar information to that found in a stand-alone metadata record. Within the CSF, these elements are meant to store the title and description within the context of the course. A reusable learning object, for example,

might have a generic name in a separate XML record used principally to identify it. Within a course, however, the course designer may want to rename that object to something more meaningful in the context of a particular course.

Similarly, a description for a reusable content object might describe what the object does (e.g., “Provides an introduction to XML for novices”). Within the CSF, however, the description might describe what the object is within a specific course (e.g., “Chapter 3—First Introduction to XML Structures”).

Developer and Curricular Labels

Identification also provides a *labels* element that can be used to store information that might be useful to course developers. *Developer* may be used to store a tag or label useful to the developer that might be in use by convention within an organization or as a by-product to the use of a tool. These elements allow such information to be contextualized and carried along with course when it is moved.

The *curricular* label is also “informative” and is intended to describe the name (label) of the element according to local practices. This label could be used to identify names such as “Course,” “Unit,” “Lesson,” “Module,” “Learning Step,” and so forth. These terms are expected to be derived from a known model using a known vocabulary that should be defined under *globalProperties* using the *curricularTaxonomy* element.

The *labels* subtree is intended to capture valuable and important information about a course and its construction; however, these elements are considered “informative” and are not expected to affect how content is actually delivered.

5.5.5 CSF Extension Mechanism

Throughout the CSF, provisions have been made for extensions. The *extension* element includes information about the source of the extension, its model, the location for the defining specification, and name/value pairs of extended elements.

This extension mechanism was included with the realization that no course structure model could completely anticipate the needs of all users; however, the use of extensions comes with great risk. Extensions unilaterally implemented by one LMS could be, at best, meaningless to another LMS and, at worst, result in the course not behaving as intended or perhaps not operating at all when moved.

Therefore, it is recommended that extensions be implemented with care and that appropriate documentation and organizational policy be established to manage community-specific additions.

It is also expected that conventions for extensions and provisions for name spaces associated with subcommunities will evolve to ease the risk of local customization. Future versions of this document are expected to provide guidance in this area.

5.6 Course Information—*globalProperties*

The *globalProperties* node of the CSF contains or references information about the course as a whole. It also provides information describing the general approach used during the design of the course. See Figure 5-2.

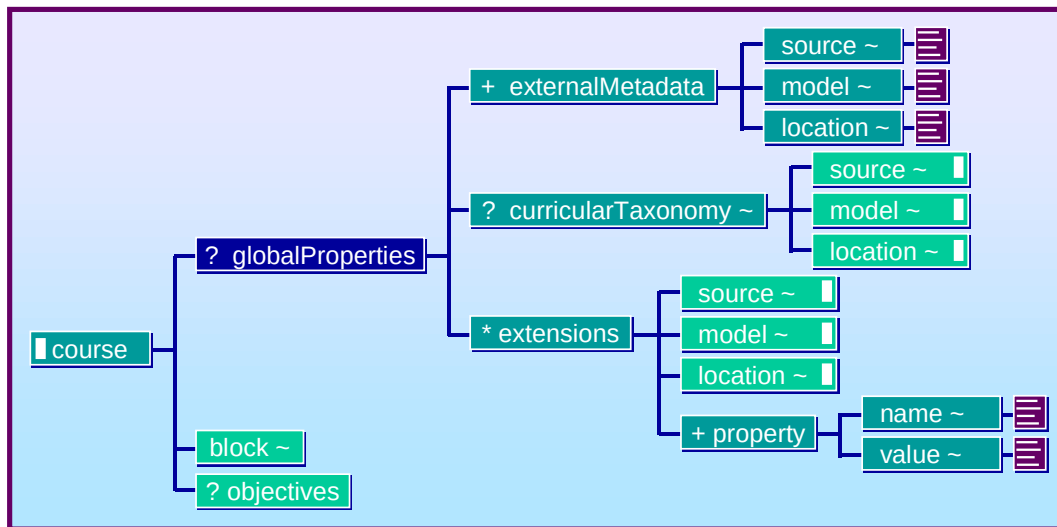


Figure 5-2. *globalProperties* XML DTD Structure

Note for Figure 5-2: No notation = one element required; “?” = zero or one (optional); “+” = one or more required; “*” = zero or more required.

5.6.1 Global Properties—*externalMetadata*

External course metadata are referenced within *globalProperties* in *metadataExternalRef*. These metadata define, among other things, information that can be searched externally, such as the course title, course description, version, and so forth. The SCORM assumes that the external course metadata “pointed to” (referenced) by *metadataExternalRef* is a separate metadata record, as defined in Section 7.2 of this document.

The CSF “points to” an external metadata record rather than including it in-line to avoid redundancy and to reduce the CSF’s overall size and complexity. Because a stand-alone metadata definition based on the IEEE LOM specification exists, it does not have to be duplicated within the CSF.

5.6.2 Global Properties—*curricularTaxonomy*

globalProperties provides a tag for how courses are constructed, called *curricularTaxonomy*. This tag can identify the methodology of a particular community of users in assembling course components. This element indicates the user community and, therefore, infers the structure of the course, naming conventions (e.g., unit, lesson, learning step, and so forth), and number of levels or tiers of content aggregation.

Appendix E of this document gives some examples of different course structures. Table 5-2 illustrates how *curricularTaxonomy* might be used:

Table 5-2. Using the *curricularTaxonomy*

Model: Army	Model: Air Force	Model: Marine Corps	Model: Canadian SCO
Course	Course	Course	Course
Module	Block	Phase	Performance Objective
Lesson	Module	SubCourse (Annex)	Enabling Objective
Learning Objective	Lesson	Lesson	Teaching Point
Learning Steps	Learning Objective	Task	
		Learning Objective	
		Learning Step	

In these examples, the *curricularTaxonomy* model element helps to predict the probable depth of the content hierarchy. For example, the Marine model contains seven course levels, the Army and Air Force models contain five course levels, and the Canadian model contains four levels.

Understanding the design methodology used during course construction and the different approaches to content aggregation will assist in content reuse and provide important information to an LMS when courses are moved.

5.6.3 Global Properties—*extensions*

See Section 5.5.5, *CSF Extension Mechanism*.

5.7 Course Structure Hierarchy—*block*

The *block* group defines all the course elements and their organizational structure. This tree structure defines a hierarchical lesson plan for a course. The ordering of the tree elements defines a default sequence for the execution of each of the “assignments” in the course. Embedded within this tree structure are data elements defining the type, source, and location of each course element. Figure 5-3 shows a sample course structure. Figure 5-4 shows the *block* XML DTD structure.

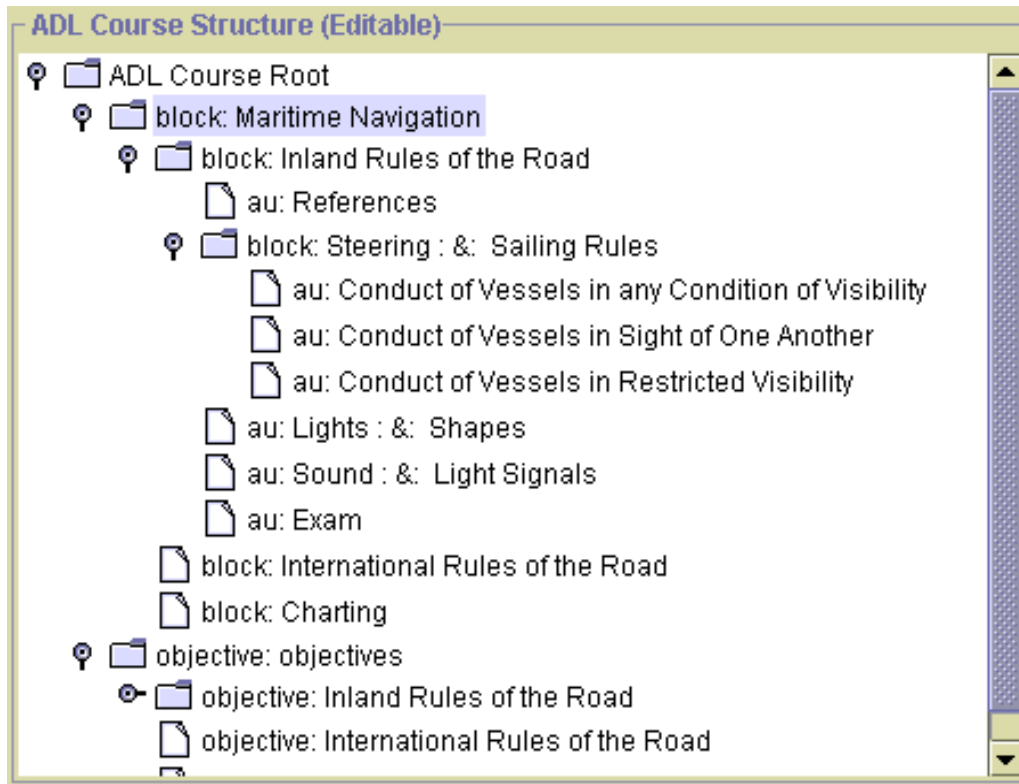


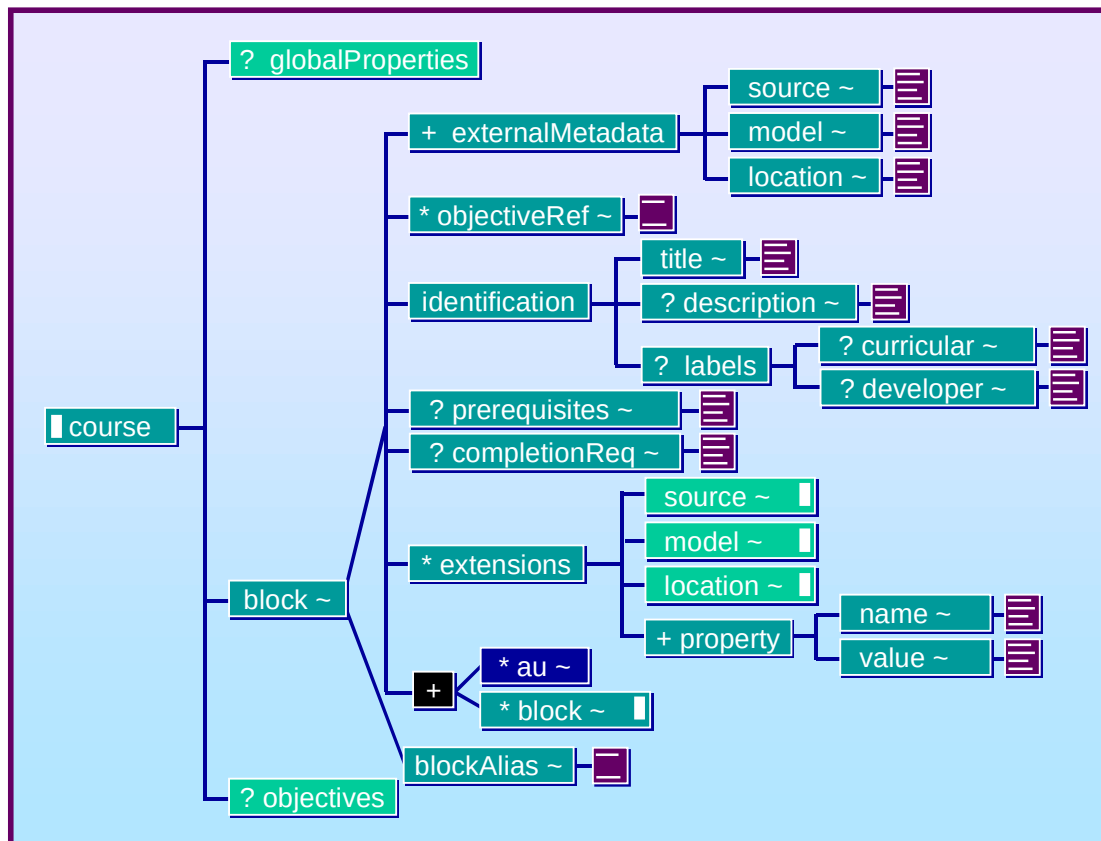
Figure 5-3. Example Course Structure

The block structure defines two types of course content: blocks and assignable units (aus). The terms block and au are drawn directly from the AICC’s CMI specification AICC’s Document CMI001, *AICC CMI Guidelines for Interoperability (Revision 3.0.1, 24 November 1999)*.

A block is a hierarchical grouping of other blocks or assignable units. The top-most block (the root of the hierarchical tree) is the top level of the “course.” This top level may contain blocks or assignable units, or both.

Blocks may be nested. Blocks might group together smaller groups of blocks, which, in turn, may contain references to content (aus). Examples of blocks might include “chapters,” “units,” “learning steps”, and so forth.

This hierarchical approach to representing course structure in the CSF is intended to accommodate a wide variety of courseware models. Simple courses may not have many levels of blocks, while complex courses could have many blocks within blocks. The *globalProperties* tag, *curricularTaxonomy*, is intended to identify the model convention type.

Figure 5-4. *block* XML DTD Structure

Note for Figure 5-4: No notation = one element required; “?” = zero or one (optional); “+” = one or more required; “*” = zero or more required.

5.7.1 Block—*externalMetadata*

The optional *externalMetadata* subelement offers course developers the opportunity to more fully describe course blocks throughout the content hierarchy. Similar to *globalProperties*, this element is expected to point to stand-alone XML metadata records as specified in Section 7 of this document.

5.7.2 Block—*objectivesRef*

This subelement is used to “point” to an objective. It ties a particular portion of a course to a specific objective. See Section 5.5.3, *Unique IDs and ID References, and Aliases*.

5.7.3 Block—*identification*

This subelement provides context-specific information about a block including title, description, curricular label, and developer label. See Section 5.5.4, *Identification Model*.

5.7.4 Block—*prerequisites*

This subelement defines what portions of a course must be completed before this element's parent may be started. See Section 5.5.2, *Course Sequencing Using Prerequisites and Completion Requirements*.

5.7.5 Block—*completionReq*

This subelement defines what portions of a course must be completed before this element's parent is considered to be "complete." See Section 5.5.2, *Course Sequencing Using Prerequisites and Completion Requirements*.

5.7.6 Block—*extensions*

See Section 5.5.5, *CSF Extension Mechanism*.

5.7.7 Block—*au/block*

This subelement can include one or more subblocks and/or one or more assignable units. This defines the course hierarchy. The following XML fragment illustrates the hierarchical tree structure:

```

. . .
<course>
  <block id="B1">
    <identification>
      <title>Maritime Navigation</title>
      <labels>
        <curricular>UNIT</curricular>
      </labels>
    </identification>

    <block id="B2">
      <identification>
        <title>Inland Rules of the Road</title>
        <labels>
          <curricular>MODULE</curricular>
        </labels>
      </identification>
      <au id="A1">
        <identification>
          <title>References</title>
        </identification>
        <launch>
          <location>/Courses/Course01/Lesson01/au01.html</location>
        </launch>
      </au>
      <block id="B3">
        <identification>
          <title>Steering &#38; Sailing Rules</title>
          <labels>
            <curricular>MODULE</curricular>
          </labels>
        </identification>
      </block>
    </block>
  </course>
. . .

```

5.7.8 Block—*blockAlias*

This subelement provides a reference to a previously defined block to avoid the need to duplicate identical block definitions within a block structure. See Section 5.5.3, *Unique IDs and ID References, and Aliases*.

5.8 Assignable Unit—*au*

References to course content in the CSF are made using the *au* (assignable unit) element. This element references learning content that is to be launched by an LMS on the student's client platform. The *au* element within the CSF contains all the context-specific information needed to launch a “chunk” of content, such as its location, name, launch parameters, and prerequisites. **[Note:** *The au elements may be arbitrarily small pieces of content that, at a minimum, are launched and then terminated with no communications with an LMS, or they may be more complex pieces of content that generate data that are tracked by an LMS.*] Figure 5-5 shows the *au* XML DTD structure.

The *au* element in the CSF may optionally use *externalMetadata* to reference an external metadata record that could contain information about a piece of content that is independent of a particular course (i.e., noncontext specific). Such metadata would be useful within “content repositories.”

5.8.1 Assignable Unit—*externalMetadata*

The optional *externalMetadata* subelement is intended to point to external stand-alone XML records that more fully describe the assignable unit content referenced in this parent's element. This reference is expected to be informational rather than functional.

5.8.2 Assignable Unit—*objectivesRef*

This subelement is used to “point” to an objective. It ties a particular portion of a course to a specific objective. See Section 5.5.3, *Unique IDs and ID References, and Aliases*.

5.8.3 Assignable Unit—*identification*

This subelement provides context-specific information about an assignable unit including title, description, curricular label, and developer label. See Section 5.5.4, *Identification Model*.

5.8.4 Assignable Unit—*prerequisites*

This subelement defines what portions of a course must be completed before this element's parent can be started. See Section 5.5.2, *Course Sequencing Using Prerequisites and Completion Requirements*.

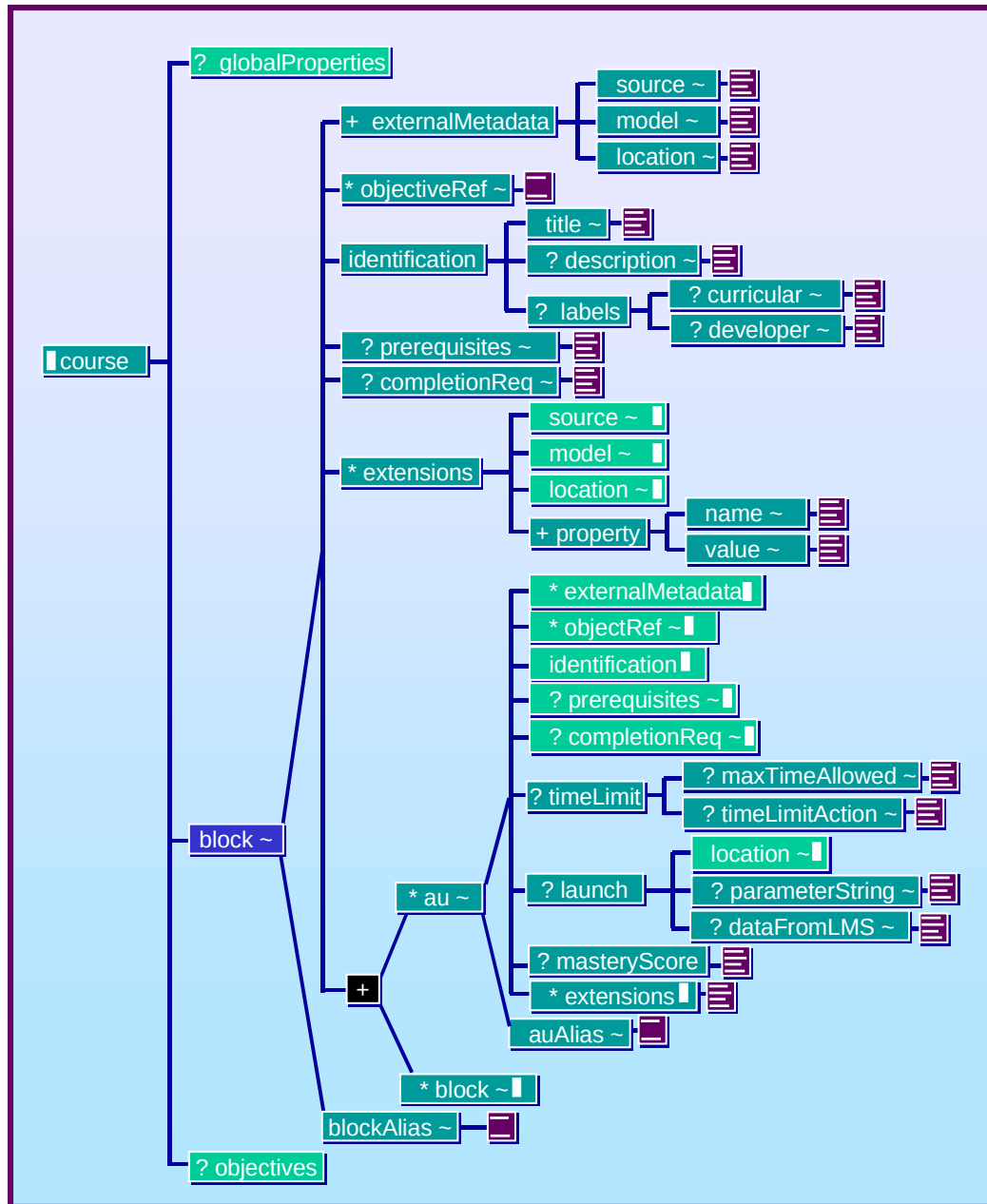


Figure 5-5. au XML DTD Structure

Note for Figure 5-5: No notation = one element required; “?” = zero or one (optional); “+” = one or more required; “*” = zero or more required.

5.8.5 Assignable Unit—*completionReq*

This subelement defines what portions of a course must be completed before this element’s parent is considered to be “complete.” See Section 5.5.2, *Course Sequencing Using Prerequisites and Completion Requirements*.

5.8.6 Assignable Unit—*timeLimit*

This subelement has two time-related subelements: *maxTimeAllowed* and *timeLimitAction*. The first subelement, *maxTimeAllowed*, defines the amount of time the student is allowed to have in the current attempt on the lesson. The second subelement, *timeLimitAction*, defines what a lesson should do when the maximum time allowed has elapsed. Similar to *prerequisites* and *completionReq*, this element has a “type” attribute for identifying the kind of value in use. The “AICC” case, which the SCORM has adopted, has a fixed vocabulary for this element:

Time Limit Action:	Exit	Continue
	Message	no message

5.8.7 Assignable Unit—*launch*

This subelement has three subelements: *location*, *parameterString*, and *dataFromLMS*.

The *location* subelement contains the Universal Resource Identifier (URI) of the actual assignable unit content to be launched. This is the manner in which the CSF “points” to launchable assignable units.

The optional *parameterString* subelement holds a character string to use during content launch if needed. [This is similar to options one might type following the name of a program or, in Java, the arguments one would process in main (String[] args)].

The *dataFromLMS* subelement provides a place for initialization data expected by the content during launch. These data are unconstrained and undefined. **Note:** *This element is not yet well defined and should be used with caution.*

5.8.8 Assignable Unit—*masteryScore*

This subelement establishes the passing score for the assignable unit in this case. What is considered a passing score often depends on the context of an assignable unit within a course. Some courses may set the mastery score for an “au” higher than in other courses.

This subelement assumes that the assignable unit has some content that will report a score (e.g., a test) over the API and data model defined in Section 6 of this document.

5.8.9 Assignable Unit—*extensions*

See Section 5.5.5, *CSF Extension Mechanism*.

5.8.10 Assignable Unit—*auAlias*

This subelement provides a reference to a previously defined assignable unit to avoid the need to reduplicate identical *au* definitions within a block structure. See Section 5.5.3, *Unique IDs and ID References, and Aliases*.

5.9 Objectives

At the highest level, course objectives are statements of skills, knowledge, and attitudes to be acquired by the student. Within the CSF, the *objectives* group provides the means for identifying objective titles, descriptions, prerequisites, completion requirements, and so forth. Figure 5-6 shows the *objectives* XML DTD structure.

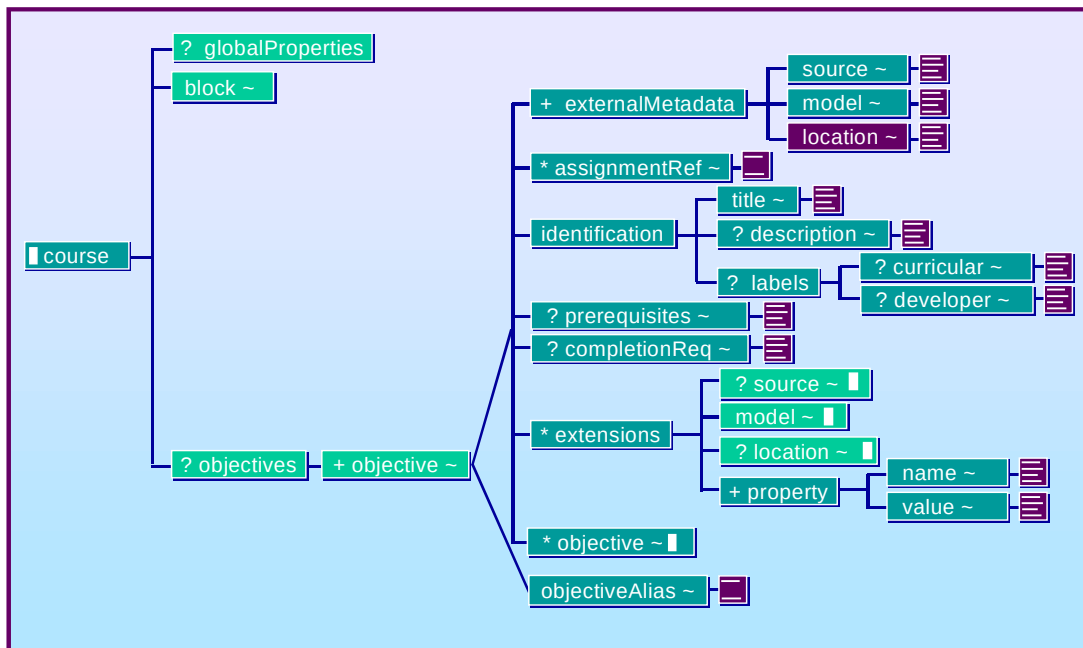


Figure 5-6. objectives XML DTD Structure

Note for Figure 5-6: No notation = one element required; “?” = zero or one (optional); “+” = one or more required; “*” = zero or more required.

These statements of objectives may be hierarchically nested (or simply listed) and may include explicit references to specific course elements within the assignments hierarchy. This permits an LMS to track the completion status of objectives and, thus, provides a means for the LMS to determine appropriate paths through course content.

Note: The root of the objective structure is the *objectives* (plural) element and is not in itself an objective (singular). All actual objective elements occur somewhere under the *objectives* root. The reason for this construct is that objectives may be a single, “flat” list of objectives, a hierarchy of objectives, or both. The top-level *objectives* element provides a specific place to “hang” objective elements, regardless of the underlying structure.

5.9.1 Objective—*externalMetadata*

This optional *externalMetadata* subelement offers course developers the opportunity to more fully describe *objectives* blocks throughout the objectives hierarchy. At this time,

no defined current practice for applying metadata to objectives exists; therefore, this category exists mainly for consistency with the rest of the CSF model, with the expectation that it will provide a potential, future value yet to be defined.

5.9.2 Objective—*assignmentRef*

This subelement is used to “point” to a particular block or assignable unit in the content hierarchy. The term “assignment” is used to refer to either a block or an assignable unit. It ties a particular objective to a specific part of a course. See Section 5.5.3, *Unique IDs and ID References, and Aliases*.

5.9.3 Objective—*identification*

This subelement provides context-specific information about a block, including title, description, curricular label, and developer label. See Section 5.5.4, *Identification Model*.

5.9.4 Objective—*prerequisites*

This subelement defines what portions of a course must be completed before this element’s parent can be started. See Section 5.5.2, *Course Sequencing Using Prerequisites and Completion Requirements*.

5.9.5 Objective—*completionReq*

This subelement defines what portions of a course must be completed before this element’s parent is considered to be “complete.” See Section 5.5.2, *Course Sequencing Using Prerequisites and Completion Requirements*.

5.9.6 Objective—*extensions*

See Section 5.5.5, *CSF Extension Mechanism*.

5.9.7 Objective—*objectiveAlias*

This subelement provides a reference to a previously defined objective to avoid the need to duplicate identical objective definitions within an objectives structure. See Section 5.5.3, *Unique IDs and ID References, and Aliases*.

5.10 CSF Element Definitions/Descriptions

Table 5-3 lists the CSF element definitions and descriptions.

Table 5-3. CSF Element Definitions/Descriptions

Element Name	Description	Value Type
assignmentRef	Reference to a particular element in the assignment hierarchy (e.g., <assignmentRef relation="satisfied by" targetIDs="B1,A23"/>).	Relation = (reserved) TargetIDs: List of Identifier Refs
au	au is the smallest element of instruction or testing to which a student can be routed by a LMS. It refers to "content" launched by the LMS system. This holds a unique (to this course) ID identifier for a particular au. IDs are generated by the application (e.g., an LMS) that creates a CSF XML file. (Other elements may refer to this unique ID.)	Id = Identifier Value; must start with "A"
auAlias	Reference to a previously defined au (permits one assignable unit to be used more than once within a course).	Targeted = Identifier Value
block	A grouping of related structural elements. Blocks contain either assignable units or other blocks or both. Blocks always contain other course elements. This holds a unique (to this course) ID identifier for a particular block. IDs are generated by the application (e.g., an LMS) that creates a CSF XML file. (Other elements may refer to this unique ID.)	Id = Identifier Value; Default = REQUIRED
blockAlias	Reference to a previously defined block (permits one block to be used more than once within a course).	Targeted = Identifier Value
completionReq	Course elements that a student must complete before considering a given structure element complete. It uses a "script" that defines the logical rules to be applied. The script type must be defined (e.g., <completion type="aicc_script"> <![CDATA[B1&B2&A1]]> </completion>).	Type = Character Data; Value = CDATA
course	Root level of Course Structure representation.	
curricular	Local name of course element (e.g., "UNIT", "MODULE," "LEARNING STEP").	PCDATA
curricularTaxonomy	Organizational methodology used to construct the course	PCDATA
dataFromLMS	Unconstrained (undefined) initialization data expected by content when it is launched by the LMS.	PCDATA
description	Context-specific textual information about the course element. It can contain the purpose, scope, or summary. (Defined by course author).	PCDATA
developer	Developer label: an organization-specific identifier (e.g., D509).	PCDATA
extensions	Defines extensions to course element definitions and their source.	
externalMetadata	The value of this element refers or points to the location of the metadata describing this course.	
globalProperties	Properties of the course as whole.	
identification	Identifies course context-specific information.	
labels	Context-specific local label (e.g., unit, chapter, learning step).	
launch	Information needed by an LMS to launch an assignable unit.	
location	Universal Resource Locator (URL) Location.	PCDATA
masteryScore	Tells LMS how to compute the score returned by LMS and defines the passing score for this assignable unit in this context.	PCDATA
MaxTimeAllowed	*** maxTimeAllowed: The amount of time the student is allowed to have in the current attempt on the lesson	PCDATA (Format TBD)

Table 5-3. CSF Element Definitions/Descriptions (Continued)

Element Name	Description	Value Type
model	Name of a specific data model used by this course (e.g., "cmi", or "ARMY314", or "IMS v1.0").	PCDATA
name	Descriptive name of a course property extension (e.g., "difficulty," as in degree of) .	PCDATA
objective	A statement of skills, knowledge, and aptitudes to be acquired by the student. This holds a unique (to this course) ID identifier for a particular objective. IDs are generated by the application (e.g., an LMS) that creates a CSF XML file (other elements may refer to this unique ID).	id = Identifier Value; Must start with "O"
objectiveAlias	Reference to a previously defined objective (permits one objective to be used more than once within a course).	targetID = Identifier Ref
objectiveRef	Reference to a particular objective in the objective hierarchy.	relation = (reserved) targetIDs = List of Identifier Refs
objectives	Root level of objectives tree. Statements of skills, knowledge, and aptitudes to be acquired by the student	
parameterString	String of characters needed to successfully launch a content assignable unit.	PCDATA
prerequisites	Expression indicating what a student must accomplish before beginning this course element. Course elements that a student must complete before beginning a block or assignable unit. It uses a "script" that defines the logical rules to be applied. The script type must be defined. e.g., <prerequisites type="aicc_script"> <![CDATA[B1&B2&A1]]> </prerequisites>.	type = Character Data;; value=CDATA
property	Name/value pair extension for this course.	
source	Authority or source of data model w/reference to a spec. If available e.g., "AICC AGRO10 v3.4", or ARMY TRADOC spec123, or "IMS v4.2"	PCDATA
timeLimit	Time values or actions associated with this assignable unit in this context.	
timeLimitAction	What the lesson is to do when the max time allowed is exceeded. AICC examples: "exit," "continue," "message."	type = Character Data; value = PCDATA
title	Context specific title. Can be used by an LMS system in menus, screens, and so forth.	PCDATA
value	Value associated with the named extension (e.g., "easy").	PCDATA

5.11 Examples

The following examples are intended to illustrate CSF concepts and, for clarity, include a much smaller number of elements than would exist in a true CSF record.

5.11.1 Simplest Example of CSF Record

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE course SYSTEM "file:scormcsf(1.0).dtd" >
<course>
  <!--*** This is the smallest possible valid CSF record.
  It has only one assignable unit (au) and lists only a title-->
  <block>
    <au id="A1">
```

```

    <identification>
      <title>A Really, Really Small Course Chunk Pointing to Nothing</title>
    </identification>
  </au>
</block>
</course>

```

5.11.2 Simple Example Pointing To Content

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE course SYSTEM "file:scormcsf(1.0).dtd" >
<course>
  <!--*** To the small CSF record example here is added reference to external content-->
  <block>
    <au id="A1">
      <identification>
        <title>A Really, Really Small Course Chunk Pointing to Something</title>
      </identification>
      <launch>
        <location>http://www.notmuch.com/smallchunk.html</location>
      </launch>
    </au>
  </block>
</course>

```

5.11.3 Example of Course With One Block

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE course SYSTEM "file:scormcsf(1.0).dtd" >
<course>
  <!--*** This is an example of a course with one block
  containing two assignable [nee atomic] content units.-->
  <block>
    <block id="B1">
      <identification>
        <title>Introduction to Blocks 101</title>
        <description>This is a simple block of course elements; not much to build with yet.</description>
      </identification>
      <au id="A1">
        <identification>
          <title>Building With Atoms</title>
        </identification>
      </au>
      <au id="A2">
        <identification>
          <title>Splitting Atoms With Hairs</title>
        </identification>
      </au>
    </block>
  </block>
</course>

```

5.11.4 Example of Course With Blocks Within Blocks

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE course SYSTEM "file:scormcsf(1.0).dtd" >
<course>
  <!--*** This is an example of a course with one block, containing two blocks,
  each with two assignable [nee atomic] content units.-->
  <block>
    <block id="B1">
      <identification>
        <title>Building Big Things From Small Things</title>
        <description>This lesson is about content aggregation</description>
      </identification>
      <block id="B2">
        <identification>

```

```

<title>Introduction to Blocks 101</title>
<description>This Unit is about splitting and fusing atoms</description>
</identification>
<au id="A1">
  <identification>
    <title>Building With Atoms</title>
  </identification>
</au>
<au id="A2">
  <identification>
    <title>Splitting Atoms With Hairs</title>
  </identification>
</au>
</block>
<block id="B3">
  <identification>
    <title>Introduction to Leggo Blocks 107</title>
    <description>This Unit is about Leggo Blocks</description>
  </identification>
  <au id="A4">
    <identification>
      <title>Building With Leggo Blocks</title>
    </identification>
  </au>
  <au id="A5">
    <identification>
      <title>Connecting Leggos Together</title>
    </identification>
  </au>
</block>
</block>
</block>
</course>

```

5.11.5 Multi-Tiered Example (Seven Levels)

Note: *This example is not really a valid CSF record since it omits some required elements for clarity. Nonetheless, it illustrates deep nesting of levels, or tiers.*

```

<course>
  <globalProperties>
    <externalMetadata>
      <location>army_courseMetadata.xml</location>
      <model>need example here</model>
      <source>need example here</source>
    </externalMetadata>
  </globalProperties>
  <block>
    <block>
      <identification>
        <title/>
        <labels>
          <curricular>COURSE</curricular>
          <developer>ATSC</developer>
        </labels>
      </identification>
    <block>
      <identification>
        <title/>
        <labels>
          <curricular>MODULE</curricular>
          <developer>ATSC</developer>
        </labels>
      </identification>
    <block>
      <identification>
        <title/>

```

```

<labels>
  <curricular>LESSON</curricular>
  <developer>ATSC</developer>
</labels>
</identification>
<block>
  <identification>
    <title/>
    <labels>
      <curricular>LEARNING OBJECTIVE</curricular>
      <developer>ATSC</developer>
    </labels>
  </identification>
  <au>
    <identification>
      <labels>
        <curricular>LEARNING STEP</curricular>
      </labels>
    </identification>
    <externalMetadata>
      <location>ARMY MEDIA REPOSITORY</location>
    </externalMetadata>
  </au>
</block>
</block>
</block>
</block>
</course>

```

5.12 Conformance Testing

Conformance testing focuses on testing CSF files generated by an LMS or authoring tool and verifying that the resulting CSF can be read and interpreted correctly by another LMS or authoring tool.

A CSF record is expected to be created from within an LMS or course-authoring environment. Within that environment, a course may have its own internal representation of course structure and its related elements. A conforming LMS or course creation environment is expected to map its internal representation to a valid CSF record (as defined by the SCORM CSF DTD).

Similarly, a conforming LMS or authoring environment is expected to read and correctly interpret the SCORM CSF format and map the contents of the CSF to its internal representation as required. The course should then execute as intended.

First, generated CSF records must be validated against the DTD. Next, they must be tested for “adequacy.” The examples in this document show that it is possible to produce valid CSF records that are not, in fact, adequate to define a course. Specific criteria are expected to be developed, and these criteria should define the necessary elements and whether the values of those elements adequately define a course. These criteria will constitute the conformance “policy” for a particular community of users.

An additional set of conformance tests will be developed using dummy course examples that can be created in one environment, exported to the CSF, and imported to another environment. These dummy examples are to be designed to test particular course behavior and to determine whether the CSF correctly captures that behavior. A simple test, for example, would be for an LMS or authoring tool to generate a CSF format and then read it back in. Having read it back, the course should behave exactly as before.

In summary, conformance testing consists of verifying that a CSF record is valid (against the DTD) and adequate to represent a course and that the LMS or authoring tools correctly implement the basic mapping from internal representation to the intermediate CSF (and back again).

Details and test criteria are expected to be developed in future versions of this document.

5.13 Sample Course Mappings

This section will contain fragments of existing courses as they would be represented in the CSF format.

6. Runtime Environment

6.1 Overview

A requirement of the SCORM is that learning content be reusable across multiple LMSs—regardless of the tools used to create the content. For this to be possible, there must be a common way to start content, a common way for a content to communicate with an LMS, and predefined data elements that are exchanged between an LMS and content during its execution. This document defines these three mechanisms as *Content Launch*, *API* (Application Program Interface), and *Data Model* (see Figure 6-1).

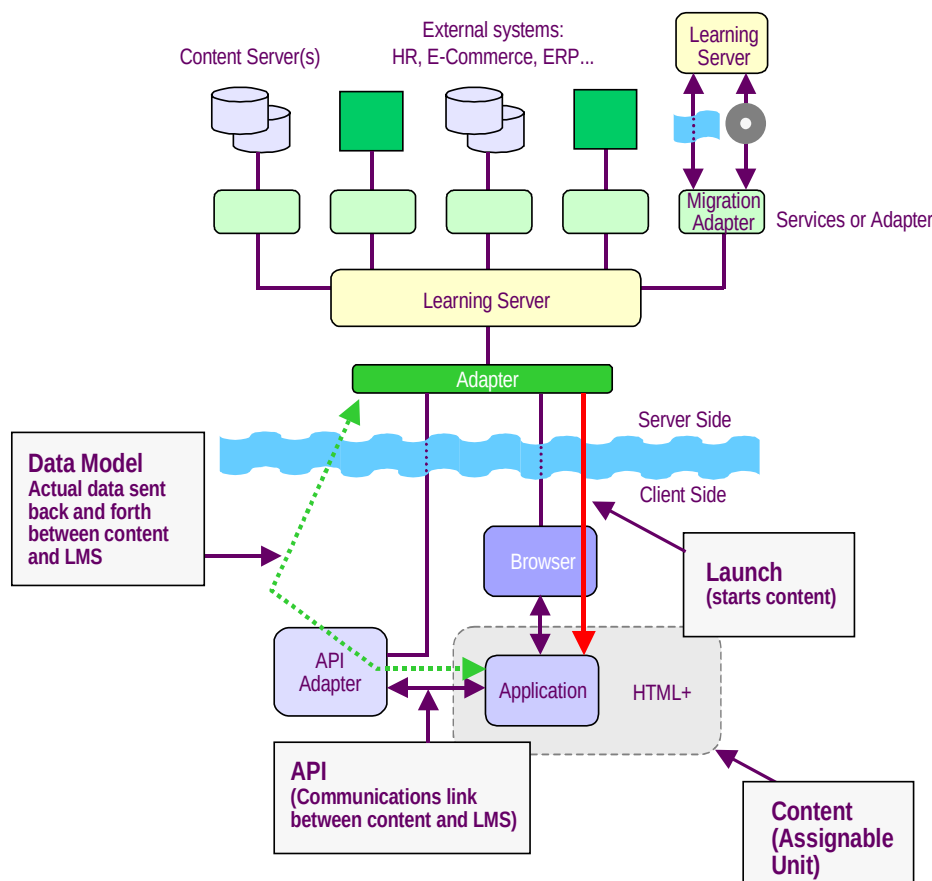


Figure 6.1. Launch, API, and Data Model Mechanisms as They Apply to the SCORM Architectural View

The *Content Launch* mechanism defines a common way for LMSs to start Web/browser-based content. It permits the content to locate the means to communicate status and tracking information back to the LMS. The communication takes place using a common API.

The *API* mechanism is the vehicle for informing the LMS of the state of the content (e.g., initialized, finished, or errors), and is used to get and set data between the LMS and the content (e.g., score, time limits).

The *Data Model* mechanism is a predefined list of data elements used to track the status of content. In its simplest form, the data model defines elements that both the LMS and content are expected to “know” about. The LMS must maintain the status of required data elements, and the content must use only these data elements if reuse across multiple systems is to occur.

6.2 SCORM and the AICC API Specification

The SCORM is based directly on the runtime environment functionality defined in Appendix B, “API-Based CMI Communication,” of AICC’s Document CMI001, *AICC CMI Guidelines for Interoperability (Revision 3.0.1, 24 November 1999)*. Appendix B of this document includes this specification.

ADL collaborated with AICC members and participants to develop a common *Launch* and *API* specification and to adopt Web-based data elements from the complete set defined in the AICC’s Semantic Document v3.0 (CMI-Sem30.doc). This document is available from the IEEE LTSC’s Web page at <http://www.manta.ieee.org>.

The following sections provide an overview of the key elements of the AICC API specification as they relate to the SCORM.

6.3 API Adapter

The SCORM assumes that an LMS will supply an API adapter that implements some kind of communications transport between the server-side LMS and the client. This adapter must shield content from the particular adapter implementation details so that content (assignable units) need not have any knowledge of the underlying communications infrastructure and, instead, relies solely on the existence of a standardized API.

For example, an API adapter might be implemented as a Java applet that looks similar to the following:

```
public class API extends applet {
    private static LMSErrorManager lmsErrorManager;
    private static LMSCoreData lmsCoreData;
    private static AUCoreData auCoreData;
    private static boolean is LMSInitialized;
    private URL servletURL;
    public void init()
    { . . . }
    public String LMSInitialize(String param)
    { . . . }
    . . .
    public String LMSGetValue(String element)
    { . . . }
    public void LMSSetValue(String element, String value)
```

```

{. . .}
public void LMSCommit()
{. . .}
public String LMSGetLastError()
{. . .}
public String LMSGetErrorString( String errorCode)
{. . .}
public String LMSGetDiagnostic( String errorCode)
{. . .}
. . .

```

Note that an API adapter can be implemented in other languages, such as C++, and loaded as a plug-in. The API adapter implementation is expected to be LMS vendor specific. The code fragment sample above is only an example approach. Figure 6-2a and Figure 6-2b illustrate several different possible approaches for implementing an LMS API adapter:

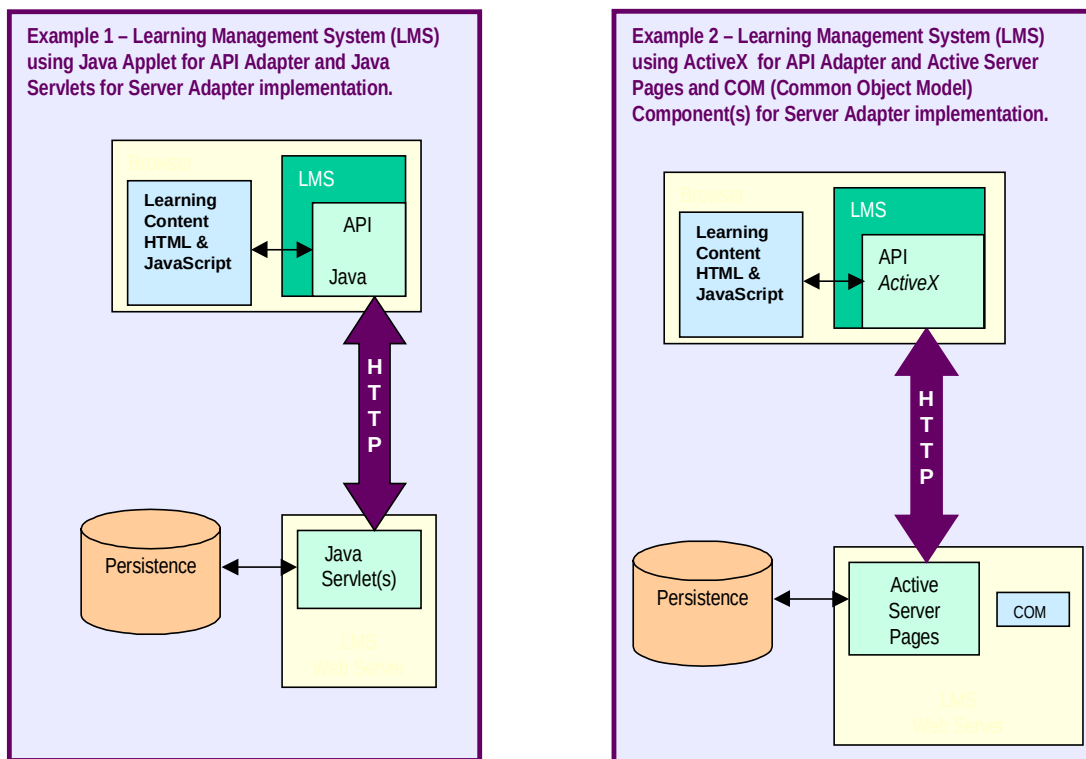


Figure 6-2a. Examples 1 and 2 of Possible Approaches for Implementing an LMS API Adapter

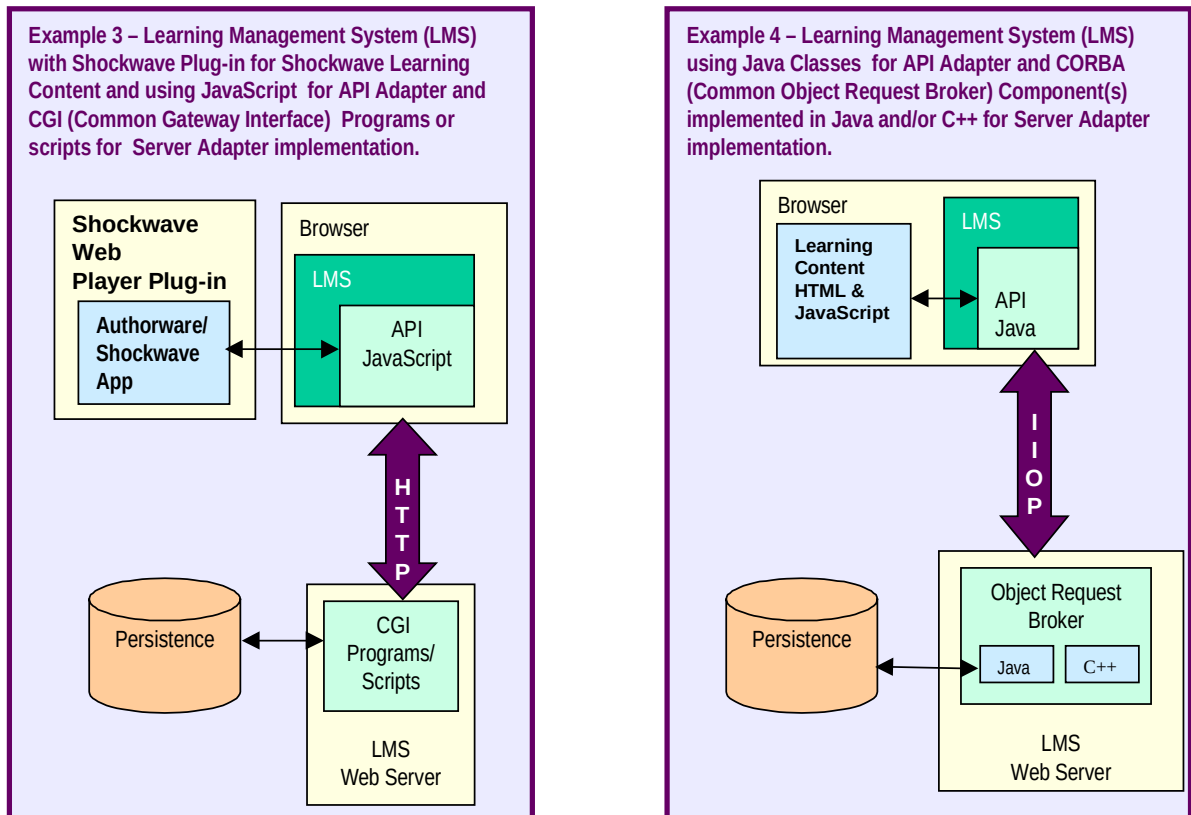
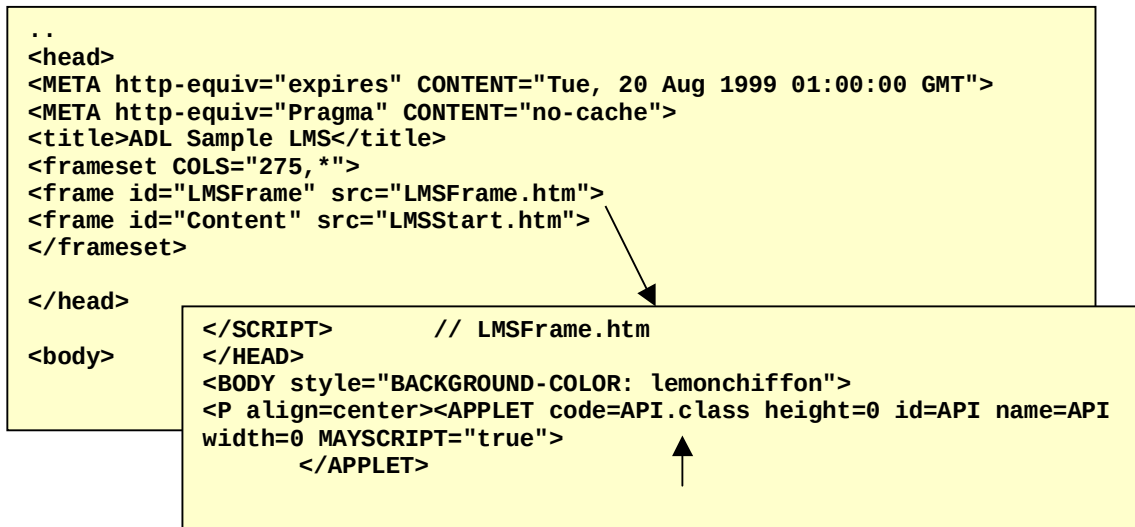


Figure 6-2b. Examples 3 and 4 of Possible Approaches for Implementing an LMS API Adapter

6.4 Content Launch

During the development of the AICC API specification and the ADL SCORM, the AICC, IEEE, and ADL participants examined the various possible approaches to launching content and initializing communications with LMS environments. Many methods were proposed, but most were either too complex or poorly supported in today's Web environment. Finally, a common approach was proposed (with many thanks to Claude Ostyn). This approach is relatively simple for content authoring tools to support and is broadly implementable with today's browser technology.

Nearly all browsers—and certainly all of the popular ones—natively support JavaScript. (The more accurate term is actually ECMAScript, which is the official standard name of JavaScript.) The launch scheme described in Appendix B requires an LMS to launch content from a window that contains a common API implementation (or, alternatively, to provide a parent frame that contains the API). The API is accessible through JavaScript calls. This ensures that content is “wrapped” with the means to establish communications with the LMS once it begins to execute.



The content (assignable unit), which may be arbitrarily small and simple or relatively complex, “connects” to an API object to establish communications. *Content Launch* obtains the API object by checking for its existence on any parent window or the opener window. The following JavaScript example (see Appendix B, Section B.2.3.1) illustrates how content might locate the API.

```

// returns the LMS API object (may be null if not found)
FindAPI(win)
{
  if (win.API != null)
    return win.API;
  else if (win.parent == null)
    return null;
  else
    return FindAPI(win.parent);
}

// obtain the LMS API
API = FindAPI(window);
If (API == null)
  API = FindAPI(window.opener);

```

As long as content contains a binding to the defined API object, an LMS can launch and track its execution, regardless of the tool used to create it. Thus, one key function that enables reuse is provided.

6.5 Application Program Interface (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 content to communicate with learning management systems, yet it shields the particular communications implementation from the content developer. How the “insides” of an API are implemented should not matter to content developers provided they use the same

“outside” interface. An API “hides” implementation details from content and, therefore, promotes reuse and interoperability.

A key aspect of the AICC API is that it is invoked *only* from within content. Once Web-based content is launched, it is assumed that it can then “get” and “set” information with an LMS. The functions of this API are threefold:

1. **Execution state.** The API has *initialize()* and *finish()* function calls that tell the LMS that the content is present and active (“I’m here! I’m running!”) or that it has completed and ended (“I’ve stopped, and I’m no longer here.”). These are the *only* two required API calls that content must make. This means that an extremely simple piece of content, such as a media clip, would only need to be “wrapped” with enough code to embed an *initialize()* and *finish()* to interoperate across multiple LMS environments. Even large “chunks” of content might only have these two calls if no other information or data need to be tracked by the LMS. (This is not likely for robust learning content; however, it is possible and provided for.)
2. **State management.** The API has four functions that are used to handle errors and to force a transfer of cached information back to an LMS: *LMSGetLastError()*, *LMSGetErrorString(errornumber)*, *LMSGetDiagnostic(parameter)*, and *LMSCommit(parameter)*.
3. **Data transfer.** The remaining two API functions are used to transfer data to and from an LMS: *LMSGetValue(cmi.category)* and *LMSSetValue(cmi.category, value)*. Note that the API is designed to get and set data values that are separately defined by an external data model. The AICC specification defines one such data model here (and in the AICC document referenced in Section 6.2) called “cmi.” Other data models could also be developed and used with this API; thus, the AICC specification in Appendix B has been designed to be generic. The API does not “know” about what specific data that it can set or get.

The following code example illustrates how API calls could be embedded within an HTML page using JavaScript. This is only one possible way to use the API; there are other approaches depending on the type of content.

<SCRIPT LANGUAGE=JAVASCRIPT >

```
function loadPage()
{
    LMSInitialize();
    getStartTime();
    LMSSetValue( "cmi.core.score.max", "5" );
    LMSSetValue( "cmi.core.score.min", "0" );

    var student_Name = LMSGetValue
                        ( "cmi.core.student_name" );
    ...
}
```

Table 6-1 is an API table. **Note:** See Appendix B for detailed API definitions and data types.

Table 6-1. API Table (From AICC CMI 3.0.1, Appendix B)

Function	Description	API Call	Return Value
Initialize	The content must call this function before calling any other API function. It indicates to the LMS system that the content is going to communicate. The LMS can take any initialization steps required in this function.	LMSInitialize()	A string convertible to CMIBoolean
Finish	The content must call this function before it terminates, if it successfully called LMSInitialize at any point. It signals to the LMS that the content has finished communicating. The content may not call any API function except LMSGetLastError after it calls LMSFinish	LMSFinish()	None
Get a value	This is used to determine values for various categories and elements in the CMI data model. Only one value is returned for each call. The category and/or element is named in the argument.	LMSGetValue(dataModel.category) LMSGetValue(dataModel.category.element)	A string convertible to appropriate data type
Set a value	This is how data categories and elements get values. The argument indicates which category or element is being set. Only one value can be set with a single function call.	LMSSetValue(dataModel.category, value) LMSSetValue(dataModel.category.element, value)	None
Send cache to LMS	If the ECMAScript is caching LMSSetValue values, this call requires that any values not yet sent to the LMS be sent.	LMSCommit(parameter)	None
Determine error code	The content must have a way of assessing whether any given API call was successful, and if it was not successful, what went wrong. This routine returns an error code from the previous API call. Each time an API function is called (with the exception of this one), the error code is reset in the API. The content may call this any number of times to retrieve the error code, and the code will not change until the next API call.	LMSGetLastError()	A string convertible to CMInteger
Obtain text related to error	This function enables the content to obtain a textual description of the error represented by the error code number.	LMSGetErrorString(errornumber)	CMString256
Determine vendor-specific diagnostics	This function enables vendor-specific error descriptions to be developed and accessed by the content. These would normally provide additional helpful detail regarding the error.	LMSGetDiagnostic(parameter)	CMString256
Security functions	-TBD-		

6.6 Further Defining Content as “Assignable Units”

References to “content” and “assignable units” in this document can be confusing at first. This confusion is compounded by the additional use of “lesson” in the AICC documents and within the CMI data model. All these terms have roots in historical practices—but from different groups and organizations. Each group has a slightly different intended meaning for these and other terms. This section attempts to clarify the SCORM concepts associated with learning content.

In an important sense, the terms “lesson” and “assignable unit” both refer to client-side content. For client-side content to qualify as an assignable unit, however, it must meet certain minimum requirements, as explained below.

6.6.1 Content as a “Lesson”

“Lesson” has multiple definitions in the AICC CMI specification, including:

- A meaningful division of learning that is accomplished by a student in a continuous effort—that is, at one sitting. That part of the learning that is between designed breaks. Frequently requires approximately 20 minutes to an hour.
- A grouping of instruction that is controlled by a single executable computer program.
- A unit of training that is a logical division of a subchapter, chapter, or course.

In a Web-based environment, a “unit of training” is likely to comprise many content pieces and is less likely to be contained within a single executable computer program; therefore, “lesson” is a somewhat imprecise term that is subject to broad interpretation. The term persists, however, within the CMI data model (see Section 5.1 of the *CMI Guidelines for Interoperability*, Revision 3.0.1) for communications between content and LMSs.

6.6.2 Content as an “Assignable Unit”

The term “assignable unit” also has its roots in the AICC CMI specification, especially related to representing course structure. The term continues to be used in the SCORM document—both in the Course Structure (Section 5) and this section—but the definition has been narrowed in this document.

AICC defines an assignable unit to be both:

- The smallest unit the CMI [LMS] system assigns and tracks
- A program or lesson launched by the CMI [LMS] system.

Three concepts are embedded here. First, an assignable unit is small and stand alone. Second, an LMS launches an assignable unit on the client-side. Finally, the LMS tracks the assignable unit.

The notion of “smallness” is subjective. A more useful way to look at an assignable unit is that it has (by definition) no separate child content components and is, thus, indivisible. An assignable unit could be a very large executable program or an HTML file with just a single letter of text. Provided both examples implement the API correctly, either could be launched and tracked by an LMS.

6.6.3 SCORM “Assignable Unit” Definition

In this document, the definition of “assignable unit” is extended to be an arbitrarily large or small piece of client-side content that, *at a minimum*, contains an *Initialize()* and a *Finish()* API call and has a means to locate an API adapter (such as an API “wrapper” as described in Section 6.4).

Thus, an assignable unit is content, but content is not (in this context) an assignable unit—*unless it implements the API*. Note that there is no obligation to implement any of the other API calls. Those calls are optional and depend on the nature of the content. However, the assignable unit *must* implement *Initialize()* and *Finish()* calls to conform to the SCORM.

The requirement that an assignable unit must implement the API yields the following benefits:

- Any LMS that supports the API can launch assignable units and track them, regardless of who generated them.
- Any conforming LMS can track any assignable unit and know when it has begun or ended.
- Any conforming LMS can launch any assignable unit in the same manner.

6.6.4 Defining a “Sharable Courseware Object” as an Assignable Unit

Because assignable units are required to implement the API consistently to enable reuse, they really are “Sharable Courseware Objects” (SCOs). We continue to use the term “assignable unit” because of its AICC heritage, but, in this document, assignable units meet the requirements of an SCO as defined in Section 4.1 of this document.

6.6.5 Small “Sharable Courseware Objects”

As discussed previously, assignable units (a.k.a. SCOs) technically can be any size; however, the SCORM’s specific intention is to guide people toward the development and use of relatively small content pieces that are suitable for reuse over time. During course design, thought should be given to the smallest logical size of content that might be reused. Such content could then form the basis of content repositories.

6.6.6 Content Sequencing Control

An implicit assumption in the SCORM is the notion that the sequencing of content (i.e., the order in which content is presented to the student) is controlled *solely* by the LMS. This is a major departure from the way courseware has been developed using stand-alone, CBI. In the past, courseware content typically embedded all the navigation information that governs what part of the course the student views next. In most cases, the way in which course sequencing was defined has been unique to the authoring system used to construct the course; thus, it was (and still is) difficult or impossible to share content between different authoring environments.

Within the SCORM, which is deliberately Web-based, flow control is assumed to be on the LMS and *not* within the content itself. This is conceptually important because content reuse cannot really happen if the content has embedded information that is context specific to the course. In this context, flow control means that the LMS decides what content (assignable unit) will next be presented to the student. This recognizes that some content may make decisions (i.e., branch) within itself, but that kind of internal flow is hidden from the LMS.

The determination of what the student experiences is determined solely by the LMS and is defined in large part by the CSF, as defined in Section 5 of this document. Section 5 defines the information about content that is context specific to the course (e.g., the default sequence of content, prerequisites that might alter the delivery path).

Summary Points: In the SCORM, a content assignable unit can *only* be launched by an LMS. An assignable unit cannot itself launch other assignable units. An assignable unit must, at a minimum, contain an *initialize()* and a *finish()* API call to conform with the SCORM.

Content information independent of a particular course (e.g., its generalized title and description, intended use, and technical data type) is contained in separate XML metadata records (see Section 7 of this document). Such data are considered “immutable” in that these data do not change from use to use but are nonetheless critical to discovery (e.g., from within a content repository) and subsequent incorporation into courses.

6.6.7 Toward Adaptive and Intelligent Tutoring

The development of small, reusable, and interoperable pieces of learning content and the shift of content flow control from the client-side to the server-side establish the basis for entirely new learning technologies.

The most obvious benefits of sharability and reuse are the possibility of large content repositories and the development of a new “content economy” where SCOs are traded widely.

An even more interesting prospect is the development of complex learning management systems that can assemble, reorder, and redefine learning content to fit the real-time needs of the student. Unfortunately, the lack of reusable and resequenceable content has delayed this vision from becoming reality. It is a specific purpose of the SCORM to

provide a starting point for the next generation of advanced learning technologies that can (at least in theory) be highly adaptive to student needs.

6.7 “CMI” Data Model

The purpose of establishing a common data model is to ensure that a defined set of information about content can be tracked by different LMS environments. If, for example, it is determined that tracking a student’s score is a general requirement, it is necessary to establish a common way for content to report scores to LMS environments for processing. If pieces of content use their own unique scoring representations, LMSs would not know how to receive, store, or process the information.

The data model in this section is defined as the “CMI” data model because it is derived directly from the AICC CMI specification. This model was chosen for inclusion in the SCORM because it is well defined and has been implemented in the past. The data model specified by AICC in Appendix B omits some of the elements that applied to earlier versions of the AICC specification. Section B.8 of Appendix B presents a table of the differences between the original CMI data model and the one incorporated in this section, along with the rationale for the differences.

Other data models may be defined in the future. This is the reason that all the elements in this section begin with “cmi.” This signals to implementers that the AICC data model is in use. Alternative data models, if developed, would begin with a different designation (e.g., *adl.elementName* instead of *cmi.elementName*).

Three categories of data elements are defined in this section and in more detail in Appendix B, which is the controlling specification:

1. LMS to Content (AU) Communications
2. Content (AU) to LMS Communications
3. Student Data Collection.

These data elements were developed so that an LMS system could evaluate the student’s performance. Examples of data elements in this category include student comments, information about a student’s performance on lesson objectives (such as mastery time), and the path through the content.

The LMS to Content (AU) Communications data set includes elements such as *student_id*, *student_name*, *lesson_status*, and *score*. The AICC document also specifies which elements require mandatory implementation by an LMS and which elements are optional. (For conformance testing purposes, those elements defined as optional must comply with the specification if they are implemented by the LMS.)

The Content (AU) to LMS Communications data set is similar (and mostly symmetrical) with the LMS to Content data set but contains some differences that make contextual sense. (Some data elements logically originate with content only.) Unlike the requirements on the LMS side, all data elements are “optional” on the content side because content may be extremely small and “not very smart.” As a result, some chunks of content might not be 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.

The AICC has determined that the third data set, Student Data Collection, is “optional” for both LMS environments and content (from a conformance testing point of view).

6.7.1 “CMI” LMS to Content (AU) Data Model

Table 6-2 represents a subset of the data model defined in the AICC CMI 3.0.1 document, which also contains more complete definitions for each term. A more complete version of this table is in Section B.4 of Appendix B. All element names are preceded by “cmi” to identify their membership in the AICC “CMI” data model.

This table summarizes the data that an assignable unit may request or “get” from the LMS. All these elements are obtained using the `LMSGetValue(elementName)` API call.

The “LMS Obligation” column represents the obligations of the LMS—not the learning content. All data elements are optional for content assignable units. Assignable units are required only to use *Initialize()* and *Finish()*. They are not required to use *LMSSetValue()* or *LMSGetValue()*. This is why all data model elements are “optional” for content.

Table 6-2. Subset of the “CMI” LMS to Content (AU) Data Model

Element Name	“CMI” DATA MODEL - Contextualized Definition LMS to Content (AU) Communications	LMS Obligation
core	Information required to be furnished by all CMI systems. What all lessons may depend upon at start up. <code>LMSGetValue(“cmi.core.children”)</code>	Man
--student_id	Unique alpha-numeric code/identifier that refers to a single user of the CMI system. <code>LMSGetValue(“cmi.core.student_id”)</code>	Man
--student_name	Normally, the official name used for the student on the course roster. A complete name, not just a first name. <code>LMSGetValue(“cmi.core.student_name”)</code>	Man
--lesson_location	This corresponds to the lesson exit point passed to the CMI system the last time the student experienced the lesson. <code>LMSGetValue(“cmi.core.lesson_location”)</code>	Man
--credit	Indicates whether the student is being credited by the CMI system for his performance (pass/fail and score) in this lesson. <code>LMSGetValue(“cmi.core.credit”)</code>	Man
--lesson_status	This is the current student status as determined by the CMI system, and sent to the lesson when it is launched. <code>LMSGetValue(“cmi.core.lesson_status”)</code>	Man
--entry	Indication of whether the student has been in the lesson before. <code>LMSGetValue(“cmi.core.entry”)</code>	Man
--score	Indication of the performance of the student during his last attempt on the lesson. <code>LMSGetValue(“cmi.core.score_children”)</code>	Man
-- --raw	Numerical representation of student performance in lesson. May be unprocessed raw score. <code>LMSGetValue(“cmi.core.score.raw”)</code>	Man
-- --max	The maximum score or total number that the student could have achieved. <code>LMSGetValue(“cmi.core.score.max”)</code>	Opt
-- --min	The minimum score that the student could have achieved. <code>LMSGetValue(“cmi.core.score.min”)</code>	Opt

Table 6-2. Subset of the “CMI” LMS to Content (AU) Data Model (Continued)

Element Name	“CMI” DATA MODEL - Contextualized Definition LMS to Content (AU) Communications	LMS Obligation
--total_time	Accumulated time of all the student sessions in the lesson. LMSGetValue(“cmi.core.time”)	Man
--lesson_mode	Identification of student-related information that may be used to change the behavior of the lesson. LMSGetValue(“cmi.core.lesson_mode”)	Opt
suspend_data	Unique information generated by the lesson during previous uses, that is needed for the current use. LMSGetValue(“cmi.suspend_data”)	Man
launch_data	Unique information generated at the lesson’s creation that is needed for every use. LMSGetValue(“cmi.launch_data”)	Man
comments	Instructor comments directed at the student that the lesson may present to the student when appropriate. LMSGetValue(“cmi.comments”)	Opt
evaluation	Assignable units may be able to generate detailed student-performance/lesson-evaluation information. This category identifies if this functionality is supported by the LMS. LMSGetValue(“cmi.evaluation._children”)	Opt
--course_id	Alpha numeric sequence that provides a unique label for a course. LMSGetValue(“cmi.evaluation.course_id “)	Opt
--comments	Identifies if the student’s comments on a lesson can be collected and made available by the LMS in a separate file. LMSGetValue(“cmi.evaluation.comments.”)	Opt
--interactions	Identifies what detailed information of a student’s interactions in a lesson can be collected. LMSGetValue(“cmi.evaluation.interactions._children”)	Opt
--objectives_status	Identifies what detailed information on lesson objectives can be collected. LMSGetValue(“cmi.evaluation.objectives_status._children”)	Opt
--path	Identifies what detailed information can be collected on the path through the lesson taken by the student. LMSGetValue(“cmi.evaluation.path._children”)	Opt
--performance	Identifies what detailed information can be collected, on the student’s performance in complex scenarios, such as simulations. LMSGetValue(“cmi.evaluation.performance._children”)	Opt
objectives	Identifies how the student has performed on individual objectives covered in the lesson. LMSGetValue(“cmi.objectives._count”) LMSGetValue(“cmi.objectives._children”)	Opt
--id	A developer-defined, lesson-specific identifier for an objective. LMSGetValue(“cmi.objectives.n.id”)	Opt
--scores	The score obtained by the student after each attempt to master the objective. LMSGetValue(“cmi.objectives.n.scores._count”)	Opt
-- --raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score. LMSGetValue(“cmi.objectives.n.scores.n.raw”)	Opt
-- --max	The maximum score or total number that the student could have achieved. LMSGetValue(“cmi.objectives.n.scores.n.max”)	Opt
-- --min	The minimum score that the student could have achieved. LMSGetValue(“cmi.objectives.n.scores.n.min”)	Opt
--statuses	The status obtained by the student after each attempt to master the objective. LMSGetValue(“cmi.objectives.n.status.n”)	Opt

Table 6-2. Subset of the “CMI” LMS to Content (AU) Data Model (Continued)

Element Name	“CMI” DATA MODEL - Contextualized Definition LMS to Content (AU) Communications	LMS Obligation
student_data	Information to support customization of a lesson based on a student's performance. LMSGetValue("cmi.student_data.children")	Opt
--attempt_number	Number of times the student has been in, or previously used the lesson. LMSGetValue("cmi.student_data.attempt_number")	Opt
--mastery_score	The passing score, as determined outside the lesson. LMSGetValue("cmi.student_data.mastery_score")	Opt
--max_time_allowed	The amount of time the student is allowed to have in the current attempt on the lesson. LMSGetValue("cmi.student_data.max_time_allowed")	Opt
--time_limit_action	What the lesson is to do when the max time allowed is exceeded. LMSGetValue("cmi.student_data.time_limit_action")	Opt
--attempt_records	Student's performance after previous times in the lesson. LMSGetValue("cmi.student_data.attempt_records.children") LMSGetValue("cmi.student_data.attempt_records.count")	Opt
-- --lesson_scores	The score obtained by the student after each previous attempt. LMSGetValue("cmi.student_data.attempt_records.n.lesson_score")	Opt
-- --lesson_statuses	Indication of the status of the lesson after each attempt. LMSGetValue("cmi.student_data.attempt_records.n.lesson_status")	Opt
student_demographics	Student attributes possessed before entering the course. LMSGetValue("cmi.student_demographics.children")	Opt
--city	Portion of student's current address. LMSGetValue("cmi.student_demographics.city")	Opt
--class	A predefined training group to which a student belongs. LMSGetValue("cmi.student_demographics.class")	Opt
--company	Student's place of employment. LMSGetValue("cmi.student_demographics.company")	Opt
--country	Portion of student's current address. LMSGetValue("cmi.student_demographics.country")	Opt
--experience	Information on the student's past that might be required by a lesson to determine what to present, or what presentation strategies to use. LMSGetValue("cmi.student_demographics.experience")	Opt
--familiar_name	An informal title that may be used to address the student. LMSGetValue("cmi.student_demographics.familiar_name")	Opt
--instructor_name	Name of the person responsible for the student's understanding of the material in the lesson. LMSGetValue("cmi.student_demographics.instructor_name")	Opt
--title	Title of the position or the degree currently held by the student. LMSGetValue("cmi.student_demographics.title")	Opt
--native_language	The language used in the student's country of origin. LMSGetValue("cmi.student_demographics.native_language")	Opt
--state	Segment of a country, also called province, district, canton, and so forth. LMSGetValue("cmi.student_demographics.state")	Opt
--street_address	Portion of student's current address. LMSGetValue("cmi.student_demographics.street_address")	Opt
--telephone	Telephone number of a student. LMSGetValue("cmi.student_demographics.telephone")	Opt
--years_experience	Number of years the student has performed in current or similar position. LMSGetValue("cmi.student_demographics.years_experience")	Opt

Table 6-2. Subset of the “CMI” LMS to Content (AU) Data Model (Continued)

Element Name	“CMI” DATA MODEL - Contextualized Definition LMS to Content (AU) Communications	LMS Obligation
--years_experience	Number of years the student has performed in current or similar position. LMSGetValue(“cmi.student_demographics.years_experience”)	Opt
student_preference	Student-selected options that are appropriate for subsequent lessons. LMSGetValue(“cmi.student_preference.children”)	Opt
--audio	Sound on/off and volume control. LMSGetValue(“cmi.student_preference.audio”)	Opt
--language	Identifies in what language the information should be delivered. LMSGetValue(“cmi.student_preference.language”)	Opt
--lesson_type	Indicates suitability of preferences to current lesson. LMSGetValue(“cmi.student_preference.lesson_type”)	Opt
--speed	Pace of content delivery. LMSGetValue(“cmi.student_preference.speed”)	Opt
--text	Written content visibility control. LMSGetValue(“cmi.student_preference.text”)	Opt
--text_color	Written content foreground and background hue. LMSGetValue(“cmi.student_preference.text_color”)	Opt
--text_location	Position of text window on the screen. LMSGetValue(“cmi.student_preference.text_location”)	Opt
--text_size	Magnitude of the written content characters on screen. LMSGetValue(“cmi.student_preference.text_size”)	Opt
--video	Motion picture tint and brightness on the screen. LMSGetValue(“cmi.student_preference.video”)	Opt
--windows	Size and location of video, help, glossary, and so forth windows. LMSGetValue(“cmi.student_preference.n.windows”)	Opt

6.7.2 “CMI” Content (AU) to LMS Data Model

Table 6-3 represents a subset of the data model defined in the AICC CMI 3.0.1 document, which also contains more complete definitions for each term. A more complete version of this table is provided in Section B.5 of Appendix B. All element names are preceded by “cmi” to identify their membership in the “CMI” data model.

This table summarizes the data that an assignable unit may send to the LMS. All these elements are sent to the LMS using the `LMSSetValue(elementName)` API call.

The “LMS Obligation” column represents the obligations of the LMS, not the learning content. All data elements are optional for content assignable units. Assignable units are required only to use `Initialize()` and `Finish()`. They are not required to use `LMSSetValue()` or `LMSGetValue()`. This is why all data model elements are “optional” for content.

Table 6-3. “CMI” Content (AU) to LMS Data Model

Element Name	“CMI” DATA MODEL - Contextualized Definition Content (AU) to LMS Communications	LMS Obligation
core	Information required by the CMI system to function.	Man
--lesson_location	This identifies the point where the student leaves the lesson. LMSSetValue(“cmi.core.lesson_location”, value)	Man

Table 6-3. “CMI” Content (au) to LMS Data Model

Element Name	“CMI” DATA MODEL - Contextualized Definition Content (AU) to LMS Communications	LMS Obligation
--lesson_status	This is the student status when he leaves the lesson. LMSSetValue(“cmi.core.lesson_status”, value)	Man
--exit	An indication of how or why the student left the lesson. LMSSetValue(“cmi.core.exit”, value)	Man
--score	Indication of the performance of the student during his time in the lesson.	Man
-- --raw	Numerical representation of student performance in lesson. May be unprocessed raw score. LMSSetValue(“cmi.core.score.raw”, value)	Man
-- --max	The maximum score or total number that the student could have achieved. LMSSetValue(“cmi.core.score.max”)	Opt
-- --min	The minimum score that the student could have achieved. LMSSetValue(“cmi.core.score.min”)	Opt
--session_time	Time spent in the lesson during the session that is ending. LMSSetValue(“cmi.core.time”, value)	Man
suspend_data	Unique information generated by the lesson, that is needed for future uses. Passed to the CMI system to hold and to return the next time the student starts this lesson. LMSSetValue(“cmi.suspend_data”, value)	Man
comments	Student’s written remarks recorded during the current use of the lesson. LMSSetValue(“cmi.comments.n “, value)	Opt
objectives	Identifies how the student has performed on individual objectives covered in the lesson.	Opt
--id	A developer defined, lesson-specific identifier for an objective. LMSSetValue(“cmi.objectives.n.id”, value)	Opt
--scores	The score obtained by the student after each attempt to master the objective.	Opt
-- --raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score. LMSSetValue(“cmi.objectives.n.scores.n.raw”. value)	Opt
-- --max	The maximum score or total number that the student could have achieved. LMSSetValue(“cmi.objectives.n.scores.n.max”, value)	Opt
-- --min	The minimum score that the student could have achieved. LMSSetValue(“cmi.objectives.n.scores.n.min”, value)	Opt
--statuses	The status obtained by the student after each attempt to master the objective. LMSSetValue(“cmi.objectives.n.status.n”, value)	Opt
student_data	Information on student performance for each attempt on a selected segment of the lesson without leaving the lesson.	Opt
--tries_during_lesson	Total number of efforts to complete the lesson or selected segment. LMSSetValue(“cmi.student_data.tries_during_lesson”, value)	Opt
--tries	Data related to each try.	Opt
-- --score	The score at the completion of each attempt.	Opt
-- -- --raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score. LMSSetValue(“cmi.student_data.tries.n.score.raw”. value)	Opt
-- -- --max	The maximum score or total number that the student could have achieved. LMSSetValue(“cmi.student_data.tries.n.score.max”. value)	Opt

Table 6-3. “CMI” Content (au) to LMS Data Model (Continued)

Element Name	“CMI” DATA MODEL - Contextualized Definition Content (AU) to LMS Communications	LMS Obligation
-- --min	The minimum score that the student could have achieved. LMSSetValue(“cmi.student_data.tries.n.score.min”, value)	Opt
-- --status	The status of the lesson or segment after each attempt. LMSSetValue(“cmi.student_data.tries.n.status “, value)	Opt
-- --time	Length of time required for each attempt on a lesson or segment. LMSSetValue(“cmi.student_data.tries.n.time “, value)	Opt
student_preferences	Student-selected options that are appropriate for subsequent lessons.	Opt
--language	Identifies in what language the information should be delivered. LMSSetValue(“cmi.student_preference.language”, value)	Opt
--lesson_type	Indicates suitability of preferences to current lesson. LMSSetValue(“cmi.student_preference.lesson_type”, value)	Opt
--speed	Pace of content delivery. LMSSetValue(“cmi.student_preference.speed”, value)	Opt
--text	Written content visibility control. LMSSetValue(“cmi.student_preference.text”, value)	Opt
--text_color	Written content foreground and background hue. LMSSetValue(“cmi.student_preference.text_color”, value)	Opt
--text_location	Position of text window on the screen. LMSSetValue(“cmi.student_preference.text_location”, value)	Opt
--text_size	Magnitude of the written content characters on screen. LMSSetValue(“cmi.student_preference.text_size”, value)	Opt
--video	Motion picture tint and brightness on the screen. LMSSetValue(“cmi.student_preference.video”, value)	Opt
--windows	Size and location of video, help, glossary, etc. windows. LMSSetValue(“cmi.student_preference.n.windows”, value)	Opt

6.7.3 Student Data Collection

Table 6-4 represents a subset of the data model defined in the AICC CMI 3.0.1 document, which also contains more complete definitions for each term. A more complete version of this table is provided in Section B.6 of Appendix B. All element names are preceded by “cmi” to identify their membership in the “CMI” data model.

This table summarizes the data that an assignable unit may send to the LMS related to interactions with the student. All these element values are sent using the `LMSSetValue(elementName)` API call.

The LMS obligation column represents the obligations of the LMS, not the learning content. All data elements are optional for content assignable units. Assignable units are required only to use *Initialize()* and *Finish()*; they are not required to use *LMSSetvalue()* or *LMSSetValue()*. This is why all data model elements are “optional” for content.

This entire category of the data model is optional.

Table 6-4. Student Data Collection Data Model

Element Name	Student Data Collection Table Contextualized Definition	LMS Obligation
lesson_id	Alphanumeric label supplied by the developer. LMSSetValue("cmi.evaluation.lesson_id", value)	Opt
date	The calendar day on which the data is created. LMSSetValue("cmi.evaluation.date", value)	Opt
comments	Freeform feedback from the student. More structured representation than the comments in the content to LMS table.	
--time	Indication of when the comment is made. LMSSetValue("cmi.evaluation.comments.n.time ", value)	Opt
--location	Indication of where in the content the comment is made. LMSSetValue("cmi.evaluation.comments.n.location ", value)	Opt
--content	The recorded statement of a student. LMSSetValue("cmi.evaluation.comments.n.content ", value)	Opt
interactions	A recognized and recordable input or group of inputs from the student to the computer	
--id	Unique alphanumeric label created by the content developer. LMSSetValue("cmi.interactions.n.id ", value)	Opt
--objective_ids	Indication of any objectives associated with the interaction. LMSSetValue("cmi.interactions.n.objective_ids.n", value)	Opt
--time	Indication of when the interaction is available to the student. LMSSetValue("cmi.interactions.n.time ", value)	Opt
--type	Indication of which category of interaction is recorded. LMSSetValue("cmi.interactions.n.type ", value)	Opt
--responses	Expected student feedback in the interaction.	
-- --description	Definition of possible student response. LMSSetValue("cmi.interactions.n.response.n.description", value)	Opt
-- --value	How the system judges the described response. LMSSetValue("cmi.interactions.n.response.n.value", value)	Opt
--weighting	Factor that is used to identify the relative importance of one interaction compared to another. LMSSetValue("cmi.interactions.n.weighting ", value)	Opt
--student_response	Description of the computer-measurable action of a student in an interaction. LMSSetValue("cmi.interactions.n.student_response ", value)	Opt
--result	Judgment of the student's response. LMSSetValue("cmi.interactions.n.result ", value)	Opt
--latency	The time from the presentation of the stimulus to the completion of the measurable response. LMSSetValue("cmi.interactions.n.latency ", value)	Opt
objectives	Information about a student's performance on content objectives. The only additional data that are not described in normal Content to LMS communication is mastery_time.	
paths	Description of the sequence of events the student experienced in the content.	
--location_id	Identification of where the student is in the content. LMSSetValue("cmi.path.n.location_id", value)	Opt
--time	Indication of when the student entered the content segment. LMSSetValue("cmi.path.n.time", value)	Opt

Table 6-4. Student Data Collection Data Model (Continued)

Element Name	Student Data Collection Table Contextualized Definition	LMS Obligation
--status	A record of the student's performance in a segment each time he leaves that element LMSSetValue("cmi.path.n.status", value)	Opt
--why_left	The reason a student departed an element in the content. LMSSetValue("cmi.path.n.why_left", value)	Opt
--time_in_element	How long the student spent in the element. LMSSetValue("cmi.path.n.time_in_element", value)	Opt

6.7.4 "CMI" Data Types and Controlled Vocabularies

A data type definition exists for each element in the CMI data model. Section B.7 of Appendix B summarizes the data type definitions. Sections B.4, B.5, and B.6 define the data type for each of the data model elements. These definitions define how the API and data model must be implemented.

In addition to data types, some data model elements are predefined with bounded vocabularies of possible values. Table 6-5 summarizes the vocabulary type and values (from Section B.7, AICC CMI 3.0.1 Appendix B). Definitions for each vocabulary are contained in Sections 5.1 and 5.2 of the AICC CMI 3.0.1 specification.

Table 6-5. Vocabulary Type and Values

Vocabulary Type	Members of Vocabulary	
Mode	normal	review
	browse	
Status	passed	completed
	failed	incomplete
	browsed	not attempted
Exit	time-out	suspend
	logout	
Why-left	student selected	lesson directed
	exit	directed departure
Credit	credit	no credit
Entry	ab-initio	resume
Time Limit Action	exit	continue
	message	no message
Interaction	true-false	multiple choice
	fill in the blank	matching
	simple performance	likert
	sequencing	unique
	numeric	
Result	correct	wrong
	unanticipated	neutral
	x.x (CMIDecimal)	

6.8 Conformance Testing

Three things need to be tested for runtime environment conformance with the SCORM: support of launch, correct implementation of the API, and correct usage of the data model. The responsibilities for the LMS and content are different.

Testing an LMS (or course authoring environment) for conformance:

- Can the LMS launch a known conforming course assignable unit?
- Does the LMS fully support the mandatory API functionality?
- Does the LMS support the mandatory and optional data elements in the data model?

Testing content for conformance:

- Can the content be launched by a known conforming LMS test environment?
- Does the content correctly implement the minimum API functionality [i.e., *initialize()* and *terminate()*]?
- Does the content correctly implement other API calls (if any)?
- Does the content exchange conforming data over the API?

Detailed test criteria and test software are expected to be developed and included in a future version of this document.

7. Metadata

7.1 Overview

Metadata—data about data—for learning content, have been under development within several national and international organizations over the past few years. The purpose of metadata is to provide a common means to describe things (electronically) so that “learning objects” (however they are defined) can be self defined, searched, and found. Learning content is only one area of metadata application. Metadata is also actively being developed in all aspects of Web-based content and commerce.

ADL has looked to the IEEE LOM subgroup and the IMS project as the harmonizing bodies that are defining metadata specifically for *learning content*. These groups, which have been working collaboratively over the past few years, have developed a core specification to which this document refers.

The IMS and IEEE specifications (<http://www.imsproject.org/metadata> and <http://ltsc.ieee.org/doc/wg12/WD3>) define a standard “dictionary” of metadata element definitions along with recommendations for “best practices” and XML bindings (critically important for implementation). These documents, however, do not specifically propose how individual communities of users might elect to apply these metadata definitions to their content models.

The ADL SCORM references and abides by the IEEE/IMS definitions and goes one step further to apply these definitions to the three components of the SCORM model: raw media, content, and external course. This mapping of standardized definitions from IEEE/IMS to the SCORM model provides the missing link between general specifications and specific-content models. Many thanks go to Wayne Hodgins, Tom Wason, Thor Anderson, and Steve Griffin for their efforts in establishing these key specifications.

The following sections define the SCORM application of IEEE/IMS definitions.

7.2 Definitions of SCORM Metadata Elements

The following definitions “map” how metadata are to be applied to the SCORM “content model.” In all cases, the IEEE/IMS documents are referenced and applied to the various components of the ADL SCORM model.

7.2.1 Raw Media Metadata

Raw media metadata can be applied to so-called “raw media” assets (e.g., illustrations, documents, or media streams) that provide descriptive information about the raw media independent of courseware content. These metadata are used to facilitate reuse and discoverability—principally during content creation of media elements within, for example, a media repository.

- Metadata that describe raw media elements in a noncontext-specific way
- Information that can be searched externally, such as media asset title, description, date of creation, and version
- Information that can be used to create a searchable repository of sharable media elements.

7.2.2 Content Metadata

Content metadata can be applied to Web-based content, which provides descriptive information about the content independent of a particular course. These metadata are used to facilitate reuse and discoverability of content within, for example, a content repository.

- Metadata that describe a [sharable] “chunk” of content
- Content metadata not related to a specific course structure (i.e., context-independent metadata)
- Information that can be searched externally, such as content asset title, description, and version.

7.2.3 External Course Metadata

External course metadata are external metadata that describe a course package for searching (enabling discoverability) within a courseware repository and for providing descriptive information about the course.

- Information about a course as a whole that describes what it is for, who can use it, who controls it, and so forth
- Information that can be searched externally, such as the course title, course description, and version.

7.2.4 SCO Structure Format (Assignment Hierarchy) Metadata

Metadata within a representation (such as XML) of a course structure that can be used to define all the course elements, structure, and external references necessary to move a course from one LMS environment to another.

- Metadata described with the specific assignments at different levels within the lesson plan hierarchy
- Course element metadata within a particular course hierarchy that is context specific to that course hierarchy.

7.3 SCORM Metadata Mapping

Table 7-1 lists each of the IEEE LOM elements as defined in the IEEE LTSC Working Group P1484.12 WD3 document. This document is accessible at Internet address <http://ltsc.ieee.org/doc/wg12/WD3> and is included as Appendix C of this document. The right-three columns define how each metadata element is to be applied to the SCORM and which elements should be used when building metadata records for raw media, content, and entire courses.

Table 7-1. IEEE LOM Elements

Note for Table 7-1: *Blank = not used; M = mandatory; O = optional; SEL = (IMS) Standard Extension Library.*

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
1. General	This category groups the general information that describes this resource as a whole.				
1.1 Identifier	A globally unique label that identifies this resource. This is reserved and shall not be used, because there is no specified method for the creation of a globally unique identifier.	RE-SERVED			
1.2 Title	Name given to this resource. This element may be an already existing one or it may be created by the indexer ad hoc. This element shall correspond with the Dublin Core element DC.Title.	CORE	M	M	M
1.3 Catalog Entry	This subcategory defines an entry within a catalogue (i.e., a listing identification system) assigned to this resource. This subcategory is intended to describe this resource according to some known cataloging system so that it may be externally searched for and located according to the methodology of the specified system. This subcategory may be used as a functional replacement for the element 1.1: Identifier, as that is currently reserved. In this way, it shall be used to store the Dublin Core element DC.Identifier. One of the catalog entries can be generated automatically by the tool.				
1.3.1 Catalog	The name of the catalogue (i.e., listing identification system).	CORE	O	M	M

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
1.3.2 Entry	Actual string value of the entry within the catalogue (i.e., listing identification system).	CORE	O	M	M
1.4 Language	The primary human language used within this resource to communicate to the intended user. An indexation tool may provide a useful default. This element shall correspond with the Dublin Core element DC.Language.	CORE	O	O	O
1.5 Description	A textual description of the content of this resource. This element shall correspond with the Dublin Core element DC.Description.	CORE	M	M	M
1.6 Keywords	Keywords or phrases describing this resource. This element should not be used for characteristics that can be described by other elements.	SEL	O	M	M
1.7 Coverage	The span or extent of such things as time, culture, geography, or region that applies to this resource. This element shall correspond with the Dublin Core element DC.Coverage.	SEL			
1.8 Structure	Underlying organizational structure of this resource. Restricted vocabulary: 1=User_defined 2=See_classification 3=Collection 4=Mixed 5=Linear 6=Hierarchical 7=Networked 8=Branched 9=Parceled 10=Atomic.	SEL			

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
1.9 Aggregation Level	The functional granularity of this resource. Level 0 means smallest level of aggregation (e.g., raw media data or fragments). Level 1 refers to a collection of atoms (e.g., an HTML document with some embedded pictures or a lesson). Level 2 indicates a collection of level 1 resources (e.g., a 'web' of HTML documents, with an index page that links the pages together or a unit). Finally, level 3 refers to the largest level of granularity (e.g., a course).	SEL		O	?TBD
2 LifeCycle	This category describes the history and current state of this resource and those who have affected this resource during its evolution.				
2.1 Version	The edition of this resource.	CORE	O	M	M
2.2 Status	The state or condition this resource is in. Restricted vocabulary: restricted vocabulary: 1=User_defined 2=See_classification 3=Draft 4=Final 5=Revised 6=Unavailable.	SEL	O	M	M
2.3 Contribute	This subcategory describes those people or organizations that have affected the state of this resource during its evolution (includes creation, edits and publication). This subcategory is different from 3.3: Contribute.				

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
2.3.1 Role	Kind of contribution. This element should include exactly one instance of Author Best practice list: 1=User_defined 2=See_classification 3=Author 4=Publisher 5=Unknown 6=Initiator 7=Terminator 8=Validator 9=Editor 10=Graphical Designer 11=Technical Implementer 12=Content Provider 13=Technical Validator 14=Educational Validator 15=Script Writer 16=Instructional Designer. It is recommended that exactly one instance of Author exists.	CORE	0	0	0
2.3.2 Entity	The identification of and information about the people or organizations contributing to this resource, most relevant first. If 2.3.1:LifeCycle.Contribute.Role equals Author, then the entity should be a person and this element shall correspond with the Dublin Core element DC.Creator. If 2.3.1:LifeCycle.Contribute.Role equals Publisher, then the entity should be an organization and this element shall correspond with the Dublin Core element DC.Publisher. If 2.3.1:LifeCycle.Contribute.Role is not equal to Author or Publisher, then this element shall correspond with the Dublin Core element DC.Contributor. If the entity is an organization, then it should be a university department, company, agency, institute, and so forth under whose responsibility the contribution was made.	CORE	0	0	0

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
2.3.3 Date	The date of the contribution.	CORE	O	O	O
3 MetaMetaData	This category describes the specific information about this metadata record itself (rather than the resource that this record describes). This category describes such things as who created this metadata record, how, when and with what references. This is <i>not</i> the information that describes the resource itself.				
3.1 Identifier	A globally unique label that identifies this metadata record. This is reserved and shall not be used, as there is no specified method for the creation of a globally unique identifier.	RE-SERVED			
3.2 Catalog Entry	This subcategory defines an entry within a catalogue (i.e., listing identification system), given to the metadata instance. This category is intended to describe this metadata instance according to some known cataloging system so that it may be externally searched for and located according to that system. This element may be used as a functional replacement for the currently reserved element 3.1:MetaMetaData.Identifier. One of the catalog entries may be generated automatically by the tool.	SEL			
3.2.1 Catalog	The name of the catalogue (i.e., listing identification system). Generally system generated.	SEL			
3.2.2 Entry	Actual string value of the entry in the catalogue. This element is usually generated by the system.	SEL			
3.3 Contribute	This subcategory describes those people or organizations that have affected the state of this metadata instance during its evolution (includes creator and validator). This element is different from 2.3:Lifecycle.Contribute.	SEL			

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
3.3.1 Role	Kind of contribution. Exactly one instance of creator should exist. Open vocabulary with best practice list: 1=User_defined 2=See_classification 3=Creator 4=Validator. It is recommended that exactly one instance of creator exists.	SEL			
3.3.2 Entity	The identification of and information about the people or organizations contributing to this metadata instance, most relevant first.	SEL			
3.3.3 Date	The date of the contribution.	SEL			
3.4 Metadata Scheme	The name and version of the authoritative specification used to create this metadata instance. This element may be user selectable or system generated. If multiple values are provided, then the metadata instance shall conform to multiple metadata schemes.	CORE	M	M	M
3.5 Language	Language of this metadata instance. This is the default language for all LangString values in this metadata instance.	CORE			
4 Technical	This category describes the technical requirements and characteristics of this resource.				
4.1 Format	Technical data type of this resource. This element shall be used to identify the software needed to access the resource. Restricted vocabulary: MIME type or "non-digital." Can be used to identify the software needed to access the resource. For example, video/mpeg, application/x-toolbook, text/html.	CORE	M	M	M
4.2 Size	The size of the digital resource in bytes. Only the digits '0'...'9' should be used. The unit is bytes, not MBytes, GB, and so forth. This element shall refer to the actual size of this resource and not to the size of a compressed version of this resource.	SEL	O	O	O

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
4.3 Location	A string that is used to access this resource. It may be a location (e.g., Universal Resource Locator), or a method that resolves to a location (e.g., Universal Resource Identifier). Preferable Location first. This is where the learning resource described by this metadata instance is physically located (e.g., http://host/id)	CORE	M	M	M
4.4 Requirements	This subcategory describes the technical capabilities required in order to use this resource. If there are multiple requirements, then all are required, i.e. the logical connector is AND.				
4.4.1 Type	The technology required to use this resource (i.e., hardware, software, network, and so forth). Open vocabulary with best practice: 1=User_defined 2=See_classification 3=Operating System 4=Browser.	SEL	O	O	O

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
4.4.2 Name	Name of the required technology to use this resource. The value for this element may be derived from 4.1:Technical.Format automatically (e.g., "video/mpeg" implies "Multi-OS"). If Type = 'Operating System,' then best practice list: 1=User_defined 2=See_classification 3=PC-DOS 4=MS-Windows 5=MacOS 6=Unix 7=Multi-OS 8=Other 9=None. If Type = 'Browser,' then best practice list: 10=Any 11=Netscape Communicator 12=Microsoft Internet Explorer 13=Opera. If other type, then open vocabulary.	SEL	O	O	O
4.4.3 Minimum Version	Lowest possible version of the required technology to use this resource.	SEL	O	O	O
4.4.4 Maximum Version	Highest version of the technology known to support the use of this resource.	SEL	O	O	O
4.5 Installation Remarks	Description on how to install this resource.	SEL	O	O	O
4.6 Other Platform Requirements	Information about other software and hardware requirements.	SEL	O	O	O
4.7 Duration	Time a continuous resource takes when played at intended speed. This is especially useful for sounds, movies or animations.	SEL	O	O	O

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
5 Educational	This category describes the key educational or pedagogic characteristics of this resource. This is the pedagogical information essential to those involved in achieving a quality learning experience. The audience for this metadata includes teachers, managers, authors, and learners.				
5.1 Interactivity Type	The flow of interaction between this resource and the intended user. In an <i>expositive</i> resource, the information flows mainly from this resource to the learner. Expositive documents are typically used for learning- by- reading. In an <i>active</i> resource, information also flows from the learner to this resource. Active documents are typically used for learning-by- doing. Activating links to navigate in hypertext documents is not considered as an information flow. Thus, hypertext documents are expositive. Restricted vocabulary: 1=User_defined 2=See_classification 3=Active 4=Expositive 5=Mixed 6=Undefined. Expositive documents include essays, video clips, all kinds of graphical material and hypertext documents. Active documents include simulations, questionnaires and exercises.	SEL			

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
5.2 Learning Resource Type	Specific kind of resource, most dominant kind first. This element shall correspond with the Dublin Core element "Resource Type." The vocabulary is adapted for the specific purpose of <i>learning</i> objects. Open vocabulary with best practice: 1=User_defined 2=See_classification 3=Exercise 4=Simulation 5=Questionnaire 6=Diagram 7=Figure 8=Graph 9=Index 10=Slide 11=Table 12=Narrative Text 13=Exam 14=Experiment 15=ProblemStatement 16=SelfAssesment.	SEL	0	0	0
5.3 Interactivity Level	This element shall define the degree of interactivity between the end user and this resource, with 0 defined as "Very Low," 1 defined as "Low," 2 defined as "Medium," 3 defined as "High," and 4 defined as "Very High."	SEL			
5.4 Semantic Density	This element shall define a subjective measure of this resource's usefulness as compared to its size or duration, with 0 defined as "Very Low," 1 defined as "Low," 2 defined as "Medium," 3 defined as "High," and 4 defined as "Very High."	SEL			

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
5.5 Intended End User Role	Principal user(s) for which this resource was designed, most dominant first. A learner works with a resource in order to learn something. An author creates or publishes a resource. A manager manages the delivery of this resource (e.g., a university or college). The document for a manager is typically a curriculum. Restricted vocabulary: 0=Teacher 1=Author 2=Learner 3=Manager.	SEL			
5.6 Context	The principal environment within which the learning and use of this resource is intended to take place. Open vocabulary with best practice: 1=User_defined 2=See_classification 3=Primary Education 4=Secondary Education 5=Higher Education 6=University First Cycle 7=University Second Cycle 8=University Postgrade 9=Technical School First Cycle 10=Technical School Second Cycle 11=Professional Formation 12=Continuous Formation 13=Vocational Training 14=Other.	SEL			

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
5.7 Typical Age Range	Age of the typical intended user. This element shall refer to developmental age, if that would be different from chronological age. The age of the learner is important for finding resources, especially for school age learners and their teachers. When applicable, the string should be formatted as minage-maxage or minage-. (This is a compromise between adding three subfields (minAge, maxAge and description) and having just a free text field.) Various reading age schemes, IQ's or developmental age measures should be represented through the 9:Classification category	SEL			
5.8 Difficulty	This element defines how hard it is to work through this resource for the typical target audience, with 0 defined as "Very Easy," 1 defined as "Easy," 2 defined as "Medium," 3 defined as "Difficult," and 4 defined as "Very Difficult."	SEL			
5.9 Typical Learning Time	Approximate or typical time it takes to work with this resource.	SEL		O	O
5.10 Description	Comments on how this resource is to be used (e.g., teacher guidelines that come with a textbook).	SEL			
5.11 Language	The human language used by the typical intended user of the resource. LanguageID = Langcode('-'Subcode)*, with Langcode a two-letter language code as defined by ISO639 and subcode a country code from ISO3166 (e.g., "en", "en-GB", "de", "fr-CA", "it").	SEL			
6 Rights	This category describes the intellectual property rights and conditions of use for this resource. The intent is to reuse results of ongoing work in the Intellectual Property Right and e-commerce communities. This category currently provides the absolute minimum level of detail only.				

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
6.1 Cost	Whether use of the resource requires payment. Restricted vocabulary: 1=User_defined 2=See_classification 3=yes 4=no.	CORE	M	M	M
6.2 Copyright and Other Restrictions	Whether copyright or other restrictions apply to the use of this resource. Restricted vocabulary: 1=User_defined 2=See_classification 3=yes 4=no.	CORE	M	M	M
6.3 Description	Comments on the conditions of use of this resource.	CORE	O	O	O
7 Relation	This category defines the relationship between this resource and other targeted resources, if any. To define multiple relationships there may be multiple instances of this category. If there is more than one target resource, each target is covered by a new relationship instance.				
7.1 Kind	Nature of the relationship between this resource and the target resource, identified by 7.2:Relation.Resource. This element shall correspond with the Dublin Core element DC.Relation. Best practice list from Dublin Core: 1=User_defined 2=See_classification 3=IsPartOf 4=HasPart 5=IsVersionOf 6=HasVersion 7=IsFormatOf 8=HasFormat 9=References 10=IsReferencedBy 11=IsBasedOn 12=IsBasisFor 13=Requires 14=IsRequiredBy.	SEL			

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
7.2 Resource	The target resource that this relationship references.				
7.2.1 Identifier	Unique Identifier of the target resource. This is reserved and shall not be used.	RE-SERVED			
7.2.2 Description	Description of the target resource.	SEL			
8 Annotation	This category provides comments on the educational use of this resource, who created this annotation and when. When multiple annotations are needed, multiple instances of this category may be used.				
8.1 Person	The person who created this annotation.	SEL			
8.2 Date	Date that this annotation was created.	SEL			
8.3 Description	The content of this annotation.	SEL			
9 Classification	This category describes where this resource is placed within a particular classification system. To define multiple classifications, there may be multiple instances of this category. If 9.1:Classification.Purpose equals Discipline, this category shall correspond with the Dublin Core element DC.Subject.				
9.1 Purpose	The purpose of classifying this resource. Open vocabulary with best practice: 1=User_defined 2=See_classification 3=Discipline 4=Idea 5=Prerequisite 6=Educational Objective 7=Accessibility Restrictions 8=Educational Level 9=Skill Level 10=Security Level.	CORE		M	M
9.2 TaxonPath	This subcategory describes a taxonomic path in a specific classification system. Each succeeding level is a refinement in the definition of the higher level. There may be different paths, in the same or different classifications, that describe the same characteristic. A taxonomy is a hierarchy of terms or phrases that are taxons.				

Table 7-1. IEEE LOM Elements (Continued)

Draft IEEE LTSC Learning Object Metadata Tag version 3.8 (11-7-99)	IEEE Explanation	I	SCORM Raw Media	SCORM Content	SCORM External Course
		MS Best Practices			
9.2.1 Source	The name of the classification system. This element may use any recognized “official” taxonomy, any user-defined taxonomy. An indexation or query tool may provide the top-level entries of a well-established classification (LOC, UDC, DDC, and so forth).	SEL			
9.2.2 Taxon	This subcategory describes a particular term within a hierarchical classification system or taxonomy. A taxon is a node that has a defined label or term. A taxon may also have an alphanumeric designation or identifier for standardized reference. Either or both the label and the entry may be used to designate a particular taxon. An ordered list of Taxons creates a taxonomic path (i.e. “taxonomic stairway”: this is a path from a more general to more specific entry in a classification). A TaxonPath shall have a depth from 1 to 9. Normal values should be defined as values between 2 and 4 (e.g., Physics/Acoustics/Instruments/Stethoscope; Medicine/Diagnostics/Instruments/Stethoscope)				
9.2.2.1 ID	The identifier of the taxon, such as a number or letter combination provided by the source of the taxonomy (e.g., 300).	SEL			
9.2.2.2 Entry	The textual label of the taxon (e.g., Social Sciences).	SEL			
9.3 Description	This is the description of the resource relative to the stated 9.1:Classification.Purpose of this specific classification, such as discipline, idea, skill level, educational objective, and so forth.	CORE		M	M
9.4 Keywords	These are the keywords and phrases descriptive of the resource relative to the stated 9.1:Classification.Purpose of this specific classification, such as accessibility, security level, and so forth, most relevant first.	CORE		M	M

7.4 Stand-Alone XML Metadata Records

It is expected that each of the three categories of SCORM metadata (raw media, content, and external course) will take the form of stand-alone XML records that conform to the

IMS *Learning Resource Meta-data XML Binding Specification*, Version 1.0 (included as Appendix D).

SCORM metadata records are expected to be valid, well-formed XML documents based on the DTD in Appendix D; however, the only metadata elements that should be used are those listed as “mandatory” or “optional” as defined in Section 7.3.

7.5 XML Schema, Namespaces, and Extensibility

The use of XML DTDs as a means to ensure conformance is understood to be highly problematic in the Internet community. DTDs fail to provide adequate mechanisms for extensions and do not guarantee interoperability.

New approaches for defining interoperable and flexible XML records are emerging from organizations such as XML Schema and name spacing being developed within the W3C (<http://www.w3c.org>). The W3C Home Page provides a reference to an XHTML version 1.0. These developments are beyond the scope of this document. It is, nonetheless, expected that the XML bindings referenced in this document that today depend on DTDs will later be converted to comply with mainstream XML practices once they are defined and adopted.

This means that XML metadata records produced using the specifications included in this document will eventually need to be converted to newer XML formats as these formats become defined. This conversion is not expected to be particularly difficult and can probably be performed semi-automatically using simple software tools.

7.6 Conformance Testing

Conformance testing of metadata records consists of verifying that a metadata record is a valid IMS/IEEE record and that it has the mandatory or optional elements for its use as specified within the SCORM for raw media, content, and external course metadata records.

7.7 XML Examples

The following examples are empty XML files based on the IMS XML DTD [see Appendix D] but only contain elements that apply to the ADL SCORM for each of the three categories.

7.7.1 Empty Raw Media XML Metadata Record

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RECORD SYSTEM "IMS_MD01.dtd" >
<RECORD xmlns="http://www.imspj.org/metadata/">
  <METAMETADATA>
    <!--Mandatory Element SCORM RMMD-->
    <METADATAScheme/>
```

```
</METAMETADATA>
<GENERAL>
  <TITLE>
    <!--Mandatory element SCORM RMMD-->
    <LANGSTRING/>
  </TITLE>
  <CATALOGENTRY>
    <!--Optional elements SCORM RMMD-->
    <CATALOGUE/>
    <ENTRY>
      <LANGSTRING/>
    </ENTRY>
  </CATALOGENTRY>
  <LANGUAGE>
    <!--Optional element SCORM RMMD-->
  </LANGUAGE>
  <DESCRIPTION>
    <!--Mandatory element SCORM RMMD-->
    <LANGSTRING/>
  </DESCRIPTION>
  <KEYWORDS>
    <!--Optional element SCORM RMMD-->
    <LANGSTRING/>
  </KEYWORDS>
</GENERAL>
<LIFECYCLE>
  <VERSION>
    <!--Optional element SCORM RMMD-->
    <LANGSTRING/>
  </VERSION>
  <STATUS>
    <!--Optional element SCORM RMMD-->
    <LANGSTRING/>
  </STATUS>
  <CONTRIBUTE>
    <!--ALL lifecycle elements optional SCORM RMMD-->
    <ROLE>
      <LANGSTRING/>
    </ROLE>
    <CENTITY>
      <!--Optional element SCORM RMMD-->
    </CENTITY>
    <DATE>
```

```

        <!--Optional element SCORM RMD-->
        <DATETIME/>
    </DATE>
</CONTRIBUTE>
</LIFECYCLE>
<TECHNICAL>
    <FORMAT>
        <!--Mandatory element SCORM RMD-->
        <LANGSTRING/>
    </FORMAT>
    <SIZE>
        <!--Optional element SCORM RMD-->
    </SIZE>
    <LOCATION>
        <!--Optional element SCORM RMD-->
    </LOCATION>
    <REQUIREMENTS>
        <TYPE>
            <LANGSTRING/>
        </TYPE>
        <!--Optional element SCORM RMD-->
        <MINIMUMVERSION>
            <!--Optional element SCORM RMD-->
        </MINIMUMVERSION>
        <MAXIMUMVERSION>
            <!--Optional element SCORM RMD-->
        </MAXIMUMVERSION>
    </REQUIREMENTS>
    <INSTALLATIONREMARKS>
        <!--Optional element SCORM RMD-->
        <LANGSTRING/>
    </INSTALLATIONREMARKS>
    <OTHERPLATFORMREQUIREMENTS>
        <LANGSTRING>
            <!--Optional element SCORM RMD-->
        </LANGSTRING>
    </OTHERPLATFORMREQUIREMENTS>
    <DURATION>
        <!--Optional element SCORM RMD-->
        <DATETIME/>
    </DURATION>
</TECHNICAL>
<EDUCATIONAL>

```

```

<LEARNINGCONTEXT>
  <!--Optional element SCORM RMMD-->
  <LANGSTRING/>
</LEARNINGCONTEXT>
</EDUCATIONAL>
<RIGHTS>
  <COST>
    <!--Mandatory element SCORM RMMD-->
    <LANGSTRING/>
  </COST>
  <COPYRIGHTOROTHERRESTRICTIONS>
    <!--Mandatory element SCORM RMMD-->
    <LANGSTRING/>
  </COPYRIGHTOROTHERRESTRICTIONS>
  <DESCRIPTION>
    <!--Optional element SCORM RMMD-->
    <LANGSTRING/>
  </DESCRIPTION>
</RIGHTS>
</RECORD>

```

7.7.2 Empty Content XML Metadata Record

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RECORD SYSTEM "IMS_MD01.dtd" >
<RECORD xmlns="http://www.imsproject.org/metadata/">
  <METAMETADATA>
    <!--Mandatory Element SCORM CONTENTMD-->
    <METADATAScheme/>
  </METAMETADATA>
  <GENERAL>
    <TITLE>
      <!--Mandatory element SCORM CONTENTMD-->
      <LANGSTRING/>
    </TITLE>
    <CATALOGENTRY>
      <!--Mandatory elements SCORM CONTENTMD-->
      <CATALOGUE/>
      <ENTRY>
        <LANGSTRING/>
      </ENTRY>
    </CATALOGENTRY>
    <LANGUAGE>

```

```

    <!--Optional element SCORM CONTENTMD-->
</LANGUAGE>
<DESCRIPTION>
    <!--Mandatory element SCORM CONTENTMD-->
    <LANGSTRING/>
</DESCRIPTION>
<KEYWORDS>
    <!--Mandatory element SCORM CONTENTMD-->
    <LANGSTRING/>
</KEYWORDS>
<AGGREGATIONLEVEL>
    <!--Optional element SCORM CONTENTMD-->
</AGGREGATIONLEVEL>
</GENERAL>
<LIFECYCLE>
    <VERSION>
        <!--Mandatory element SCORM CONTENTMD-->
        <LANGSTRING/>
    </VERSION>
    <STATUS>
        <!--Mandatory element SCORM CONTENTMD-->
        <LANGSTRING/>
    </STATUS>
    <CONTRIBUTE>
        <!--ALL lifecycle elements optional SCORM CONTENTMD-->
        <ROLE>
            <LANGSTRING/>
        </ROLE>
        <CENTITY>
            <!--Optional element SCORM CONTENTMD-->
        </CENTITY>
        <DATE>
            <!--Optional element SCORM CONTENTMD-->
            <DATETIME/>
        </DATE>
    </CONTRIBUTE>
</LIFECYCLE>
<TECHNICAL>
    <FORMAT>
        <!--Mandatory element SCORM CONTENTMD-->
        <LANGSTRING/>
    </FORMAT>
    <SIZE>

```

```

    <!--Optional element SCORM CONTENTMD-->
</SIZE>
<LOCATION type="URI">
    <!--Optional element SCORM CONTENTMD-->
</LOCATION>
<REQUIREMENTS>
    <TYPE>
        <LANGSTRING/>
    </TYPE>
    <!--Optional element SCORM CONTENTMD-->
    <MINIMUMVERSION>
        <!--Optional element SCORM CONTENTMD-->
    </MINIMUMVERSION>
    <MAXIMUMVERSION>
        <!--Optional element SCORM CONTENTMD-->
    </MAXIMUMVERSION>
</REQUIREMENTS>
<INSTALLATIONREMARKS>
    <!--Optional element SCORM CONTENTMD-->
    <LANGSTRING/>
</INSTALLATIONREMARKS>
<OTHERPLATFORMREQUIREMENTS>
    <LANGSTRING>
        <!--Optional element SCORM CONTENTMD-->
    </LANGSTRING>
</OTHERPLATFORMREQUIREMENTS>
<DURATION>
    <!--Optional element SCORM CONTENTMD-->
    <DATETIME/>
</DURATION>
</TECHNICAL>
<EDUCATIONAL>
    <LEARNINGCONTEXT>
        <!--Optional element SCORM CONTENTMD-->
        <LANGSTRING/>
    </LEARNINGCONTEXT>
    <TYPICALLEARNINGTIME>
        <!--Optional element SCORM CONTENTMD-->
        <DESCRIPTION>
            <LANGSTRING/>
        </DESCRIPTION>
    </TYPICALLEARNINGTIME>
</EDUCATIONAL>

```

```

<RIGHTS>
  <COST>
    <!--Mandatory element SCORM CONTENTMD-->
    <LANGSTRING/>
  </COST>
  <COPYRIGHTOROTHERRESTRICTIONS>
    <!--Mandatory element SCORM CONTENTMD-->
    <LANGSTRING/>
  </COPYRIGHTOROTHERRESTRICTIONS>
  <DESCRIPTION>
    <!--Optional element SCORM CONTENTMD-->
    <LANGSTRING/>
  </DESCRIPTION>
</RIGHTS>
<CLASSIFICATION>
  <PURPOSE>
    <!--Mandatory element SCORM CONTENTMD-->
    <LANGSTRING/>
  </PURPOSE>
  <DESCRIPTION>
    <!--Mandatory Element SCORM CONTENTMD-->
    <LANGSTRING/>
  </DESCRIPTION>
  <KEYWORDS>
    <!--Mandatory element SCORM CONTENTMD-->
    <LANGSTRING/>
  </KEYWORDS>
</CLASSIFICATION>
</RECORD>

```

7.7.3 External Course Metadata XML Record

External course metadata and content metadata currently have the same mandatory and optional elements. This could change during evaluation of these specifications; therefore, a placeholder is provided here. The issue at hand is that overhead associated with metadata tags may result in some elements being eliminated from content or course records. This must be determined through trial implementations.

8. Sample SCORM Code

The release of SCORM version 1.0 includes a package of samples that illustrate various aspects of this document. They are available for free and may be downloaded from <http://www.adlnet.org>.

Note: *The sample code described in this section will be updated and expanded over time. Be sure to download the most recent version and view the read me.htm file for new information. The samples described in this section may have been superseded by newer versions by the time this document has been published. You are free to use and modify these samples any way you wish. If you create new or improved samples, please send them to secretariat@adlnet.org so others may benefit from what you have learned.*

This release includes a sample LMS and content, and an XML CSF editor.

8.1 Sample LMS and Content

The sample LMS (See Figure 8-1) is intended to provide an example implementation of the concepts described in the ADL SCORM version 1.0. This sample was developed over an extremely short period of time to provide an illustration of the concepts described in the SCORM. It will be continuously expanded. The main focus during this iteration of development has been the runtime environment communication between the LMS and assignable units using the API mechanism described in the SCORM. It is *not* a complete LMS implementation.

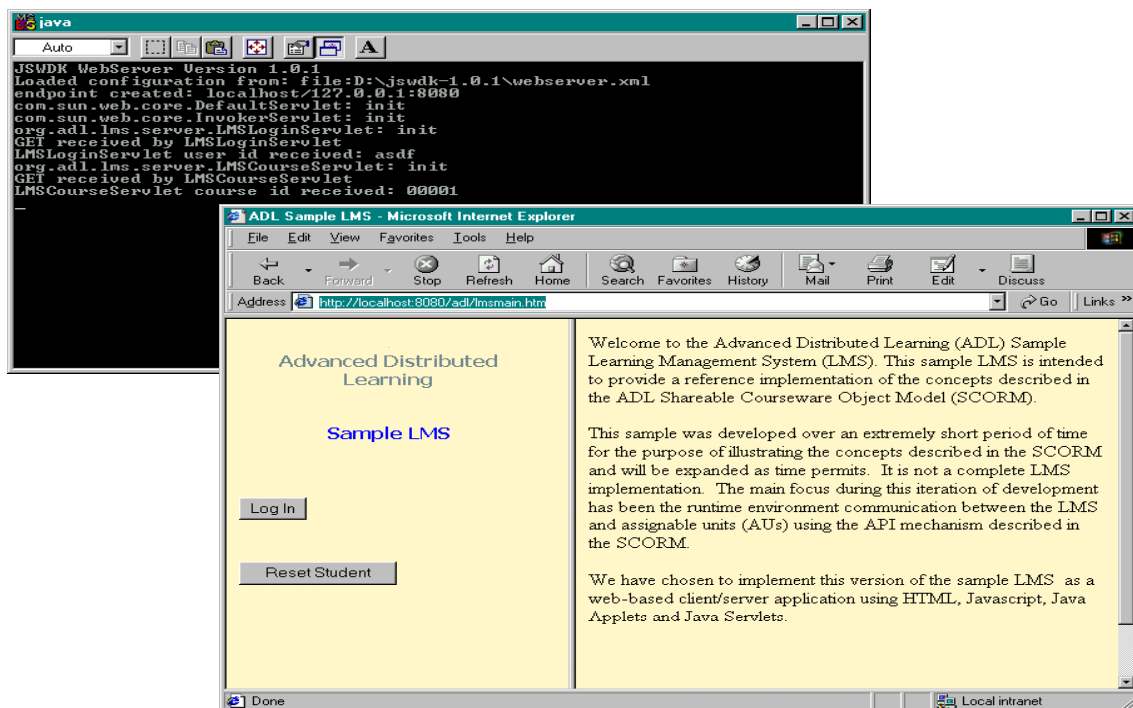


Figure 8-1. Sample LMS

Several shortcuts have been taken in this version:

- Robust exception and error handling have not been included in this sample code.
- There are a near infinite number of possible LMS implementations. This version of the sample LMS was implemented as a Web-based client/server application using HTML, JavaScript, Java applets, and Java servlets.
- This sample LMS supports only a single student and a single course with one lesson at this time.
- Concurrent access is not supported.

This sample code consists of the following components:

- **LMS Server.** This is implemented using java servlets.
- **LMS Client.** This is implemented using Java applets, HTML, and JavaScript.
- **Sample Course.** This is implemented using HTML and JavaScript.

8.1.1 LMS Server

The LMS server functionality (see Figure 8-2) is implemented using Java servlets and HTML. The servlets respond to requests from the LMS client and are responsible for CMI data model persistence, serving the student course and lesson menus, and launching the selected lesson.

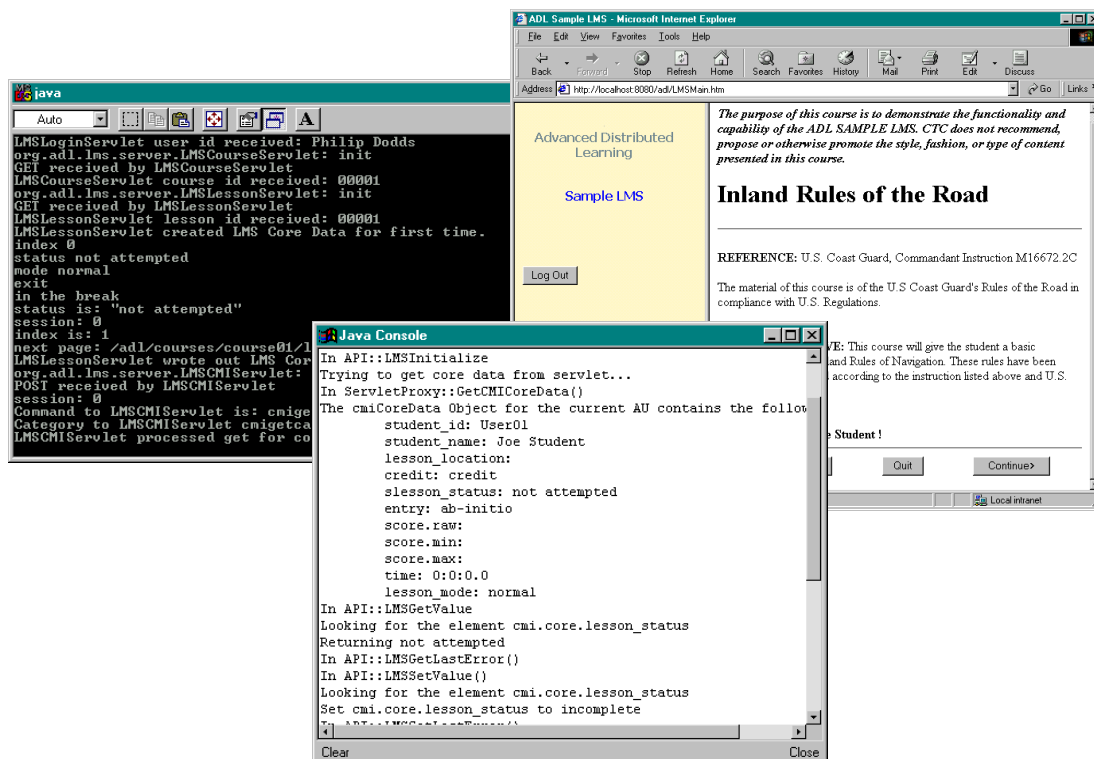


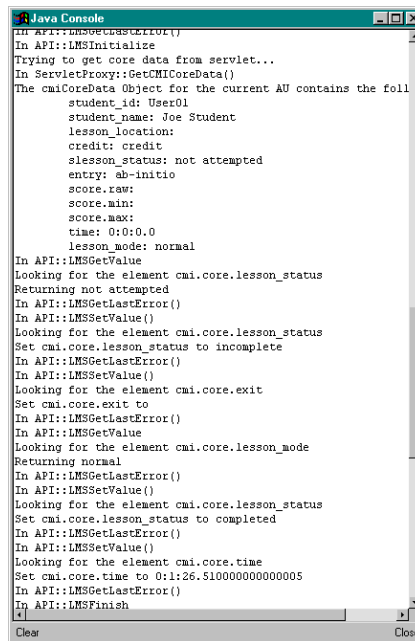
Figure 8-2. LMS Server Functionality

There are five servlets, as described below:

1. **LMSCMIServlet.java.** The LMSCMIServlet provides a mechanism for communication between the LMS and the assignable unit—in this case the lesson. This particular implementation is specifically for the CMI data model described in the SCORM. Other modules could be created to handle other data models as they come into existence. This servlet sends and receives serialized data model objects via HTTP to and from the LMS API client (see Section 8.1.2). This sample supports only the core data elements defined in Appendix B of the AICC CMI *Guidelines for Interoperability*, Version 3.0. Data persistence is handled by Java's built-in serialization mechanism using a file on the local file system. No database systems are being used at this time.
2. **LMSLoginServlet.java.** The LMSLoginServlet provides the ability to validate the student and serve the student's course menu. The course menu is an HTML page dynamically built by the servlet. This servlet is expandable to allow for future dynamic generation of course menus based on student course registration. It is invoked from the LMS client when the user presses "submit" on the LMS Login form.
3. **LMSCourseServlet.java.** The LMSCourseServlet provides the ability to serve the course lesson menu. Similar to the student course menu, the course lesson menu is an HTML page that is dynamically built by the servlet. This servlet is also expandable to allow for dynamic lesson menu generation based on a CSF XML document in a future release. This servlet is called when the student clicks on the course.
4. **LMSLessonServlet.java.** The LMSLessonServlet provides the ability to launch the assignable units for the lesson. The servlet also handles the sequencing of the assignable units. In this version, the sequencing is hard coded and not based on a CSF. This servlet serves the appropriate assignable unit page of the selected lesson to the browser within the LMS client framework. It is called when the user navigates through the lesson.
5. **LMSResetDBServlet.java.** The LMSResetDBServlet provides the ability to delete the student-persistent data that the LMS maintains. This action will reset the student back to the "not have entered the lesson yet" state. This is provided as a convenience to the user so that the student data does not have to be deleted manually at the server. This capability is necessary if you want to use the sample lesson multiple times. Because the LMS tracks lesson completion and does not allow students to take the lesson multiple times, it is necessary to delete the student data and reset the state of the sample lesson.

8.1.2 LMS Client

The LMS client side consists of an LMS sample user interface implemented in HTML and JavaScript and the LMS API implemented as a Java applet (see Figure 8-3). The applet is downloaded to the client when the user accesses the LMS Main start page



```

In API::LMSGetLastError()
In API::LMSInitialize
Trying to get core data from servlet...
In ServletProxy::GetCMICoreData()
The cmicoreData Object for the current AU contains the foll
  student_id: User01
  student_name: Joe Student
  lesson_location:
  credit: credit
  slesson_status: not attempted
  entry: ab-initio
  score.raw:
  score.min:
  score.max:
  time: 0:0:0.0
  lesson_mode: normal
In API::LMSGetValue
Looking for the element cmicore.lesson_status
Returning not attempted
In API::LMSGetLastError()
In API::LMSSetValue()
Looking for the element cmicore.lesson_status
Set cmicore.lesson_status to incomplete
In API::LMSGetLastError()
In API::LMSSetValue()
Looking for the element cmicore.exit
Set cmicore.exit to
In API::LMSGetLastError()
In API::LMSGetValue
Looking for the element cmicore.lesson_mode
Returning normal
In API::LMSGetLastError()
In API::LMSSetValue()
Looking for the element cmicore.lesson_status
Set cmicore.lesson_status to completed
In API::LMSGetLastError()
In API::LMSSetValue()
Looking for the element cmicore.time
Set cmicore.time to 0:1:26.5100000000000005
In API::LMSGetLastError()
In API::LMSFinish

```

Figure 8-3. LMS Client

through a Web browser. The API applet provides the communication to the LMS server for data model element persistence. The assignable units make calls to the API functions from JavaScript. (See the description of the sample course below for a description of how the LMS API function calls are made from the assignable units). The assignable units do not need to know about any of the LMS implementation details.

The following java source files make up the client side LMS API implementation:

- **API.java.** This file contains the API class that is extended from the Java Abstract Window Toolkit (AWT) applet class and implements the LMS API functions (i.e., LMSInitialize, LMSFinish, and so forth).
- **LMSErrorManager.java.** This file contains the LMSErrorManager class, which encapsulates the error-handling capabilities specified for the LMS API. It maintains the most recent error code and the mapping of error codes to the error text and diagnostic information.
- **ServletProxy.java.** This file contains the ServletProxy class, which encapsulates the communication between the LMS client API applet and the LMS server.
- **ServletWriter.java.** This file contains the ServletWriter class, which provides the low-level input and output serialized object streaming capability used by ServletProxy to communicate with the servlet(s) via HTTP. This class was downloaded from <http://www.javasoft.com>.

The HTML/JavaScript sample LMS User Interface is made up of the following HTML files:

- **LMSMain.htm.** This file is the main LMS client page in which all student access must begin. This page contains a frameset, which, in turn, contains two frames: an LMS navigation frame (left-side), which loads the LMSFrame.htm and a

content frame (right-side). The content frame initially contains the start page (see LMSStart.htm). As the user logs in and selects a course, lesson, and so forth, the right-side frame houses this content. Currently, no internal checks exist to prevent the user from typing in the URL of one of the other LMS HTML pages before starting at LMSMain.htm. If the user attempts to access one of the other pages, undetermined LMS behavior will result.

- **LMSFrame.htm.** This page contains the LMS API applet. The LMS API applet has no visual display elements and is therefore invisible to the user. The assignable units communicate with the LMS via this API. This page also contains a login and logout button.
- **LMSStart.htm.** This page contains a brief textual description of the ADL Sample LMS. It is initially displayed in the right frame of the LMSMain.htm page when the user first accesses the sample LMS.
- **LMSLogin.htm.** This page contains an HTML form that prompts the user for a user name and password. Currently, no checks are performed on the entries on this form, and the user may proceed simply by clicking the “Submit” button. Submitting this form invokes the LMSLoginServlet.
- **LMSResetConfirm.htm.** This page displays an “Are You Sure?” message to the user when they choose Reset Student from the main LMS client page.
- **LMSResetComplete.htm.** This page displays a “Reset Complete” message to the user when the student data have been deleted.

8.1.3 AICC CMI Data Model

As mentioned previously, the AICC CMI data model (see Figure 8-4) is the data model used to communicate between the LMS and the assignable units and vice versa. At this time, only the core data elements are implemented in this sample.

Several classes are included to represent the core data elements of the CMI model. Both the servlets and the applet use these classes. Eventually, each data type described in the AICC CMI guidelines will be implemented as its own class. At this time, only the CMITime data type is implemented as a java class.

The source for these classes is as follows:

- **CMIScoreData.java.** The CMIScoreData contains the implementation for cmc.core.score data element.
- **CMITime.java.** The CMITime contains the implementation of the CMITime data type.
- **CMICoreData.java.** The CMICoreData contains the implementation of the cmc.core data model elements needed for the AU to LMS and LMS to AU communication.

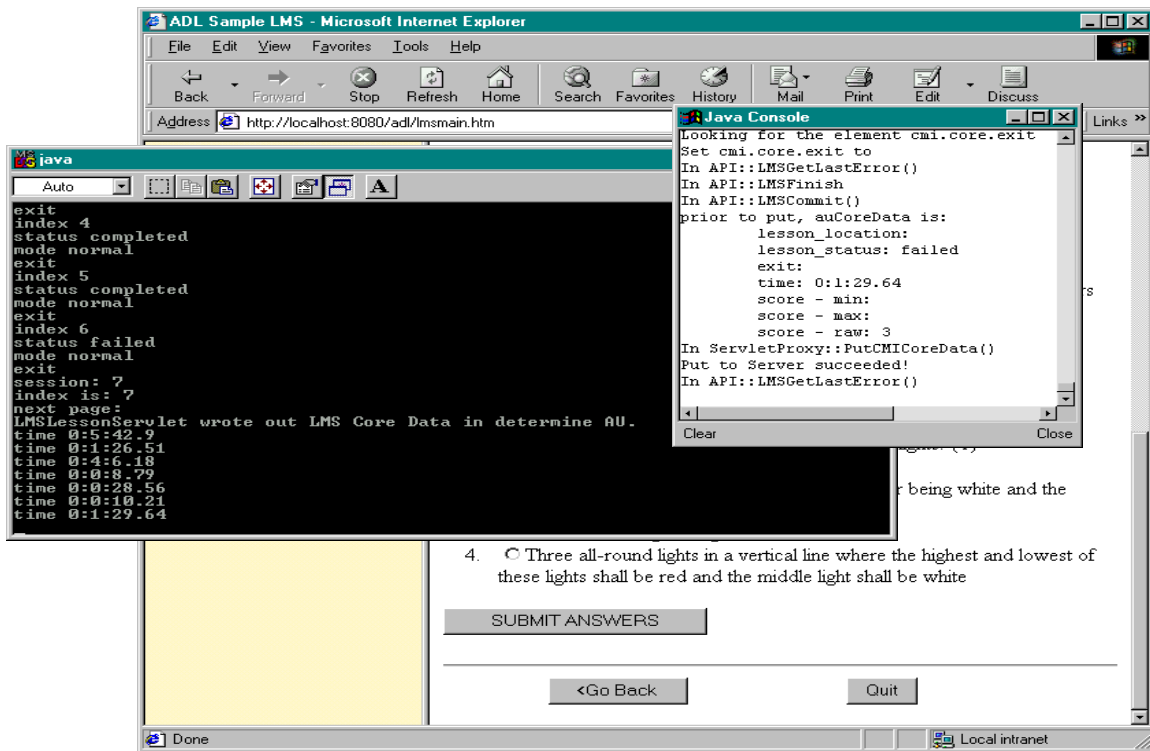


Figure 8-4. AICC CMI Data Model

8.1.4 Sample Course

The course provided with this sample LMS contains one lesson. The lesson is a simple “page turner” written in HTML and JavaScript and consists of seven assignable units. The lesson is not meant to serve as “a great example of a how to develop learning content,” but rather as an example of how to communicate data between the LMS and the assignable units using the API. The assignable unit HTML pages used in this lesson make use of two JavaScript “include” files called APIWrapper.js and AUFunctions.js.

The APIWrapper.js file contains a set of “API wrapper” functions that encapsulate the functionality an assignable unit might use to communicate with the LMS via the API. This file does not implement the API functions. It encapsulates the logic needed to:

- Find the API in the LMS client framework
- Call the desired API function
- Handle errors that might be generated by the call to the API function.

The assignable units are written using the wrapper, thus providing a level of abstraction above the actual implementation of the API and hiding the functionality of the communication between the client and the server.

The lesson assignable units consist of the following source files:

- au01.htm, au02.htm, au03.htm, au04.htm
- au05.htm, au06.htm, au07.htm.

The AUFunctions.js file contains a set of navigation functions common to all of the assignable units. This file contains JavaScript functions used by all of the lesson HTML pages. It is included at runtime in each of the above HTML pages. (Something similar to this might be provided by authoring tools to encapsulate general functionality commonly built into all the courses authored by that particular tool.)

8.1.5 Mapping Example Code to the SCORM

Figure 8-5 shows the mapping of example code to the SCORM.

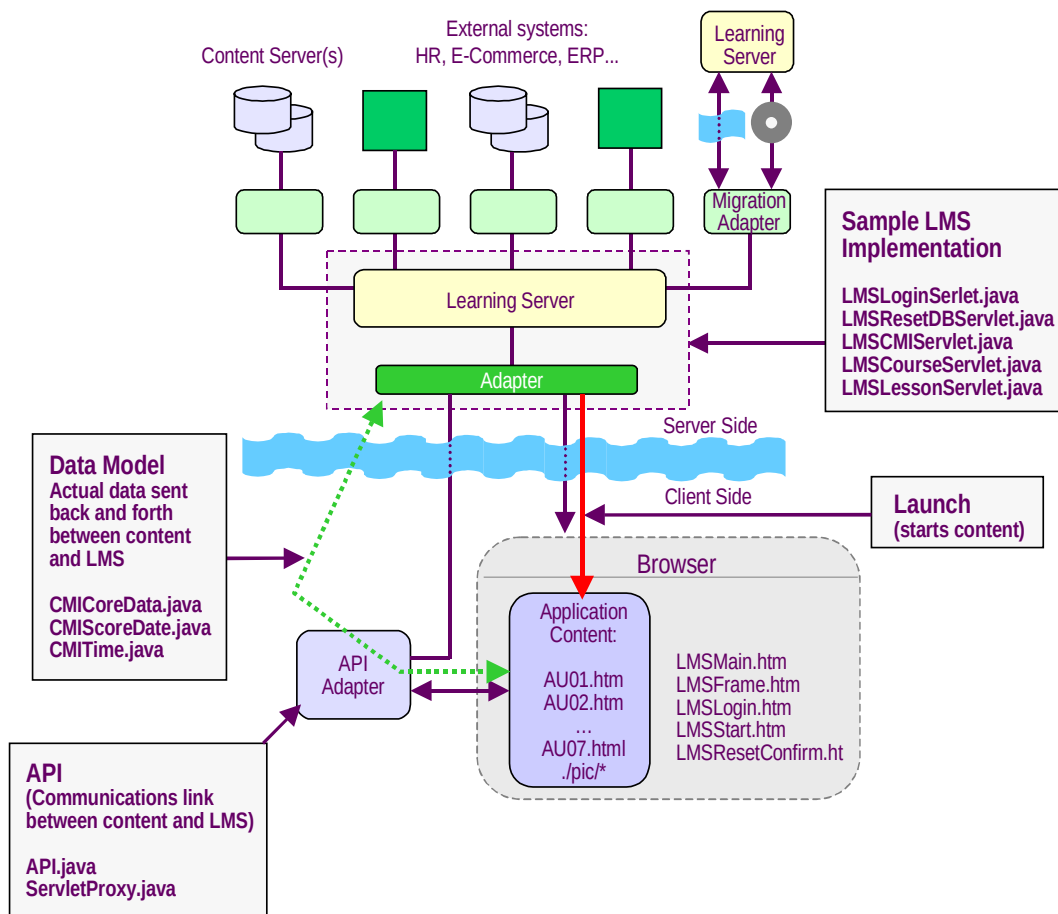


Figure 8-5. Mapping Example Code to the SCORM

8.1.6 Structure of Sample LMS Application

Figure 8-6 illustrates the relationship of the SCORM Sample LMS files.

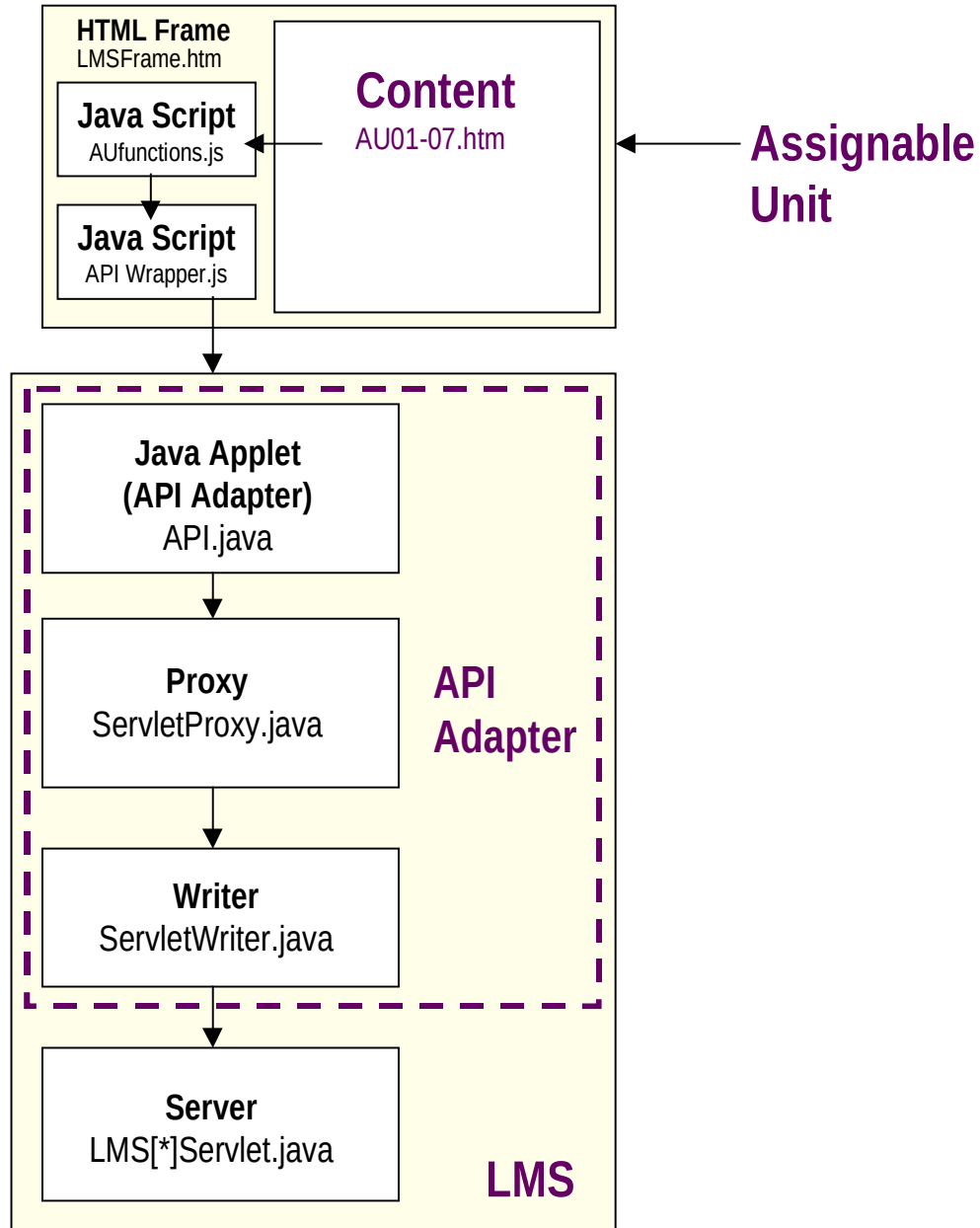


Figure 8-6. The Relationship of the SCORM Sample LMS Files

8.1.7 Flow of Sample LMS Application

Figure 8-7 illustrates the flow of the sample SCORM LMS code:

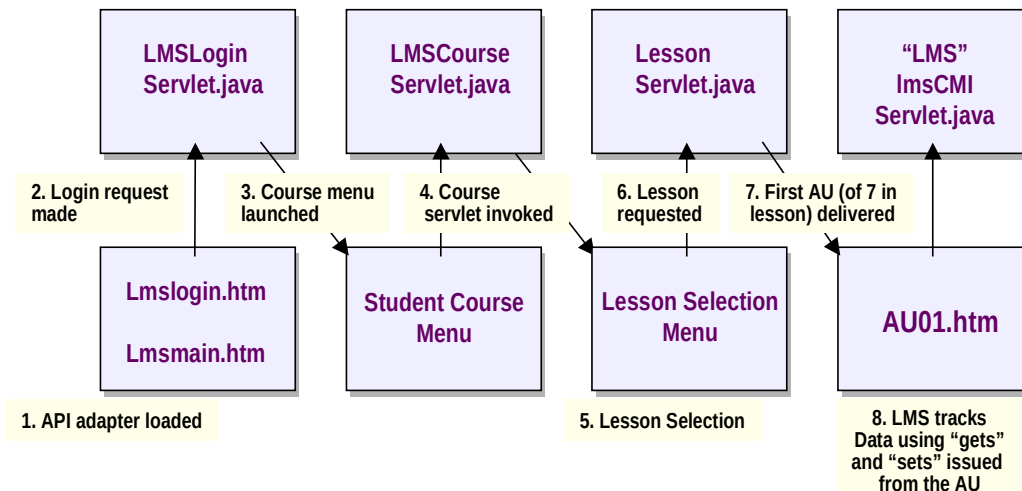


Figure 8-7. The Flow of the Sample SCORM LMS Code

8.1.8 API Wrapper JavaScript Code Fragment

The following code fragment is from APIWrapper.js in the SCORM sample LMS package. This code is used on the client side to locate the API and to call the API in the applet API.java.

```

function LMSInitialize()
{
    var api = GetAPI();
    if (api == null)
    {
        alert("Unable to locate the LMS's API Implementation.\nLMSInitialize was not successful.");
        return false;
    }
    // call the LMSInitialize function that should be implemented by the API
    var emptyString = new String("");
    var initResult = api.LMSInitialize(emptyString);
    if (initResult.toString() != "1")
    {
        // LMSInitialize did not complete successfully.
        var err = ErrorHandler();
    }
    return initResult;
}

function LMSFinish()
{
    var api = GetAPI();
    if (api == null)
    {
        alert("Unable to locate the LMS's API Implementation.\nLMSFinish was not successful.");
    }
    else
    {
        api.LMSFinish();
        var err = ErrorHandler();
    }
    return;
}

function LMSGetValue(name)
{ ... }

function LMSSetValue(name, value)

```

```

{ ... }
function LMSCcommit()
{ ... }
function LMSGetLastError()
{ ... }
function LMSGetErrorString(errorCode)
{ ... }
function LMSGetDiagnostic(errorCode)
{ ... }
function LMSIsInitialized()
{ ... }
function FindAPI()
{
    if(_Debug){alert("in FindAPI()");}
    //Is it in the current window?
    if (window.document.API != null)
    {
        if(_Debug){alert("found api in this window");}
        return window.document.API;
    }
    // Is it in the window's opener?
    if (window.opener != null)
    {
        if (_Debug) {alert("window opener is NOT NULL");}

        if (window.opener.API != null)
        {
            if(_Debug){alert("found api in this window's opener.");}

            return window.opener.API;
        }
        else
        {
            // look in the openers window's frames...
            if (window.opener.parent != null)
            {
                if (_Debug) { alert("looking in window opener parent's frames");}
                for (i=0; i<window.opener.parent.frames.length; i++)
                {
                    if (window.opener.parent.frames.item(i).API != null)
                    {
                        if(_Debug){alert("found api in this window's opener's parent's frames.");}
                        return window.opener.parent.frames.item(i).API;
                    }
                }
            }
        }
    }
}
// Is it in the current window's parent?
if (window.parent != null)
{
    if (window.parent.document.API != null)
    {
        if(_Debug){alert("found api in this window's parent.");}

        return window.parent.API;
    }
    else
    {
        // look in the parent window's frames...
        // this is probably the most likely place for it to be...
        for (i=0; i<window.parent.frames.length; i++)
        {
            if (window.parent.frames.item(i).API != null)
            {
                if(_Debug){alert("found api in this window's parents frames.");}
                return window.parent.frames.item(i).API;
            }
        }
    }
}

```

```

    }
    // The API was not found
    if (_Debug)
    {
        alert("couldn't find an API implementation");
    }
    return null;
}

function GetAPI()
{
    return apiHandle;
}

```

8.1.9 Course Structure Format XML Fragment

```

<?xml version="1.0"?>
<!--File Name: MaritimeNavigation.xml-->
<!DOCTYPE course SYSTEM "scocsf(1.0).dtd">
<!--This is an example of a course-->
<course>
    <globalProperties>
        <externalMetadata>
            <source>IEEE 3.0</source>
            <model>IMSBP</model>
            <location>\metadata\MaritiemNavigation.xml</location>
        </externalMetadata>
    </globalProperties>
    <block id="B1">
        <objectiveRef targetIDs="01"></objectiveRef>
        <identification>
            <title>Maritime Navigation</title>
            <labels>
                <curricular>UNIT</curricular>
            </labels>
        </identification>
        <extensions>
            <source>AICC CMI AGR-006</source>
            <model>cmi</model>
            <property>
                <name>difficulty</name>
                <value>easy</value>
            </property>
        </extensions>
    </block>
    <block id="B2">
        <objectiveRef targetIDs="02"></objectiveRef>
        <identification>
            <title>Inland Rules of the Road</title>
            <labels>
                <curricular>MODULE</curricular>
            </labels>
        </identification>
        <extensions>
            <source>AICC CMI AGR-006</source>
            <model>cmi</model>
            <property>
                <name>difficulty</name>
                <value>easy</value>
            </property>
        </extensions>
    </block>
    <au id="A1">
        <objectiveRef targetIDs="03"></objectiveRef>
        <identification>
            <title>References</title>
        </identification>
        <launch>
            <location>http://&#60;host&#62;/Courses/Course01/Lesson01/au01.html</location>
        </launch>
    </au>

```

```

<block id="B3">
  <objectiveRef targetIDs="04"></objectiveRef>
  <identification>
    <title>Steering &#38; Sailing Rules</title>
    <labels>
      <curricular>MODULE</curricular>
    </labels>
  </identification>
  <au id="A2">
    <externalMetadata>
      <source>IMSBP</source>
      <model>IEEE 3.0</model>
      <location>\metadata\au02MetaData.xml</location>
    </externalMetadata>
    <objectiveRef targetIDs="05"></objectiveRef>
    <identification>
      <title>Conduct of Vessels in any Condition of
Visibility</title>
      <labels>
        <curricular>LEARNING STEP</curricular>
      </labels>
    </identification>
    <launch>

<location>http://&#60;host&#62;/Courses/Course01/Lesson01/au02.html</location>
    </launch>
  </au>

. . .

```

8.1.10 Course Metadata XML Example

```

<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE RECORD SYSTEM "http://www.imsproject.org/xml/IMS-MD01.dtd" -->
<!-- Uses Master DTD at IMS site. -->
<!DOCTYPE RECORD SYSTEM "IMS-MD01.dtd" >
<!-- If used with a local DTD, use this DOCTYPE declaration instead of Master above. -->

<RECORD xmlns="http://www.imsproject.org/metadata/">
  <METAMETADATA>
    <METADATAScheme>
      LOM-3.8
    </METADATAScheme>
  </METAMETADATA>

  <GENERAL>
    <TITLE>
      <LANGSTRING>Inland Rules of the Road</LANGSTRING>
    </TITLE>
    <CATALOGENTRY>
      <CATALOGUE>ADL Course Catalogue ID</CATALOGUE>
      <ENTRY>
        <LANGSTRING>1000</LANGSTRING>
      </ENTRY>
    </CATALOGENTRY>
    <CATALOGENTRY>
      <CATALOGUE>Course Configuration Management System Path</CATALOGUE>
      <ENTRY>
        <LANGSTRING>VOB:/vob/adli/source/SampleLMS/Courses/Course01/</LANGSTRING>
      </ENTRY>
    </CATALOGENTRY>
    <CATALOGENTRY>
      <CATALOGUE>U.S. Coast Guard, Commandant Instruction</CATALOGUE>
      <ENTRY>
        <LANGSTRING>M6672.2C</LANGSTRING>
      </ENTRY>
    </CATALOGENTRY>
    <LANGUAGE>en-US</LANGUAGE>
    <DESCRIPTION>

```

```

    <LANGSTRING>Basic instruction on U.S. Coast Guard and U.S. Regulation of Inland Vessel Rules of
Navigation</LANGSTRING>
</DESCRIPTION>
<KEYWORDS>
    <LANGSTRING>Vessel</LANGSTRING>
    <LANGSTRING>Inland Navigation</LANGSTRING>
    <LANGSTRING>Coast Guard</LANGSTRING>
</KEYWORDS>
</GENERAL>

<LIFECYCLE>
<VERSION>
    <LANGSTRING>1.0</LANGSTRING>
</VERSION>
<STATUS>
    <LANGSTRING>Final</LANGSTRING>
</STATUS>
<CONTRIBUTE>
    <ROLE>
        <LANGSTRING>Author</LANGSTRING>
    </ROLE>
    <CENTITY>
        <VCARD>
BEGIN:VCARD
ORG:Concurrent Technologies Corporation;ADLI Project
END:VCARD
        </VCARD>
    </CENTITY>
    <DATE>
        <DATETIME>
            2000-01-28
        </DATETIME>
    </DATE>
</CONTRIBUTE>
</LIFECYCLE>

<TECHNICAL>
<FORMAT>
    <LANGSTRING>text/html</LANGSTRING>
</FORMAT>
<LOCATION type="URI">
    /Courses/Course01/
</LOCATION>
<REQUIREMENTS>
    <TYPE>
        <LANGSTRING>Browser</LANGSTRING>
    </TYPE>
    <NAME>
        <LANGSTRING>Microsoft Internet Explorer</LANGSTRING>
    </NAME>
    <MINIMUMVERSION>5.0</MINIMUMVERSION>
</REQUIREMENTS>
</TECHNICAL>

<EDUCATIONAL>
<LEARNINGRESOURCETYPE>
    <LANGSTRING>Narrative Text</LANGSTRING>
</LEARNINGRESOURCETYPE>
</EDUCATIONAL>

<RIGHTS>
<COST>
    <!-- yes or no -->
    <LANGSTRING>no</LANGSTRING>
</COST>
<COPYRIGHTOROTHERRESTRICTIONS>
    <!-- yes or no -->
    <LANGSTRING>no</LANGSTRING>
</COPYRIGHTOROTHERRESTRICTIONS>
<DESCRIPTION>

```

```

    <LANGSTRING>Provided as-is; only to be used as an example.</LANGSTRING>
  </DESCRIPTION>
</RIGHTS>

<CLASSIFICATION>
  <PURPOSE>
    <LANGSTRING>Educational Objective</LANGSTRING>
  </PURPOSE>
  <DESCRIPTION>
    <LANGSTRING>This course will give the student a basic understanding of the Inland Rules of
Navigation.</LANGSTRING>
  </DESCRIPTION>
  <KEYWORDS>
    <LANGSTRING>Inland Navigation</LANGSTRING>
  </KEYWORDS>
</CLASSIFICATION>
</RECORD>

```

8.1.11 Assignable Unit (Content) Metadata XML Example

```

<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE RECORD SYSTEM "http://www.imsproject.org/xml/IMS-MD01.dtd" -->
<!-- Uses Master DTD at IMS site. -->
<!DOCTYPE RECORD SYSTEM "IMS-MD01.dtd" >
<!-- If used with a local DTD, use this DOCTYPE declaration instead of Master above. -->

<RECORD xmlns="http://www.imsproject.org/metadata/">
  <METAMETADATA>
    <METADATAScheme>
      LOM-3.8
    </METADATAScheme>
    <LANGUAGE>en-US</LANGUAGE>
  </METAMETADATA>

  <GENERAL>
    <TITLE>
      <LANGSTRING>Conduct of Vessels in any Condition of Visibility</LANGSTRING>
    </TITLE>
    <CATALOGENTRY>
      <CATALOGUE>ADL Course Catalogue ID</CATALOGUE>
      <ENTRY>
        <LANGSTRING>1000-02</LANGSTRING>
      </ENTRY>
    </CATALOGENTRY>
    <CATALOGENTRY>
      <CATALOGUE>Course Configuration Management System Path</CATALOGUE>
      <ENTRY>
        <LANGSTRING>VOB:/vob/adli/source/SampleLMS/Courses/Course01/Lesson01/AU01</LANGSTRING>
      </ENTRY>
    </CATALOGENTRY>
    <LANGUAGE>en-US</LANGUAGE>
    <DESCRIPTION>
      <LANGSTRING>Discusses general rules of operation for vessels on inland waters.
        Topics discussed include: Look-out, Safe Speed, Collision, Channels, Traffic Separation.</LANGSTRING>
    </DESCRIPTION>
    <KEYWORDS>
      <LANGSTRING>Vessel</LANGSTRING>
      <LANGSTRING>Any Visibility</LANGSTRING>
      <LANGSTRING>Look-out</LANGSTRING>
      <LANGSTRING>Safe Speed</LANGSTRING>
      <LANGSTRING>Collision</LANGSTRING>
      <LANGSTRING>Channels</LANGSTRING>
      <LANGSTRING>Traffic Separation</LANGSTRING>
    </KEYWORDS>
    <AGGREGATIONLEVEL>1</AGGREGATIONLEVEL>
  </GENERAL>

  <LIFECYCLE>
    <VERSION>

```

```

    <LANGSTRING>1.0</LANGSTRING>
  </VERSION>
  <STATUS>
    <LANGSTRING>Final</LANGSTRING>
  </STATUS>
  <CONTRIBUTE>
    <ROLE>
      <LANGSTRING>Author</LANGSTRING>
    </ROLE>
    <CENTITY>
      <!-- Stub of a vCard entry with Formatted Name (FN) property -->
    </VCARD>
  </VCARD>
  BEGIN:VCARD
  ORG:Concurrent Technologies Corporation; ADLI Project
  END:VCARD
  </VCARD>
  </CENTITY>
  <DATE>
    <DATETIME>
      2000-01-28
    </DATETIME>
  </DATE>
  </CONTRIBUTE>
</LIFECYCLE>

<TECHNICAL>
  <FORMAT>
    <LANGSTRING>text/html</LANGSTRING>
  </FORMAT>
  <SIZE>
    14368
  </SIZE>
  <LOCATION type="URI">
    au02.htm
  </LOCATION>
  <!-- type of URI or TEXT -->
  <REQUIREMENTS>
    <TYPE>
      <LANGSTRING>Browser</LANGSTRING>
    </TYPE>
    <NAME>
      <LANGSTRING>Microsoft Internet Explorer</LANGSTRING>
    </NAME>
    <MINIMUMVERSION>5.0</MINIMUMVERSION>
  </REQUIREMENTS>
</TECHNICAL>

<EDUCATIONAL>
  <LEARNINGRESOURCE TYPE>
    <LANGSTRING>Narrative Text</LANGSTRING>
  </LEARNINGRESOURCE TYPE>
  <TYPICALLEARNINGTIME>
    <DATETIME>
      0000-00-00T00:05:00
    </DATETIME>
  </TYPICALLEARNINGTIME>
</EDUCATIONAL>

<RIGHTS>
  <COST>
    <!-- yes or no -->
    <LANGSTRING>no</LANGSTRING>
  </COST>
  <COPYRIGHTOROTHERRESTRICTIONS>
    <!-- yes or no -->
    <LANGSTRING>no</LANGSTRING>
  </COPYRIGHTOROTHERRESTRICTIONS>
</RIGHTS>

<CLASSIFICATION>

```

```

<PURPOSE>
  <LANGSTRING>Educational Objective</LANGSTRING>
</PURPOSE>
<DESCRIPTION>
  <LANGSTRING>Vessel Conduct</LANGSTRING>
</DESCRIPTION>
<KEYWORDS>
  <LANGSTRING>Vessel Conduct</LANGSTRING>
</KEYWORDS>
</CLASSIFICATION>
</RECORD>

```

8.1.12 Raw Media Metadata XML Example

```

<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE RECORD SYSTEM "http://www.imspjproject.org/xml/IMS-MD01.dtd" -->
<!-- Uses Master DTD at IMS site. -->
<!DOCTYPE RECORD SYSTEM "IMS-MD01.dtd" >
<!-- If used with a local DTD, use this DOCTYPE declaration instead of Master above. -->

<RECORD xmlns="http://www.imspjproject.org/metadata/">
  <METAMETADATA>
    <METADATAScheme>
      LOM-3.8
    </METADATAScheme>
    <LANGUAGE>en-US</LANGUAGE>
  </METAMETADATA>

  <GENERAL>
    <TITLE>
      <LANGSTRING>Vessel Aground Lighting</LANGSTRING>
    </TITLE>
    <DESCRIPTION>
      <LANGSTRING>
        Picture showing lighting requirements for inland vessel less than 50 meters in length in an aground condition.
      </LANGSTRING>
    </DESCRIPTION>
    <KEYWORDS>
      <LANGSTRING>Vessel</LANGSTRING>
      <LANGSTRING>Lighting</LANGSTRING>
      <LANGSTRING>Aground</LANGSTRING>
    </KEYWORDS>
  </GENERAL>

  <LIFECYCLE>
    <VERSION>
      <LANGSTRING>1.0</LANGSTRING>
    </VERSION>
    <STATUS>
      <LANGSTRING>Final</LANGSTRING>
    </STATUS>
    <CONTRIBUTE>
      <ROLE>
        <LANGSTRING>Author</LANGSTRING>
      </ROLE>
      <CENTITY>
        <!-- Stub of a vCard entry with Formatted Name (FN) property -->
        <VCARD>
          BEGIN:VCARD
          ORG:Concurrent Technologies Corporation; ADLI Project
          END:VCARD
        </VCARD>
      </CENTITY>
      <DATE>
        <DATETIME>
          2000-01-28
        </DATETIME>
      </DATE>
    </CONTRIBUTE>
  </LIFECYCLE>

```

```

</LIFECYCLE>

<TECHNICAL>
  <FORMAT>
    <LANGSTRING>image/jpeg</LANGSTRING>
  </FORMAT>
  <SIZE>
    10612
  </SIZE>
  <LOCATION type="URI">
    aground.jpg
  </LOCATION>
  <!--type of URI or TEXT-->
</TECHNICAL>

<EDUCATIONAL>
  <LEARNINGRESOURCE TYPE>
    <LANGSTRING>Figure</LANGSTRING>
  </LEARNINGRESOURCE TYPE>
</EDUCATIONAL>

<RIGHTS>
  <COST>
    <!-- yes or no -->
    <LANGSTRING>no</LANGSTRING>
  </COST>
  <COPYRIGHTOROTHERRESTRICTIONS>
    <!-- yes or no -->
    <LANGSTRING>no</LANGSTRING>
  </COPYRIGHTOROTHERRESTRICTIONS>
</RIGHTS>

</RECORD>

```

8.2 CSF Browser/Editor

The CSF Browser/Editor (see Figure 8-8) is a java application that can create, edit, display, and write CSF XML files. It runs on top of the IBM XML For Java Parser using the Simple API for XML driver available from IBM's AlphaWorks at <http://www.alphaworks.ibm.com/formula/XML>.

Note: As of the release of this document, the CSFBrowser is "alpha" code. Soon after the release of this document, the application and source code will be available on the ADL web site (<http://www.adlnet.org>).

8.2.1 Overview of CSF Browser/Editor

This tool was designed principally to exercise all the aspects of the CSF specification. It parses an XML CSF file and displays the file's "original" XML structure. Then, it maps each element into a Java "data model" using a separate course tree. The course structure tree can be edited and written back out as XML. Thus, this utility tests the CSF through the input, display, and output translations of CSF data.

The CSF Browser reads XML CSF files and validates them against the DTD specified in the XML record. The application has four view panes:

1. **XML Representation of Course Structure pane.** This pane displays in a tree format each of the XML elements exactly as they were parsed when a CSF XML

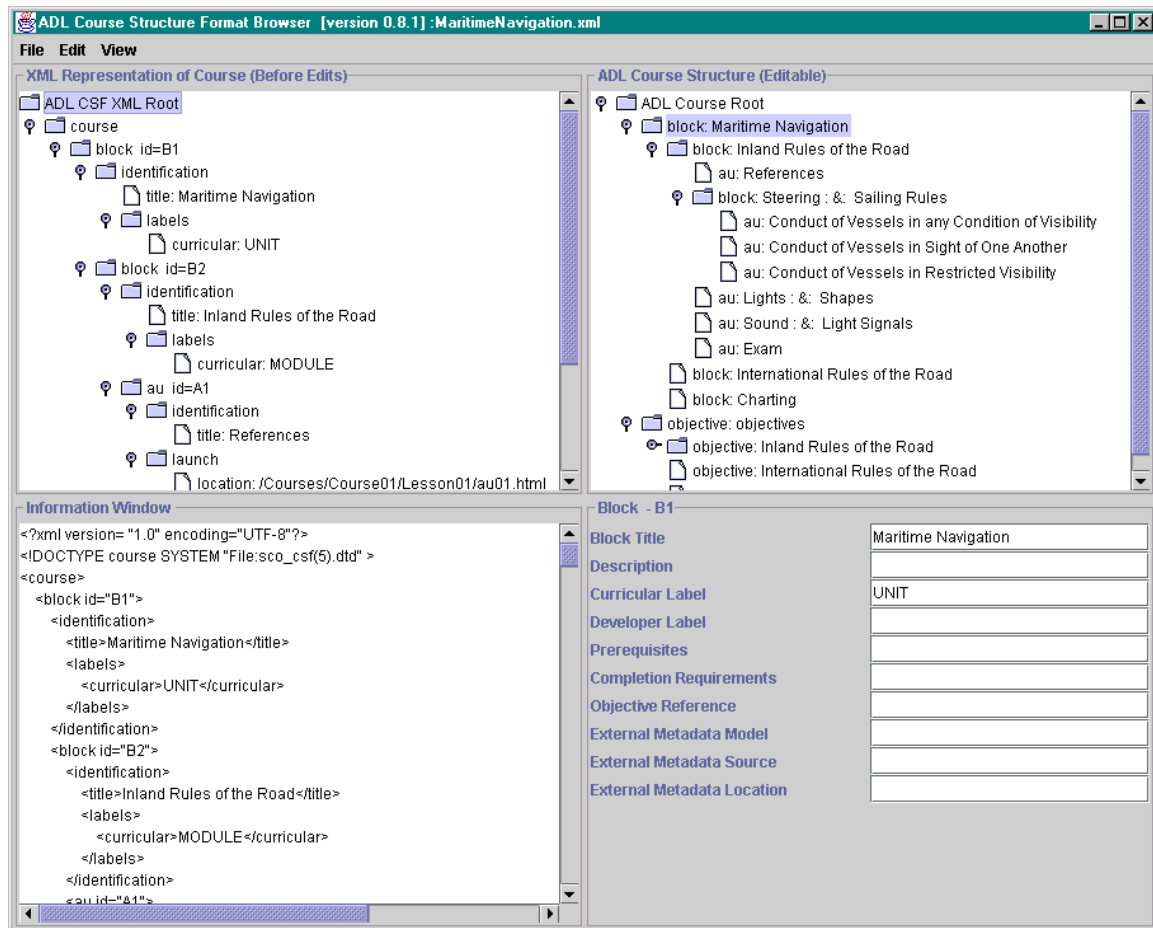


Figure 8-8. Sample ADL CSF Format Browser

file is opened. Parsing errors during loading are displayed in the Information pane. This tool makes viewing the contents of a CSF XML file easier than reading the raw code. The XML Representation pane cannot be edited and only displays initially loaded files.

2. **ADL Course Structure pane.** This pane contains another tree similar to the XML Representation pane, except that it only shows *course*, *globalProperties*, *block*, *au*, and *objectives* in the tree. This makes viewing the course structure easier.

The Course Structure tree can be edited. Clicking on *course*, *globalProperties*, *block*, *au*, or *objectives* brings up an editor for each of the elements of these groups. Nodes in the tree can be added, deleted, or moved.

3. **Information pane.** This pane displays guidance information about items selected and displays relevant text messages or outputs when selected.
4. **Editor pane.** This pane displays the elements of *course*, *globalProperties*, *block*, *au*, or *objectives* tree nodes when they are selected.

After editing, CSF trees may be saved as CSF XML files or displayed in the Information Pane.

9. Acronym List

ADL	Advanced Distributed Learning [Initiative]
ADL-CoLab	ADL Collaboration Laboratory
AICC	Aviation Industry CBT Committee
API	Application Programming Interface
AU, au	Assignable Unit
AWT	Abstract Window Toolkit
CBI	Computer-Based Instruction
CMI	Computer-Managed Instruction
CSF	Course Structure Format
DC	Dublin Core
DoD	Department of Defense
DTD	Document Type Definition
HTML	Hypertext Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IMS	Instructional Management Systems [Project]
ISO	International Standards Organization
JPEG	Joint Photographic Experts Group
LMS	Learning Management System
LOM	Learning Object Metadata
LTSC	Learning Technology Standards Committee
OSTP	Office of Science and Technology Policy
SCO	Sharable Courseware Object
SCORM	Sharable Courseware Object Reference Model
SEL	Standard Extension Library
URI	Universal Resource Identifier
URL	Universal Resource Locator
XML	Extensible Markup Language

Appendix A – Supporting Documents

A.1 SCORM Course Structure Format DTD

```

<!--*** scormcsf(1.0).dtd
*** course: Root level of Course Structure representation-->
<!ELEMENT course (globalProperties?,block,objectives?) >

<!--*** globalProperties: Properties of the course as whole-->
<!ELEMENT globalProperties (externalMetadata+,curricularTaxonomy?,extensions*) >

<!--*** block: A grouping of related structural elements.
Blocks contain assignable units or other blocks.
Blocks always contain other course elements.
This holds an unique (to this course) ID identifier
for a particular block.
ID's are generated by the application (e.g., an LMS)
that creates a CSF XML file
(other elements may refer to this unique ID)-->

<!ELEMENT block ((externalMetadata*,objectiveRef*,identification,prerequisites?,
completionReq?,extensions*,(au* | block*)+) | blockAlias) >

<!ATTLIST block
    id ID #REQUIRED >

<!--*** objectives: Root level of objectives tree;
Statements of skills, knowledge, and attitudes
to be acquired by the student.-->
<!ELEMENT objectives (objective+) >

<!--*** externalMetadata: The value of this element refers
or points to the location of the metadata
describing this course.-->

<!ELEMENT externalMetadata (source,model,location) >

<!--*** curricularTaxonomy: Organizational methodology
used to construct the course-->
<!ELEMENT curricularTaxonomy (source?,model,location?) >

<!--*** extensions: defines extensions to course element
definitions and their source-->
<!ELEMENT extensions (source?,model,location?,property+) >

<!--*** objectiveRef: Reference to a particular objective
in the objective hierarchy-->

<!ELEMENT objectiveRef EMPTY >

<!ATTLIST objectiveRef
    targetIDs IDREFS #IMPLIED >

<!--*** identification: Identifies course context-
specific information-->
<!ELEMENT identification (title,description?,labels?) >

<!--*** prerequisites: Expression indicating what a student
must accomplish before beginning this course element.
Course elements that a student must complete before beginning
a block or assignable unit. It uses a "script" that defines
the logical rules to be applied. The script type must be defined.
e.g., <prerequisites type="aicc_script"> <![CDATA[B1&B2&A1]]>
</prerequisites> -->
<!ELEMENT prerequisites (#PCDATA) >

```

```

<!--ATTLIST prerequisites
      type CDATA      #IMPLIED >

<!--*** completionReq: Course elements that a student must complete
      before considering a given structure element complete. It uses
      a "script" that defines the logical rules to be applied. The
      script type must be defined. e.g.,
      <completion type="aicc_script"> <![CDATA[B1&B2&A1]]> </completion>-->
<!--ELEMENT completionReq (#PCDATA) >

<!--ATTLIST completionReq
      type CDATA      #IMPLIED >

<!--*** au: An AU is the smallest element of instruction or testing to
      which a student may be routed by a LMS. It refers to "content"
      launched by the LMS system.
      This holds a unique (to this course) ID identifier for a particular
      au ID's are generated by the application (e.g., an LMS) that creates
      a CSF XML file (other elements may refer to this unique ID)-->
<!--ELEMENT au ((externalMetadata*,objectiveRef*,identification,prerequisites?,
      completionReq?,timeLimit?,launch?,masteryScore?,extensions*) |
      auAlias) >

<!--ATTLIST au
      id ID      #REQUIRED >

<!--*** blockAlias: Reference to a previously defined block
      (permits one block to be used more than once within a course)-->
<!--ELEMENT blockAlias EMPTY >

<!--ATTLIST blockAlias
      targetID ID      #IMPLIED >

<!--*** objective: A statement of skills, knowledge, and attitudes to be
      acquired by the student. This holds an unique (to this course) ID
      identifier for a particular objective ID's are generated by the
      application (e.g., an LMS) that creates a CSF XML file
      (other elements may refer to this unique ID)-->
<!--ELEMENT objective ((externalMetadata*,assignmentRef*,identification,
      prerequisites?,completionReq?,extensions*,objective*) |
      objectiveAlias) >

<!--ATTLIST objective
      id ID      #REQUIRED >

<!--*** source: Authority or source of data model w/reference to a spec. if available
      e.g., "AICC AGRO10 v3.4", or ARMY TRADOC spec123, or "IMSBP v4.2"-->
<!--ELEMENT source (#PCDATA) >

<!--*** model: Name of a specific data model used by this course
      e.g., "cmi", or "ARMY314", or "IMS v1.0"-->
<!--ELEMENT model (#PCDATA) >

<!--*** location: URI Location-->
<!--ELEMENT location (#PCDATA) >

<!--*** property: Name/value pair extension for this course-->
<!--ELEMENT property (name,value) >

<!--*** title: Context specific title.
      May be used by an LMS system in menus, screens, etc.-->
<!--ELEMENT title (#PCDATA) >

<!--*** description: Context specific textual information about the course element.
      It may contain the purpose, scope, or summary. (Defined by course author)-->
<!--ELEMENT description (#PCDATA) >

<!--*** labels: Context specific local label (e.g., unit, chapter,
      learning step)-->
<!--ELEMENT labels (curricular?,developer?) >

```

```

<!--*** Time values or actions associated with this au in this context-->
<!ELEMENT timeLimit (maxTimeAllowed?,timeLimitAction?) >

<!--***\launch: Information needed by an LMS to launch an au-->
<!ELEMENT launch (location,parameterString?,dataFromLMS?) >

<!--*** score: Values to be used in this course context for tracking
score within an au-->
<!ELEMENT masteryScore (#PCDATA) >

<!--*** auAlias: Reference to a previously defined au
(permits one au to be used more than once within a course-->
<!ELEMENT auAlias EMPTY >

<!--ATTLIST auAlias
targetID ID #IMPLIED >

<!--*** assignmentRef: Reference to a particular block or au element
in the course structure hierarchy e.g.,
<assignmentRef "targetIDs="B1,A23"/>-->
<!ELEMENT assignmentRef EMPTY >

<!--ATTLIST assignmentRef
relation CDATA #IMPLIED
targetIDs IDREFS #IMPLIED >

<!--*** objectiveAlias: Reference to a previously defined objective
(permits one objective to be used more than once within a course)-->
<!ELEMENT objectiveAlias EMPTY >

<!--ATTLIST objectiveAlias
targetID IDREF #IMPLIED >

<!--*** name: Descriptive name of a course property extension
e.g., "difficulty" (as in degree of)-->
<!ELEMENT name (#PCDATA) >

<!--*** value: Value associated with the named extension
e.g., "easy"-->
<!ELEMENT value (#PCDATA) >

<!--*** curricular label: Local name of course element
e.g., "UNIT", "MODULE", "LEARNING STEP"-->
<!ELEMENT curricular (#PCDATA) >

<!--*** developer label: an organization-specific identifier (e.g., D509)-->
<!ELEMENT developer (#PCDATA) >

<!--*** maxTimeAllowed: The amount of time the student is allowed
to have in the current attempt on the lesson.-->
<!ELEMENT maxTimeAllowed (#PCDATA) >

<!--*** timeLimitAction: What the lesson is to do when the max time
allowed is exceeded. AICC examples: "exit", "continue",
"message", "continue".-->
<!ELEMENT timeLimitAction (#PCDATA) >

<!--*** parameterString: String of characters needed to successfully
launch a content au-->
<!ELEMENT parameterString (#PCDATA) >

<!--*** dataFromLMS: unconstrained (undefined) initialization data expected
by content when it is launched by the LMS.-->
<!ELEMENT dataFromLMS (#PCDATA) >

```

A.2 Course Structure Format Mapping to AICC Structure

CSF Element Name	AICC Name	Contextualized Definition	List	Level	Data Type
Global Properties	Properties	Information that applies to the course as a whole.	S	1	
GlobalProperties.externalMetadata (located in external metadata record)	--Creator	Name of the vendor and/or author of the course.	*	1	CMString256
GlobalProperties.externalMetadata (located in external metadata record)	--Identifier	Unique label for the course.	S	1	CMIdentifier
GlobalProperties.externalMetadata (located in external metadata record)	--System	Predominant authoring system used to create the course.	S	1	DString
(Root)block.identification.title GlobalProperties.externalMetadata (can also be located in external metadata record)	--Title	Common name given to course.	S	1	CMString256
GlobalProperties.externalMetadata (located in external metadata record)	--Level	Indicator of the complexity of structure and sequencing data.	S	1	CMString256
	--Max Block Members	Number of members in most populous block.	S	1	CMInteger
	--Max Objective Members	Number of members in most populous objectives relationship.	S	1	CMInteger
	--Total AU's	Total number of unique assignable units in the course.	S	1	CMInteger
	--Total Blocks	Total number of blocks in the course.	S	1	CMInteger
	--Total Objectives	Total number of objectives (simple and complex) in the course.	S	1	CMInteger
	--Total Complex Objectives	Total number of complex objectives in the course.	S	1	CMInteger
GlobalProperties.externalMetadata (located in external metadata record)	--Version	The revision number of the IEEE CMI Standard on which the course sequencing data is based.	S	1	CMString256
(Root)block.identification.description GlobalProperties.externalMetadata (can also be located in external metadata record)	--Description	Textual information about the course.	S	1	CMString4096
	--Max Normal	The maximum number of assignable units that may be taken for credit and incomplete.	S	3A	CMInteger
(root) block	Structure	Information on organization and sequencing of the course.	S	1	-
block	--Block	A grouping of related structural elements.	+	1	-
block.identification.labels.developer	-- --Identifier	Developer created unique label for block.	S	1	CMString256
Block id	-- --System Identifier	Course unique identifier created by the CMI system.	S	1	CMIdentifier
Block.identification.title	-- --Title	Commonly used name for the block.	S	1	CMString256
Block.identification.description	-- --Description	Textual summary of block's contents.	S	1	CMString4096

CSF Element Name	AICC Name	Contextualized Definition	List	Level	Data Type
Block.prerequisites	-- --Prerequisite	Expression identifying what a student must accomplish before beginning the block.	S	2	CMILogic
Block.completionReq	-- --Completions	What a student must do to gain a specific status for a block.	+	2	--
	-- --Requirement	Expression that may be evaluated as true or false.	S		CMILogic
	-- --Status	The credit given to a student when the requirements expression is true.	S	2	CMIVocabulary
	-- --Next AU	Forced assignment when status is true.	S	2	CMISIdentifier
	-- --Return to Next AU.	Forced assignment after leaving Next AU.	S	2	CMISIdentifier
Au	--Assignable Unit	Information relating to an assignable unit.	*	1	-
Au.identification.labels.developer	-- --Identifier	Developer created unique label for the assignable unit.	S	1	CMISString256
Au.id	-- --System Identifier	Course unique identifier created by the CMI system.	S	1	CMISIdentifier
Au.identification.title	-- --Title	Commonly used name for an assignable unit, block, objective, or complex objective.	S	1	CMISString256
Au.identification.description	-- --Description	Textual summary of unit's contents.	S	1	CMISString4096
Au.identification.label.curricular (in most cases)	-- --Type	Developer defined category for AU.	S	1	CMISString256
Au.launch.parameterString	-- --Launch Line	The string of characters needed to successfully launch an executable program.	S	1	CMISString256
Au.launch.location	-- --File Name	The full identifier of the files containing the content of the lesson.	+	1	CMISString256
	-- --Max Score	Largest raw score that can be reported by the lesson.	S	1	CMIDecimal
Au.masteryScore	-- --Mastery Score	The minimum raw score required for the student to pass the lesson.	S	1	CMIDecimal
Au.timelimit.maxTimeAllowed	-- --Max Time Allowed	Amount of time permitted for a student's single use of a lesson.	S	1	CMITimespan
Au.timelimit.timeLimitAction	-- --Time Limit Action	What the lesson should do when the max time allowed is exceeded.	S	1	CMIVocabulary
Au.externalMetadata (located in external metadata record)	-- --System Vendor	Authoring system used to create the lesson.	S	1	CMISString256
Au.launch.dataFromLMS	-- --Launch Data	Unique information required by the lesson's design.	S	1	CMISString4096
Au.prerequisites	-- --Prerequisite	Expression identifying what a student must accomplish before beginning the assignable unit.	S	2	CMILogic
	-- --Completions	What a student must do to gain a specific status for an assignable unit.	+	2	--

CSF Element Name	AICC Name	Contextualized Definition	List	Level	Data Type
Au.completionReq	-- --Requirement	Expression that may be evaluated as true or false.	S	2	CMILogic
	-- --Status	The credit given to a student when the requirements expression is true.	S	2	CMIVocabulary
	-- --Next AU	Forced assignment when status is true.	S	2	CMISIdentifier
	-- --Return to	Forced assignment after leaving Next AU.	S	2	CMISIdentifier
Au.objectivesRef	-- --Embedded Objectives	Objective that is in the assignable unit.	*	2	CMISIdentifier
(Root) objectives	Objectives	Measurable learning goal.	*	3B	-
Objective.identification.label.developer	--Identifier	Developer assigned unique label for objective.	S	3B	CMISString256
Objective id	--System Identifier	Course unique identifier created by the CMI system.	S	3B	CMISIdentifier
Objective.identification.title	--Title	Commonly used name for the objective.	S	3B	CMISString256
Objective.identification.description	--Description	Textual summary of the objective.	S	3B	CMISString4096
Objective.assignmentRef	--Member IDs	CMI assigned unique label for each course element in the objective.	*	3B	CMISIdentifier
Objective.completionReq	--Completions	What a student must do to gain a specific status for an objective	+	3B	--
	-- --Requirement	Expression that may be evaluated as true or false.	S	3B	CMILogic
	-- --Status	The credit given to a student when the requirements expression is true.	S	3B	CMIVocabulary

Appendix B – AICC API Specification

This appendix is excerpted in its entirety from AICC Document No. CMI001, *CMI Guidelines for Interoperability*, Version 3.0.1 (www.aicc.org). The excerpted portion that follows is also Section B in the AICC document.

Appendix B: API-based CMI Communication

B.1 Introduction

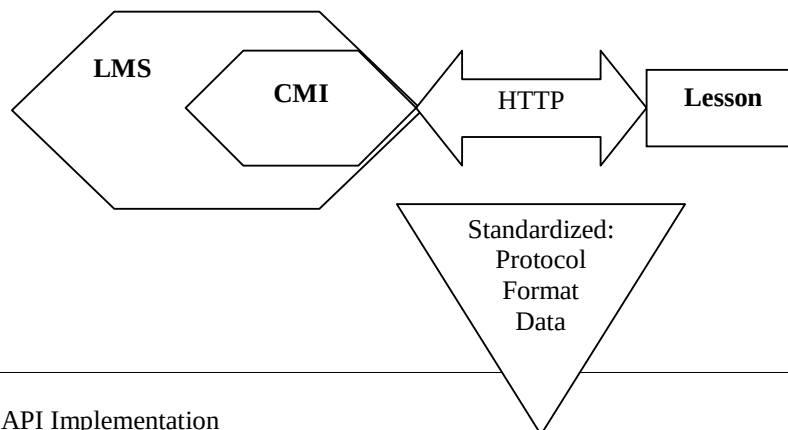
API

This appendix describes an Application Programming Interface (API) implementation for the AICC Computer Managed Instruction (CMI) standards. It defines an API which may be used over the Web by learning content to communicate with a Learning Management System (LMS). In this document a CMI system may be thought of as a separate management system or a subset of the functionality of an LMS.

This document also defines a mechanism for launching content that enables an LMS to “bind” the LMS neutral API to an LMS specific data transfer mechanism.

B.1.1 HTTP Implementation

Appendix A of this document, defines data exchange in terms of HACP (HTTP AICC CMI Protocol), an HTTP-based protocol. HACP has proven successful in commercial products and in large-scale LMS applications. However, the average content developer finds HACP difficult to understand and some LMS applications require protocols other than HTTP.

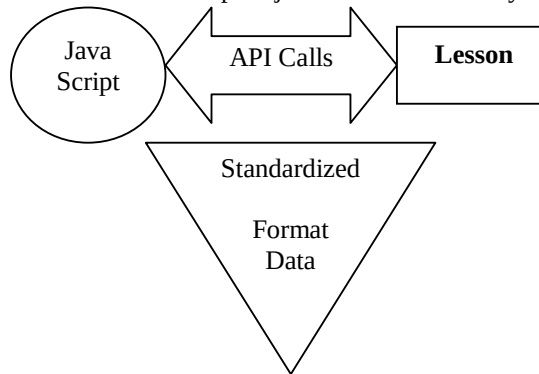


B.1.2 API Implementation

Description

This API standardizes the way content sends and receives information. It assumes that content will communicate using the widely supported ECMAScript calling conventions⁵. ECMAScript was selected as the method for implementing this API since nearly all browser platforms natively support it. This standard defines several calls, the data in these calls, and the format of that data.

The figure below illustrates what is standardized. Note that the communication of the ECMAScript with the LMS is outside the scope of this standard. Implementations of the communications of the JavaScript object with the LMS may vary from product to product.”

**Advantages**

There are several reasons for expanding the number of IEEE implementations to include an API standard in addition to an HTTP-based standard.

Generally speaking, an API is more abstract and implementation neutral than an approach based on a specific protocol such as HTTP. A content API essentially “hides” the implementation details of communication with an LMS.

Another advantage of an API is that it can make it easier for the content developer to understand and use communication with the LMS. Another advantage is the ability of a single API to work with several different data models. And finally, an API enables learning content, without being changed, to work with different data transfer mechanisms.

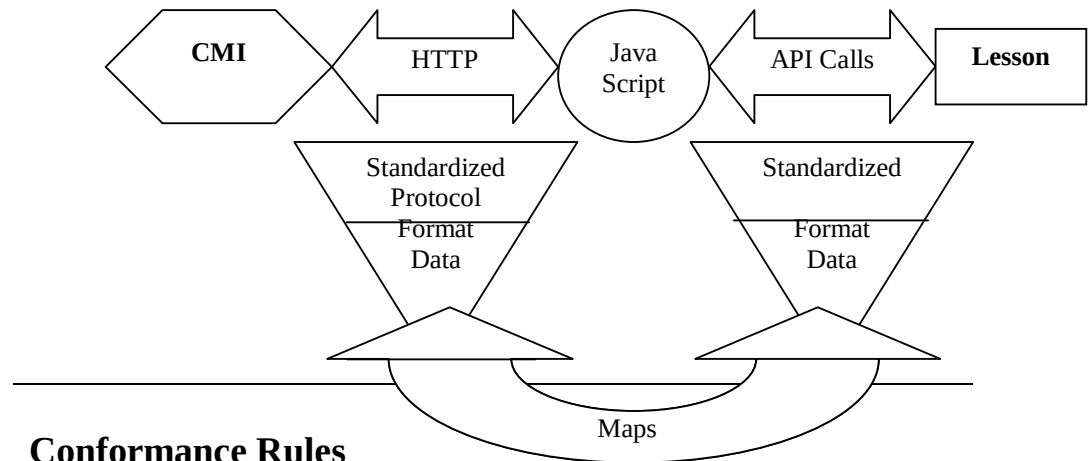
The approach defined in this document simplifies the creation of CMI compliant content by allowing content developers to think in terms of a higher-level API. This document also defines how to support the AICC and IEEE CMI data models. Although designed to support the CMI data model, the API defines a generic capability that can be applied to related data models as these are standardized.

Content using the API can be reused without modification with different data transfer mechanisms to suit application needs and with future versions of HACP as these are defined. The LMS dynamically determines which data transfer mechanism to use when content is launched.

B.1.3**Two Web Implementations**

⁵ ECMAScript is the ISO standard version of JavaScript. In this document the use of the term “JavaScript” is actually a reference to ECMAScript.

The API standards defined here may be used to complement the HTTP standards already defined in Appendix A. HTTP may be thought of as one possible implementation for communication. In other words, an LMS can support either an API or HTTP implementation or both implementations simultaneously.



B.2 Conformance Rules

Two viewpoints

Conformance to this standard may be looked at from two viewpoints, that of the learning content and that of the LMS.

B.2.1

Obligation

Levels

There are three levels of obligation for the API's and the data elements described in this standard:

- Mandatory
- Optional
- Extension

Obligations for the content and LMS are different.

For the LMS

Mandatory means that the LMS shall perform the action that the API calls for. If the action is to return a value to the content, then the call must succeed in returning a value of the proper format and range. Additionally, if the action is for the content to set a value, then that value must assume the form requested by the content, and be returned if requested in the future.

Optional means that a conforming LMS may not respond at all to the parameters in a get value or set value call. A conforming LMS may support many options.

An extension is an API or data element that is not described in this standard. Extensions may be supported by an LMS. However, extension API's may not perform the same function as a defined API; and extension data elements may not contain the same semantic values as defined data elements. If extensions are used to duplicate mandatory and optional features, the LMS is non-conforming.

For content

Mandatory means that the content shall execute the API. Only two API's are mandatory for content: LMSInitialize and LMSFinish.

Optional means that the content may execute the API with the specified parameter and value at least once. Furthermore, the parameter and value shall be in the proper format and range.

An extension is an API or data element that is not described in this standard. Extensions may be supported by learning content. However, extension API's may not perform the same function as a defined API; and extension data elements may not contain the same semantic values as defined data elements. If extensions are used to duplicate mandatory and optional features, the learning content is non-conforming.

B.2.2

CMI Responsibilities

The mechanism described here assumes a clean separation between the API function calls used in content and the API implementation. The API function calls are embedded in content. The API implementation is provided by the LMS when content is launched.

For browser and Web-based content, the LMS shall launch the content from a window that contains the API implementation, or must provide a parent frame that contains the API implementation.

The API implementation provided by the LMS must support all the API function calls described in this document as required.

The functions to “get” and “set” data element values are generic in nature and do not specify particular data elements. Data elements can be retrieved from the LMS using the LMSGetValue function and modified using a LMSSetValue function. Regardless of implementation details, if a data element is supported by the LMS, an LMSSetValue function call shall affect the value returned by a subsequent LMSGetValue function call on that same data element.

All return values shall be strings which are convertible to the designated data type.

The LMS shall support the ability of the content to “get” and “set” the “communication” data elements defined as mandatory in this standard. “Support” means that when the content executes an “ LMSGetValue “ on an element, a legal value of the proper format and type and range will be returned. When the content executes a legal “ LMSSetValue “ on a supported element, that value will be taken and the appropriate value returned when the next “ LMSGetValue “ on it is executed.

The LMS may support the ability of the content to “get” and “set” the optional data elements.

The LMS may also support extensions not defined in this standard as long as those extensions do not duplicate any mandatory or optional features. Additionally, the support of any extensions must not cause the failure of any content not using the extensions.

The table below summarizes the requirements for a conforming LMS.

LMS Conformance Requirements
- Supports the following transactions • LMSInitialize • LMSFinish • LMSGetValue • LMSSetValue • LMSCommit • LMSGetLastError • LMSGetErrorString • LMSGetDiagnostic - May support security transactions - Supports all mandatory elements • LMSGetValue shall succeed • LMSSetValue shall succeed - May support any or all optional elements • LMSGetValue may succeed • LMSSetValue shall succeed - May support extension elements if they do not duplicate defined mandatory or optional elements • LMSGetValue may succeed (or may fail) • LMSSetValue may succeed (or may be ignored) - Supported elements shall be proper type - Supported elements shall be in proper range - Keywords are all supported

B.2.3

Content Responsibilities

Content shall be able to call ECMAScript functions in a “foreign window”. The content does not have to be developed in ECMAScript but shall be able to call it. This capability enables the clean separation between the function calls used in content and the implementation of those function calls provided by a learning management system.

For conforming Assignable Units, content shall call the LMSInitialize function before calling any other API functions. If it calls the Initialize function successfully, it shall also call the LMSFinish function before it terminates, even if it does not call any other API functions.

Content may support the required set of “communication” data elements defined in the AICC/IEEE Web CMI specification.

The table below summarizes the requirements for conforming content.

Conformance Requirements for Content
Must support the following transactions: - Initialize - Zero or more transactions of: - LMSGetValue(X) - LMSSetValue(X,Y) - Other - Finish - X is an optional or extension data element - Y must be in range - Y must be the right type

B.2.3.1**Binding Mechanism**

Learning content shall communicate with an LMS system through a JavaScript API. This API will be part of an ECMAScript (JavaScript) object attached to either a parent window or the “opener” window for the HTML page. The content will obtain the API object by checking for its existence on any parent window or the opener window. The following JavaScript example demonstrates how this might work:

```
// returns the LMS API object (may be null if not found)
FindAPI(win)
{
  if (win.API != null)
    return win.API;
  else if (win.parent == null)
    return null;
  else
    return FindAPI(win.parent);
}

// obtain the LMS API
API = FindAPI(window);
If (API == null)
  API = FindAPI(window.opener);
```

B.2.3.2**Parameter Identification**

The parameters in the API function calls have two or more parts. Each part is separated by a period (dot). The first part is always the name of the data model. The second part is always the name of an element in the data model. Subsequent parts are either the name of an element in the data model, or a number, which refers to a location within the preceding data element which, is an array.

- datamodel.element
- datamodel.element.element
- datamodel.element.number.element
- datamodel.element.number.element.number

Data model indicates which data model the value or return value is based on. In this document the data model is “CMI” as defined in the AICC and IEEE CMI standards.

The highest level of element is sometimes referred to as a *Group* in the CMI data model. In this document the word “category” is used interchangeably with the word “group.” Each group element has a unique name in the CMI data model.

Element refers to a specific name in the CMI data model. In the AICC documentation, each element that is a sub-element or member of another element is referred to as a keyword or a field. Some sub-elements may have the same name. To enable precise identification, the element (sub-element) name must always be accompanied by the name of the group in which it appears.

Number is a simple integer that refers to the location in an array, if the named value is in an array. The first element in an array is 0.

B.3

API Set

The API's

The set of API function calls consists of the following:

- LMSInitialize()
- LMSFinish()
- LMSGetValue(cmi.group.element)
- LMSSetValue(cmi.group.element, value)
- LMSCommit()
- LMSGetLastError()
- LMSGetErrorString(errornumber)
- LMSGetDiagnostic(parameter)
- Security Request/Respond --- TBD

B.3.1

API General Rules

The following list summarizes the usage rules for the API.

- The function names are all case sensitive, and must always be expressed exactly as shown above.
 - The function parameters or arguments are case sensitive. All parameters are lower case.
 - The first symbol in the data element name identifies the data model. For example, “cmi” indicates the AICC/IEEE CMI data model. This expands the functionality of these API’s by allowing the same API to be used with other data models.
 - There are three reserved keywords. These are all lower case and preceded by an underscore.
 - `_version`
 - `_children`
 - `_count`
 - When `LMSGetValue` is executed, it returns the last set value if there was one.
-

B.3.2

Handling Lists

There are several data elements that appear in a list or an array. An example of this would be objective status. There may be more than one objective covered in a lesson, and a student may be allowed to experience an objective more than once.

To get or set values in a list, the index number may be used. The only time an index number may be omitted is when there is only one member in a potential list. Index numbering starts at 0. If a value is to be appended to the list, the Assignable Unit must know the last index number used.

If the student is entering the lesson for the second time, the `_count` keyword can be used to determine the current number of records in the list. For instance, to determine the number of objective records currently recorded, the following API would be used:

`LMSGetValue(“cmi.objective._count”)`

If the lesson does not know the count of the objective records, it can begin the current student count with 0. This would overwrite any information about objectives currently stored in the first index position. Overwriting or appending is a decision that is made by the lesson author when he creates the lesson.

Elements in a list are referred to with a dot-number notation (represented by `.n`). For instance the value of the status element in the first objective in a lesson would be referred to as `“cmi.objective.0.status”`. The status element in the fourth objective would be referred to as `“cmi.objective.3.status”`. If a student experienced the first objective twice, there could be two status’s associated with the first objective. These would be identified as `“cmi.objective.0.status.0”` and `“cmi.objective.0.status.1”`.

API Function Table

Function	Description	API Call	Return Value
Initialize	The content must call this function before calling any other API function. It indicates to the LMS system that the content is going to communicate. The LMS can take any initialization steps required in this function.	LMSInitialize()	A string convertible to CMIBoolean
Finish	The content must call this function before it terminates, if it successfully called LMSInitialize at any point. It signals to the LMS that the content has finished communicating. The content may not call any API function except LMSGetLastError after it calls LMSFinish	LMSFinish()	None
Get a value	This is used to determine values for various categories and elements in the CMI data model. Only one value is returned for each call. The category and/or element is named in the argument.	LMSGetValue(cmi.category) LMSGetValue(cmi.category.element)	A string convertible to appropriate data type
Set a value	This is how data categories and elements get values. The argument indicates which category or element is being set. Only one value may be set with a single function call.	LMSSetValue(cmi.category, value) LMSSetValue(cmi.category.element, value)	None
Send cache to LMS	If the ECMAScript is caching LMSSetValue values, this call requires that any values not yet sent to the LMS be sent.	LMSCommit(parameter)	None
Determine error code	The content must have a way of assessing whether or not any given API call was successful, and if it was not successful, what went wrong. This routine returns an error code from the previous API call. Each time an API function is called (with the exception of this one), the error code is reset in the API. The content may call this any number of times to retrieve the error code, and the code will not change until the next API call.	LMSGetLastError()	A string convertible to CMInteger
Obtain text related to error	This function enables the content to obtain a textual description of the error represented by the error code number.	LMSGetErrorString(errornumber)	CMString256
Determine vendor-specific diagnostics	This function enables vendor-specific error descriptions to be developed and accessed by the content. These would normally provide additional helpful detail regarding the error.	LMSGetDiagnostic(parameter)	CMString256
Security functions	-TBD-		

B.3.3**Initialize****Description**

This function indicates to the API that the learning content is going to communicate with the LMS. It allows the LMS to handle LMS specific initialization issues. It is called by content before it can call any other API function.

Syntax

LMSInitialize(parameter)

Parameter

Null. A null must be passed for conformance to this standard. This parameter is reserved for future extensions.

Return value

Boolean.
A “true” result indicates that the initialization was successful and a “false” result indicates that it was not.

Examples

LMSInitialize()
The learning content tells the API that the content wants to establish communication with the LMS. A typical return value is “true”.

B.3.4**Finish****Description**

The content must call this function before it terminates, if it successfully called LMSInitialize at any point. It signals to the LMS that the content has finished communicating. The content may not call any API function except LMSGetLastError after it calls LMSFinish. In other words, all LMSSetValue commands must be made before the LMSFinish call.

Syntax

LMSfinish(parameter)

Parameter

Null. A null must be passed for conformance to this standard. This parameter is reserved for future extensions.

Return value

None

B.3.5**Get a Value****Description**

This function allows content (the assignable unit) to obtain information from the LMS. It is used to determine

- Values for various categories (groups) and elements in the CMI data model.
- The version of the data model supported.
- Whether a specific category or element is supported.
- The number of items currently in an array or list of elements.

The complete data element name and/or keywords are provided as a parameter. The current value of that parameter is returned. Only one value -- always a string -- is returned for each call.

Syntax

LMSGetValue(parameter)

Parameters

cmi.element.element

Returns the value of the named sub-element

cmi._version

The _version keyword is used to determine the version of the data model supported by the LMS.

cmi.element._count

The _count keyword is used to determine the number of elements currently in an array. This number is not changed by use of the LMSCommit call.

cmi.element._children

The _children keyword is used to determine all of the elements in a group or category that are supported by the LMS.

Return value

All return values are strings which can be converted to the appropriate type.

For LMSGetValue(cmi.group.element) the return value is a string representing the current value of the requested element or group.

For LMSGetValue(cmi._version) the return value is a string representing the version of the data model supported by the LMS.

For LMSGetValue(cmi.group._children) the return value is a comma separated list of all the element names in the specified group or category that are supported by the LMS. If an element has no children, but is supported, an empty string is returned. If an element is not supported, there is no return. A subsequent request for last error [LMSGetLastError()] can verify that the element is not supported.

For LMSGetValue(cmi.group._count) the return value is an integer that indicates the number of items currently in an element list or array.

Examples

LMSGetValue("cmi.core.student_name")

A typical return value might be "Jackson Hyde".

LMSGetValue("cmi.core.lesson_status")

A typical return value might be "Incomplete".

LMSGetValue(cmi._version)

The current draft standard for the IEEE document defining the CMI data model is entitled *Draft Standard for Computer Managed Instruction*, and has an ID of P1484.11.2 and a version number of 2.2. This call returns the version number of that IEEE document which is 2.2.

LMSGetValue("cmi.student_preferences._children")

This is a request for category support. One typical return value would be, "audio,speed,text". If there is no return, preferences are probably not supported. An additional API call to determine the last error could verify this.

LMSGetValue("cmi.comments._children")

The comments element has no children. A zero length string indicates that comments are supported. No return implies no support for comments.

LMSGetValue("cmi.evaluation.comments._children")

This is a data element request. The empty string means that any list of student-generated comments will be forwarded to the LMS. Further, this means the LMS will, when requested, produce a file matching the description in the "Comments File" chapter of this document.

B.3.6

Set a Value

Description

This function allows the learning content (the assignable unit) to send information to the API. The API may be designed to immediately forward the information to the LMS, or it may be designed to forward information based on some other approach. For instance, the API could accumulate the information and forward everything to the LMS when the LMSFinish call is executed by the learning content.

This function is used to set the current values for various categories (groups) and elements in the CMI data model.

The data element name and its group are provided as a parameter. The current value of that parameter is included in the call. Only one value is sent with each call.

Syntax

LMSSetValue(parameter, value)

Parameter

This is the name of a fully qualified atomic element defined in the CMI Data Model. The argument is case sensitive. The argument is a string surrounded by quotes.

The following represents some forms this parameter may take.

cmi.element

This is the name of a category or group defined in the CMI Data Model. An example is "cmi.comments".

cmi.element.element

This is the name of an element defined in the CMI Data Model. An example is "cmi.core.student_name".

cmi.element.n.element

The value of the sub-element in the nth-1 member of the element array (zero-based indexing is used).

Value

This is a string which must be convertible to the data type defined in this standard for the element identified in the first parameter.

Return value None

B.3.7 Send Cache to CMI

Description If the ECMAScript is caching LMSSetValue values, this call requires that any values not yet sent to the LMS be sent.

In some implementations, the ECMAScript may send the set values to the LMS as soon as they are received, and not cache them locally. In such implementations, this API is redundant and would result in no additional action from the ECMAScript.

Syntax LMSCommit(parameter)

Parameter Null. A null must be passed for conformance to this standard. This parameter is reserved for future extensions.

Return value None

B.3.8 Determine Error Code

Description The learning content must have a way of assessing whether or not any given API call was successful, and if it was not successful, what went wrong. This routine returns an error code from the previous API call. Each time an API function is called (with the exception of this one, LMSGetErrorString, and LMSGetDiagnostic -- the error functions), the error code is reset in the API. The content may call the error functions any number of times to retrieve the error code, and the code will not change until the next API call.

Syntax LMSGetLastError()

Parameter None.

Return value The return values are integer numbers that identify errors falling into the following categories:

- 100 General errors
- 200 Syntax errors
- 300 LMS errors
- 400 Data model errors

The following codes are available for error messages:

- 0. No error
- 101. General exception
- 201. Invalid argument error
- 202. Element cannot have children
- 203. Element not an array – cannot have count
- 204. Element cannot have a value
- 301. Not initialized
- 401. Not implemented error
- Additional codes TBD

B.3.9**Obtain Text Related to Error**

Description

This function enables the content to obtain a textual description of the error represented by the error code number.

Syntax

LMSGetErrorString(errornumber)

Parameter

An integer number representing an error code..

Return value

A string that represents the verbal description of an error.

B.3.10**Determine Vendor-specific Diagnostics**

Description

This function enables vendor-specific error descriptions to be developed and accessed by the content. These would normally provide additional helpful detail regarding the error.

Syntax

LMSGetDiagnostic(parameter)

Parameter

The parameter may take one of two forms.

- An integer number representing an error code. This requests additional information on the listed error code.
- Null value. This requests additional information on the last error that occurred.

Return value

The return value is a string that represents any vendor-desired additional information relating to either the requested error or the last error.

B.3.11**Security Request/Respond -- TBD**

B.4 LMS to Lesson Communications

The following table represents a subset of the Data Model defined in this document, which also contains more complete definitions for each term. The missing data elements are those required for a file-based CMI, but not required for an API-based CMI.

The Mult column indicates whether the data element may be an array (Arr) or is always a single value (SV). This obligation column represents the obligations of the LMS, not the lesson or learning content. All data elements are optional for the lesson.

Table of LMS to Lesson Communications

Name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
core	Information required to be furnished by all CMI systems. What all lessons may depend upon at start up.	SV	Man	LMSGetValue("cmi.core._children")	CMIStrng256
--student_id	Unique alpha-numeric code/identifier that refers to a single user of the CMI system.	SV	Man	LMSGetValue("cmi.core.student_id")	CMIIentifier
--student_name	Normally, the official name used for the student on the course roster. A complete name, not just a first name.	SV	Man	LMSGetValue("cmi.core.student_name")	CMIStrng256
--lesson_location	This corresponds to the lesson exit point passed to the CMI system the last time the student experienced the lesson.	SV	Man	LMSGetValue("cmi.core.lesson_location")	CMIStrng256
--credit	Indicates whether the student is being credited by the CMI system for his performance (pass/fail and score) in this lesson.	SV	Man	LMSGetValue("cmi.core.credit")	CMIVocabulary
--lesson_status	This is the current student status as determined by the CMI system, and sent to the lesson when it is launched.	SV	Man	LMSGetValue("cmi.core.lesson_status")	CMIVocabulary
--entry	Indication of whether the student has been in the lesson before.	SV	Man	LMSGetValue("cmi.core.entry")	CMIVocabulary

Table of LMS to Lesson (cont.)

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
--score	Indication of the performance of the student during his last attempt on the lesson.	SV	Man	LMSGetValue("cmi.core.score_children")	CMIStrng256
-- raw	Numerical representation of student performance in lesson. May be unprocessed raw score.	SV	Man	LMSGetValue("cmi.core.score.raw")	CMIDecimal CMIBlank
-- --max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSGetValue("cmi.core.score.max")	CMIDecimal CMIBlank
-- --min	The minimum score that the student could have achieved.	SV	Opt	LMSGetValue("cmi.core.score.min")	CMIDecimal CMIBlank
--total_time	Accumulated time of all the student sessions in the lesson.	SV	Man	LMSGetValue("cmi.core.time")	CMITimespan
--lesson_mode	Identification of student-related information that may be used to change the behavior of the lesson.	SV	Opt	LSMGetValue("cmi.core.lesson_mode")	CMIVocabulary
suspend_data	Unique information generated by the lesson during previous uses, that is needed for the current use.	SV	Man	LMSGetValue("cmi.suspend_data")	CMIStrng4096
launch_data	Unique information generated at the lesson's creation that is needed for every use.	SV	Man	LMSGetValue("cmi.launch_data")	CMIStrng4096
comments	Instructor comments directed at the student that the lesson may present to the student when appropriate.	SV	Opt	LMSGetValue("cmi.comments")	CMIStrng4096

Table of LMS to Lesson (cont.)

name	Contextualized Definition	Cont Obl	LMS Obl	Typical API Calls	Return Value
evaluation	Assignable units may be able to generate detailed student-performance/lesson-evaluation information. This category identifies if this functionality is supported by the LMS.	SV	Opt	LMSGetValue("cmi.evaluation._children")	CMIString256
--course_id	Alpha numeric sequence that provides a unique label for a course.	SV	Opt	LMSGetValue("cmi.evaluation.course_id")	CMIIdentifier
--comments	Identifies if the student's comments on a lesson can be collected and made available by the LMS in a separate file.	SV	Opt	LMSGetValue("cmi.evaluation.comments.")	CMIBoolean
--interactions	Identifies what detailed information of a student's interactions in a lesson can be collected.	SV	Opt	LMSGetValue("cmi.evaluation.interactions._children")	CMIString256
--objectives_status	Identifies what detailed information on lesson objectives can be collected.	SV	Opt	LMSGetValue("cmi.evaluation.objectives_status._children")	CMIString256
--path	Identifies what detailed information can be collected on the path through the lesson taken by the student.	SV	Opt	LMSGetValue("cmi.evaluation.path._children")	CMIString256
--performance	Identifies what detailed information can be collected, on the student's performance in complex scenarios, such as simulations.	SV	Opt	LMSGetValue("cmi.evaluation.performance._children")	CMIString256
objectives	Identifies how the student has performed on individual objectives covered in the lesson.	Arr	Opt	LMSGetValue("cmi.objectives._count") LMSGetValue("cmi.objectives._children")	CMIInteger CMIInteger
--id	A developer defined, lesson-specific identifier for an objective.	SV	Opt	LMSGetValue("cmi.objectives.n.id")	CMIIdentifier

Table of LMS to Lesson (cont.)

name	Contextualized Definition	Cont Obl	LMS Obl	Typical API Calls	Return Value
--scores	The score obtained by the student after each attempt to master the objective.	Arr	Opt	LMSGetValue("cmi.objectives.n.scores._count")	CMIString256 CMIInteger
--raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score.	SV	Opt	LMSGetValue("cmi.objectives.n.scores.n.raw")	CMIDecimal CMIBlank
--max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSGetValue("cmi.objectives.n.scores.n.max")	CMIDecimal, CMIBlank
--min	The minimum score that the student could have achieved.	SV	Opt	LMSGetValue("cmi.objectives.n.scores.n.min")	CMIDecimal CMIBlank
--statuses	The status obtained by the student after each attempt to master the objective.	Arr	Opt	LMSGetValue("cmi.objectives.n.status.n")	CMIVocabulary
student_data	Information to support customization of a lesson based on a student's performance.	SV	Opt	LMSGetValue("cmi.student_data_children")	CMIString256
--attempt_number	Number of times the student has been in, or previously used the lesson.	SV	Opt	LMSGetValue("cmi.student_data_attempt_number")	CMIInteger
--mastery_score	The passing score, as determined outside the lesson.	SV	Opt	LMSGetValue("cmi.student_data_mastery_score")	CMIDecimal
--max_time_allowed	The amount of time the student is allowed to have in the current attempt on the lesson.	SV	Opt	LMSGetValue("cmi.student_data_max_time_allowed")	CMITimespan
--time_limit_action	What the lesson is to do when the max time allowed is exceeded.	SV	Opt	LMSGetValue("cmi.student_data_time_limit_action")	CMIVocabulary
--attempt_records	Student's performance after previous times in the lesson.	Arr	Opt	LMSGetValue("cmi.student_data_attempt_records_children") LMSGetValue("cmi.student_data_attempt_records_count")	CMIString256 CMIInteger
--lesson_scores	The score obtained by the student after each previous attempt.	Arr	Opt	LMSGetValue("cmi.student_data_attempt_records.n.lesson_score")	CMIDecimal
--lesson_statuses	Indication of the status of the lesson after each attempt.	Arr	Opt	LMSGetValue("cmi.student_data_attempt_records.n.lesson_status")	CMIVocabulary

Table of LMS to Lesson (cont.)

name	Contextualized Definition	Cont Obl	LMS Obl	Typical API Calls	Return Value
student_demographics	Student attributes possessed before entering the course.	SV	Opt	LMSGetValue("cmi.student_demographics._children")	CMIString256
--city	Portion of student's current address.	SV	Opt	LMSGetValue("cmi.student_demographics.city")	CMIString256
--class	A predefined training group to which a student belongs.	SV	Opt	LMSGetValue("cmi.student_demographics.class")	CMIString256
--company	Student's place of employment.	SV	Opt	LMSGetValue("cmi.student_demographics.company")	CMIString256
--country	Portion of student's current address.	SV	Opt	LMSGetValue("cmi.student_demographics.country")	CMIString256
--experience	Information on the student's past that might be required by a lesson to determine what to present, or what presentation strategies to use.	SV	Opt	LMSGetValue("cmi.student_demographics.experience")	CMIString256
--familiar_name	An informal title that may be used to address the student.	SV	Opt	LMSGetValue("cmi.student_demographics.familiar_name")	CMIString256
--instructor_name	Name of the person responsible for the student's understanding of the material in the lesson.	SV	Opt	LMSGetValue("cmi.student_demographics.instructor_name")	CMIString256
--title	Title of the position or the degree currently held by the student.	SV	Opt	LMSGetValue("cmi.student_demographics.title")	CMIString256
--native_language	The language used in the student's country of origin.	SV	Opt	LMSGetValue("cmi.student_demographics.native_language")	CMIString256
--state	Segment of a country, also called province, district, canton, etc.	SV	Opt	LMSGetValue("cmi.student_demographics.state")	CMIString256
--street_address	Portion of student's current address.	SV	Opt	LMSGetValue("cmi.student_demographics.street_address")	CMIString256
--telephone	Telephone number of a student.	SV	Opt	LMSGetValue("cmi.student_demographics.telephone")	CMIString256
--years_experience	Number of years the student has performed in current or similar position.	SV	Opt	LMSGetValue("cmi.student_demographics.years_experience")	CMIString256

Table of LMS to Lesson (cont.)

name	Contextualized Definition	Cont Obl	LMS Obl	Typical API Calls	Return Value
student_preference	Student selected options that are appropriate for subsequent lessons.	SV	Opt	LMSGetValue("cmi.student_preference._children")	CMISString256
--audio	Sound on/off and volume control.	SV	Opt	LMSGetValue("cmi.student_preference.audio")	CMISInteger
--language	Identifies in what language the information should be delivered.	SV	Opt	LMSGetValue("cmi.student_preference.language")	CMILocale
--lesson_type	Indicates suitability of preferences to current lesson.	SV	Opt	LMSGetValue("cmi.student_preference.lesson_type")	CMISString256
--speed	Pace of content delivery.	SV	Opt	LMSGetValue("cmi.student_preference.speed")	CMISInteger
--text	Written content visibility control.	SV	Opt	LMSGetValue("cmi.student_preference.text")	CMISInteger
--text_color	Written content foreground and background hue.	SV	Opt	LMSGetValue("cmi.student_preference.text_color")	CMISString256
--text_location	Position of text window on the screen.	SV	Opt	LMSGetValue("cmi.student_preference.text_location")	CMISString256
--text_size	Magnitude of the written content characters on screen.	SV	Opt	LMSGetValue("cmi.student_preference.text_size")	CMISString256
--video	Motion picture tint and brightness on the screen.	SV	Opt	LMSGetValue("cmi.student_preference.video")	CMISString256
--windows	Size and location of video, help, glossary, and so forth windows.	SV	Opt	LMSGetValue("cmi.student_preference.n.windows")	CMISString256

B.5 Lesson to LMS Communication

Table of Lesson to LMS Communication

name	Contextualized Definition	Mult	LMS Obl	API Call	Value Data Type
core	Information required by the CMI system to function.	SV	Man		-
--lesson_location	This identifies the point where the student leaves the lesson.	SV	Man	LMSSetValue("cmi.core.lesson_location", value)	CMIStrng256
--lesson_status	This is the student status when he leaves the lesson.	SV	Man	LMSSetValue("cmi.core.lesson_status", value)	CMIVocabulary
--exit	An indication of how or why the student left the lesson.	SV	Man	LMSSetValue("cmi.core.exit", value)	CMIVocabulary
--score	Indication of the performance of the student during his time in the lesson.	SV	Man		
-- --raw	Numerical representation of student performance in lesson. May be unprocessed raw score.	SV	Man	LMSSetValue("cmi.core.score.raw", value)	CMIDecimal
-- --max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSSetValue("cmi.core.score.max")	CMIDecimal, CMIBlank
-- --min	The minimum score that the student could have achieved.	SV	Opt	LMSSetValue("cmi.core.score.min")	CMIDecimal, CMIBlank
--session_time	Time spent in the lesson during the session that is ending.	SV	Man	LMSSetValue("cmi.core.time", value)	CMITimespan
suspend_data	Unique information generated by the lesson, that is needed for future uses. Passed to the CMI system to hold and to return the next time the student starts this lesson.	SV	Man	LMSSetValue("cmi.suspend_data", value)	CMIStrng4096
comments	Student's written remarks recorded during the current use of the lesson.	array	Opt	LMSSetValue("cmi.comments.n ", value)	CMIStrng4096

Table of Lesson to LMS Communication (cont.)

name	Contextualized Definition	Cont Obl	LMS Obl	API Call	Value Data Type
objectives	Identifies how the student has performed on individual objectives covered in the lesson.	Arr	Opt		-
--id	A developer defined, lesson-specific identifier for an objective.	SV	Opt	LMSSetValue("cmi.objectives.n.id", value)	CMIdentifier
--scores	The score obtained by the student after each attempt to master the objective.	Arr	Opt		
-- --raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score.	SV	Opt	LMSSetValue("cmi.objectives.n.scores.n.raw", value)	CMIDecimal
-- --max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSSetValue("cmi.objectives.n.scores.n.max", value)	CMIDecimal
-- --min	The minimum score that the student could have achieved.	SV	Opt	LMSSetValue("cmi.objectives.n.scores.n.min", value)	CMIDecimal
--status	The status obtained by the student after the each attempt to master the objective.	Arr	Opt	LMSSetValue("cmi.objectives.n.status.n", value)	CMIVocabulary
student_data	Information on student performance for each attempt on a selected segment of the lesson without leaving the lesson.	SV	Opt	--	-
--tries_during_lesson	Total number of efforts to complete the lesson or selected segment.	SV	Opt	LMSSetValue("cmi.student_data.tries_during_lesson", value)	CMInteger
--tries	Data related to each try.	Arr	Opt		
-- --score	The score at the completion of each attempt.	SV	Opt		
-- -- --raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.score.raw", value)	CMIDecimal
-- -- --max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.score.max", value)	CMIDecimal
-- -- --min	The minimum score that the student could have achieved.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.score.min", value)	CMIDecimal

Table of Lesson to LMS Communication (cont.)

name	Contextualized Definition	Cont Obl	LMS Obl	API Call	Value Data Type
-- status	The status of the lesson or segment after each attempt.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.status", value)	CMIVocabulary
-- time	Length of time required for each attempt on a lesson or segment.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.time", value)	CMITimespan
student_preferences	Student selected options that are appropriate for subsequent lessons.	SV	Opt	--	-
-- language	Identifies in what language the information should be delivered.	SV	Opt	LMSSetValue("cmi.student_preference.language", value)	CMILocale
-- lesson_type	Indicates suitability of preferences to current lesson.	SV	Opt	LMSSetValue("cmi.student_preference.lesson_type", value)	CMISString256
-- speed	Pace of content delivery.	SV	Opt	LMSSetValue("cmi.student_preference.speed", value)	CMISInteger
-- text	Written content visibility control.	SV	Opt	LMSSetValue("cmi.student_preference.text", value)	CMISInteger
-- text_color	Written content foreground and background hue.	SV	Opt	LMSSetValue("cmi.student_preference.text_color", value)	CMISString256
-- text_location	Position of text window on the screen.	SV	Opt	LMSSetValue("cmi.student_preference.text_location", value)	CMISString256
-- text_size	Magnitude of the written content characters on screen.	SV	Opt	LMSSetValue("cmi.student_preference.text_size", value)	CMISString256
-- video	Motion picture tint and brightness on the screen.	SV	Opt	LMSSetValue("cmi.student_preference.video", value)	CMISString256
-- windows	Size and location of video, help, glossary, and so forth windows.	Arr	Opt	LMSSetValue("cmi.student_preference.n.windows", value)	CMISString256

B.6 Student Data Collection

Student Data Collection Table

name	Contextualized Definition	Mult	API Call	Value Data Type
lesson_id	Alphanumeric label supplied by the developer.	SV	LMSSetValue("cmi.evaluation.lesson_id", value)	CMISString256
date	The calendar day on which the data is created.	SV	LMSSetValue("cmi.evaluation.date", value)	CMIDate
comments	Freeform feedback from the student. More structured representation than the comments in the Lesson to LMS table.	Arr		-
--time	Indication of when the comment is made.	SV	LMSSetValue("cmi.evaluation.comments.n.time ", value)	CMITime
--location	Indication of where in the lesson the comment is made.	SV	LMSSetValue("cmi.evaluation.comments.n.location ", value)	CMISString256
--content	The recorded statement of a student.	SV	LMSSetValue("cmi.evaluation.comments.n.content ", value)	CMISString4096
interactions	A recognized and recordable input or group of inputs from the student to the computer	Arr		-
--id	Unique alphanumeric label created by the lesson developer.	SV	LMSSetValue("cmi.interactions.n.id ", value)	CMISString256
--objective_ids	Indication of any objectives associated with the interaction.	Arr	LMSSetValue("cmi.interactions.n.objective_ids.n", value)	CMISString256
--time	Indication of when the interaction is available to the student.	SV	LMSSetValue("cmi.interactions.n.time ", value)	CMITime
--type	Indication of which category of interaction is recorded.	SV	LMSSetValue("cmi.interactions.n.type ", value)	CMIVocabulary
-- responses	Expected student feedback in the interaction.	Arr		
-- --description	Definition of possible student response.	SV	LMSSetValue("cmi.interactions.n.response.n.description", value)	CMIFeedback
-- --value	How the system judges the described response.	SV	LMSSetValue("cmi.interactions.n.response.n.value", value)	CMIVocabulary

Student Data Collection Table

name	Contextualized Definition	Obl	API Call	Value Data Type
--weighting	Factor that is used to identify the relative importance of one interaction compared to another.	SV	LMSSetValue("cmi.interactions.n.weighting ", value)	CMIDecimal
--student_response	Description of the computer-measurable action of a student in an interaction.	SV	LMSSetValue("cmi.interactions.n.student_response ", value)	CMIFeedback
--result	Judgment of the of the student's response.	SV	LMSSetValue("cmi.interactions.n.result ", value)	CMIVocabulary
--latency	The time from the presentation of the stimulus to the completion of the measurable response.	SV	LMSSetValue("cmi.interactions.n.latency ", value)	CMITimespan
objectives	Information about a student's performance on lesson objectives. The only additional data that is not described in normal Lesson to LMS communication is mastery_time.	Arr	-	-
--mastery_time	Chronological period spent in the objective.	SV	LMSSetValue("cmi.objectives.n.mastery_time ", value)	CMITimespan
paths	Description of the sequence of events the student experienced in the lesson.	Arr	-	-
--location_id	Identification of where the student is in the lesson.	SV	LMSSetValue("cmi.path.n.location_id", value)	CMIString256
--time	Indication of when the student entered the lesson segment.	SV	LMSSetValue("cmi.path.n.time", value)	CMITime
--status	A record of the student's performance in a segment each time he leaves that element	SV	LMSSetValue("cmi.path.n.status", value)	CMIVocabulary
--why_left	The reason a student departed an element in the lesson.	SV	LMSSetValue("cmi.path.n.why_left", value)	CMIVocabulary
--time_in_element	How long the student spent in the element.	SV	LMSSetValue("cmi.path.n.time_in_element", value)	CMITimespan

B.7

Data Types

Description	These definitions are for the data types used to describe the format of each data element. All of the data types have the first three characters of “CMI” to clearly indicate they are data types that may be unique to the CMI data model.
CMIBlank	An empty string.
CMIBoolean	A vocabulary of two words. true or false
CMIDate	A period in time of one day, defined by year, month, and day in the following numerical format YYYY/MM/DD.
CMIFeedback	Structured description of student response in an interaction.
CMIDecimal	Number which may have a decimal point. If not preceded by a minus sign, the number is presumed to be positive. Examples are “2” and “2.2”.
CMIIentifier	Alphanumeric group of characters with no white space or unprintable characters in it. Maximum of 255 characters.
CMILocale	A country and a language.
CMIIinteger	An integer number from 0 to 65536.
CMISIdentifier	CMI System Identifier: Alphanumeric group of characters that begins with a single letter: A, B, or J and ends with an integer number. One to five numerals may follow the letter. See <i>System Identifier</i> .
CMISInteger	A signed integer number from –32768 to +32768.
CMISString256	A set of ASCII characters with a maximum length of 255 characters.
CMISString4096	A set of ASCII characters with a maximum length of 4096 characters.
CMITime	A chronological point in a 24 hour clock. Identified in hours, minutes and seconds in the format: HH:MM:SS.S Hours and seconds shall contain two digits. Seconds shall contain 2 digits with an optional decimal point and additional digits.
CMITimespan	A length of time in hours, minutes, and seconds shown in the following numerical format: HHHH:MM:SS.S Hours and seconds shall contain two or more digits. Hours has a maximum of 4 digits. Minutes shall consist of 2 digits. Seconds shall contain 2 digits with an optional decimal point and additional digits.
CMIVocabulary	Used to attach specific vocabularies within contexts in a schema. Vocabulary words must be complete and exact matches to those below. Single letters and abbreviations may not be used in API communication. The following are vocabularies included in the CMI Data Model:

Vocabulary Type	Members of Vocabulary	
Mode	normal	review
	browse	
Status	passed	completed
	failed	incomplete
	browsed	not attempted
Exit	time-out	suspend
	logout	
Why-left	student selected	lesson directed
	exit	directed departure
Credit	credit	no credit
Entry	ab-initio	resume
Time Limit Action	exit	continue
	message	no message
Interaction	true-false	multiple choice
	fill in the blank	matching
	simple performance	likert
	sequencing	unique
	numeric	
Result	correct	wrong
	unanticipated	neutral
	x.x (CMIDecimal)	

B.8 Data Comparison

Description

The following tables compare the Group and Keyword names used in file-based communication, with the data element names used in API communications. The tables indicate where there are differences and

- Why a new element was created
- Why a keyword was deleted
- Why a group or keyword name was changed.

Table of CMI to Lesson Communication

Data Model Name	Group/Keyword Title	Why Different
core	[Core]	
--student_id	Student_ID	
--student_name	Student_Name	
	Output_File	Only needed for file-based communication.
--lesson_location	Lesson_Location	
--credit	Credit	
--lesson_status	Lesson_Status	
--entry	status flag	Need separate variable for every value
--score	Score	
-- --raw		Need separate variable for every value
-- --max		Need separate variable for every value
-- --min		Need separate variable for every value
--total_time	Time	Avoid get and set time ambiguity.
--lesson_mode	Lesson_Mode	
suspend_data	[Core_Lesson]	Name better describes the data.
launch_data	[Core_Vendor]	Name better describes the data.
comments	[Comments]	
evaluation	[Evaluation]	
--course_id	Course_ID	
--comments	Comments_File	API communication does not use files.
--interactions	Interactions_File	API communication does not use files.
--objectives_status	Objectives_Status_File	API communication does not use files.
--path	Path_File	API communication does not use files.
--performance	Performance_File	API communication does not use files.

Table of CMI to Lesson Communication (cont.)

Data Model Name	Group/Keyword Title	Why Different
objectives	[Objectives_Status]	Status is only one element in this group.
--id	J_ID.1	Objectives relationship established by parent name (i.e., objectives.n.id)
--scores	J_Score.1	Plural convention for possible array.
--raw		Need separate variable for every value
--max		Need separate variable for every value
--min		Need separate variable for every value
--statuses	J_Status.1	Objectives relationship established by parent name (i.e., objectives.n.status)
student_data	[Student_Data]	
--attempt_number	Attempt_Number	
--mastery_score	Mastery_Score	
--max_time_allowed	Max_Time_Allowed	
--time_limit_action	Time_Limit_Action	
--attempt_records		Must have a name with no value to enable children.
--lesson_scores	Score.1	Plural for array convention. Added lesson for consistency with status.
--lesson_statuses	Lesson_Status.1	Plural convention for possible array.
student_demographics	[Student_Demographics]	
--city	City	
--class	Class	
--company	Company	
--country	Country	
--experience	Experience	
--familiar_name	Familiar_Name	
--instructor_name	Instructor_Name	
--title	Job_Title	Generalize person's title.
--native_language	Native_Language	
--state	State	
--street_address	Street_Address	
--telephone	Telephone	
--years_experience	Years_Experience	

Table of CMI to Lesson Communication (cont.)

Data Model Name	Group/Keyword Title	Why Different
student_preference	[Student_Preferences]	Singular because not an array.
--audio	Audio	
--language	Language	
--lesson_type	Lesson_Type	
--speed	Speed	
--text	Text	
--text_color	Text_Color	
--text_location	Text_Location	
--text_size	Text_Size	
--video	Video	
--windows	Window.1	Plural convention for arrays.

Table of Lesson to CMI Communication

Data Model Name	Group/Keyword Title	Why Different
core	[Core]	
--lesson_location	Lesson_Location	
--lesson_status	Lesson_Status	
--exit	status flag	Need separate variable for every value
--score	Score	
-- --raw		Need separate variable for every value
-- --max		Need separate variable for every value
-- --min		Need separate variable for every value
--session_time	Time	Avoid get and set time ambiguity.
suspend_data	[Core_Lesson]	Name better describes the data.
comments	[Comments]	
objectives	[Objectives_Status]	Status is only one element in this group.
--id	J_ID.1	Objectives relationship established by parent name (i.e., objectives.n.id)
--scores	J_Score.1	Plural convention for possible array.
-- --raw		Need separate variable for every value
-- --max		Need separate variable for every value
-- --min		Need separate variable for every value
--statuses	J_Status.1	Objectives relationship established by parent name (i.e., objectives.n.status)

Table of Lesson to CMI Communication (cont.)

Data Model Name	Group/Keyword Title	Why Different
student_data	[Student_Data]	
--tries_during_lesson	Tries_During_Lesson	
--tries		Need name for parent.
-- --score	Try_Score.1	Relationship to tries established by parent name.
-- -- --raw		Need separate variable for every value
-- -- --max		Need separate variable for every value
-- -- --min		Need separate variable for every value
-- --status	Try_Status.1	Relationship to tries established by parent name.
-- --time	Try_Time.1	Relationship to tries established by parent name.
student_preference	[Student_Preferences]	Singular for non-array.
--audio	Audio	
--language	Language	
--lesson_type	Lesson_Type	
--speed	Speed	
--text	Text	
--text_color	Text_Color	
--text_location	Text_Location	
--text_size	Text_Size	
--video	Video	
--windows	Window.1	

Table of Lesson Evaluation Data

In a web environment all Lesson Evaluation information must pass through the CMI system before it can be stored in standard files. Each of the fields in the set of Lesson Evaluation files becomes just another data element that is being passed to the CMI. The CMI is responsible for assembling this information plus any additional information that is required from the Lesson to LMS table to create the files specified in this standard.

Data Model Name	Field Title	Why Different
	Course_ID	CMI already has this information. Do not need to return it to the CMI.
	Student_ID	CMI already has this information. Do not need to return it to the CMI.
lesson_id	Lesson_ID	
date	Date	
comments	Comments File	
--time	Time	
--location	Location	
--content	Comment	Content of comment. Keeping comment would be redundant.
interactions	Interactions File	
	Course_ID	Superfluous (see above).
	Student_ID	Superfluous (see above).
	Lesson_ID	Already in data model. (See above)
--id	Interaction_ID	Full name superfluous in data model. (Appears as interactions.id)
--objective_ids	Objective_ID	Plural for array names.
	Date	Already in data model. (See above)
--time	Time	
--type	Type_Interaction	
--responses		Need name for parent.
-- --description	Correct_Response	More accurate term. Element can represent incorrect answers as well as correct.
-- --value	Response_Value	
--weighting	Weighting	
--student_response	Student_Response	
--result	Result	
--latency	Latency	

Table of Lesson Evaluation Data (cont.)

Data Model Name	Field Title	Why Different
objectives	Objectives Status File	Same as objectives in lesson to LMS data.
	Course_ID	Superflous (see above).
	Student_ID	Superflous (see above).
	Lesson_ID	Already in data model. (See above)
	Date	Already in data model. (See above)
	Time	Passed with objectives information in Lesson to LMS data.
	Objective_ID	In lesson to LMS data under objectives.
	Score	In lesson to LMS data under objectives.
	Status	In lesson to LMS data under objectives.
--mastery_time	Mastery_Time	Must added to objectives in lesson to LMS data.
paths	Path File	
	Course_ID	Superflous (see above).
	Student_ID	Superflous (see above).
	Lesson_ID	Already in data model. (See above)
	Date	Already in data model. (See above)
--location_id	Element_Location	Element location is an ID.
--time	Time	
--status	Status	
--why_left	Why_Left	
--time_in_element	Time_in_Element	

Appendix C – IEEE Learning Object Metadata Draft 3

This appendix incorporates the entire IEEE LOM document and is available at ltsc.ieee.org.

IEEE Learning Technology Standards Committee (LTSC)

Learning Object Metadata

Working Draft Document 3 (approved 1999-11-27)

Contents

- [1. Background Information](#)
 - [1.1 Creation of this Document](#)
 - [1.2 Disclaimers](#)
 - [1.3 Submitting comments and corrections](#)
 - [1.4 Revisions](#)
 - [1.5 Acknowledgements](#)
- [2. Introduction](#)
 - [2.1 Scope](#)
 - [2.2 Purpose](#)
- [3. Overview of the Metadata Structure](#)
 - [3.1 Basic Metadata Structure](#)
 - [3.2 Data Elements](#)
 - [3.3 List Values](#)
 - [3.4 Vocabularies](#)
 - [3.5 Multiplicity](#)
 - [3.6 Minmax Values](#)
 - [3.7 Character Sets](#)
 - [3.8 Derived Schemes](#)
 - [3.9 Indexation](#)
 - [3.10 Representation](#)
- [4. Conformance](#)
- [5. Base Scheme](#)
- [6. LangStringType](#)
- [7. DateType](#)
- [8. VocabularyType](#)
- [9. References to other standards](#)
 - [9.1 Complete Dublin Core mapping](#)
 - [9.2 Miscellaneous](#)

1 Background Information

1.1 Creation of this document

This document has been created at the request of the IEEE LTSC P1484.12 Learning Object Metadata (LOM) Working Group. This draft document is to represent the best possible convergence of all the information collected to date, input from the working group and existing work in this area.

This version 3.8 incorporates further work on version 3.7, posted on October, 24, 1999 to the LOM mailing list, as documented in [section 1.4 Revisions](#).

1.2 Disclaimers

This document is a proposal for the next Working Draft of the IEEE 1484.12 Working Group. It is the latest in a series of versions that have been discussed at the meetings of the working group and on the mailing list, as documented in [section 1.4](#).

Copyright (C) 1999 IEEE. Do not use or claim conformance to this document.

1.3 Submitting comments and corrections

Please send all comments and corrections via Email to <wayne.hodgins@autodesk.com>.

1.4 Revisions

The previous release of the LOM specification was version 3.7, distributed through the LEARNING-OBJECTS mailing list on October, 24 1999.

Based on discussions after the release of version 3.7, a number of further modifications have been made to the specification. These are listed below:

- [section 3.2](#):
 - added note “these typically refer to other standards ([section 9](#)) or vocabularies ([section 3.4](#))” to explanation of domain (based on comment from Dan Rehak);
 - replaced “In that case, only the lowest level subelement shall have a value.” by “Elements with subelements shall not have values directly; only elements with no subelements shall have values directly. Elements with subelements shall have values indirectly only, through their subelements.” (based on comment from Dan Rehak);
- [section 3.4](#):
 - repeated example with index value as representation for entry (based on comment from Dan Rehak);
 - replaced “If the entry is an item from the vocabulary, then the detail is optional. If present, the detail provides additional specialization of the entry.” by “If the entry is an item from the vocabulary, then the detail can be null. If not null, then the detail provides additional specialization of the entry.” (based on comment from Dan Rehak);
- [section 3.7](#):
 - replaced “For instance, a derived scheme can define some elements as mandatory that are optional or conditional in the Base Scheme.” by “For instance, a derived scheme can restrict a vocabulary

for an element to a subset of the vocabulary defined herein.” (based on comment from Dan Rehak);

- [section 5](#) and [section 6](#):
 - o replaced “unordered list” by “multiple unordered instance” for elements with subelements (suggestion from Dan Rehak);
 - o replaced “ordered list” by “multiple ordered instance” for elements with subelements (suggestion from Dan Rehak);
 - o for elements with subelements, the terminology “single instance” was already in use;
 - o changed minmax value 4, 8, 16 or 32 for list items to 5, 10, 15 or 30 (suggestion from Dan Rehak);
- [section 5](#):
 - o changed wording (but not intended meaning!) of explanation throughout (suggestions from Phill Dodds, Frank Farance, Wayne Hodgins, Dan Rehak and Tom Wason);
- [section 6](#):
 - o [1.1:LangString.Language](#): changed minmax from 120 to 100 (suggestion from Dan Rehak);
- section 5, 6 and 7:
 - o Removed notes column: absorbed it into explanation and domain, as appropriate (based on suggestion from Scott Lewis).
- [section 8](#):
 - o [2:Detail](#): changed explanation to “Additional detail on the vocabulary entry.” (based on comment from Dan Rehak);
- throughout:
 - o many editorial changes, including appropriate use of ‘shall’ and ‘may’ (based on comments from Scott Lewis);

The LOM standard is being developed in IEEE 1484.12. Further information on past and current revisions may be found at <http://ltsc.ieee.org/wg12>.

1.5 Acknowledgements

The IEEE LTSC P1484.12 LOM working group wishes to thank Erik Duval, Tom Wason and Wayne Hodgins for their tireless efforts and commitment to developing a high quality solution and document.

This document has its origins in both the ARIADNE <http://ariadne.unil.ch> and IMS <http://www.imsproject.org> Projects, without which this document could not have been created.

This document also builds on metadata work done by the Dublin Core group <http://purl.org/dc>.

2 Introduction

Metadata is information about an object, be it physical or digital. As the number of objects continues to grow exponentially and especially as our needs for learning expand equally dramatically, the lack of information or

metadata about objects has produced a critical and fundamental constraint on our ability to discover, manage and use objects. To address this problem, the IEEE LTSC LOM working group has created a standard for “Learning Object Metadata”. This is the first documentation of this work and describes the *structured metadata* model, which has been developed by the working group.

2.1 Scope

This standard specifies the syntax and semantics of *learning objectmetadata*, defined as the *attributes required to fully and adequately describe a learning object*. A learning object is defined here as any entity, digital or non-digital, that can be used, re-used or referenced during technology-supported learning. Examples of technology-supported learning applications include computer-based training systems, interactive learning environments, intelligent computer-aided instruction systems, distance learning systems, web-based learning systems and collaborative learning environments.

Examples of learning objects include multimedia content, instructional content, instructional software and software tools that are referenced during technology supported learning. In a wider sense, learning objects could include learning objectives, persons, organizations, or events.

The LOM standards focuses on the minimal set of properties needed to allow learning objects to be managed, located, and evaluated. The standard accommodates the ability for locally extending the minimal set of properties.

Relevant properties of learning objects include type of object, author, owner, terms of distribution, and format. Where applicable, learning object metadata may include pedagogical properties, such as teaching or interaction style, grade level, mastery level and prerequisites. Any given learning object can have more than one description (i.e., LOM set or instance of the metadata scheme defined below).

The standard supports security, privacy, commerce, and evaluation, but only to the extent that metadata fields are provided for specifying descriptive tokens related to these areas; the standard does not concern itself with how these features are implemented. The LOM standard references existing open standards and existing work in related areas. For example, the data scheme below takes into account the efforts to standardize the description of content objects in general, as developed in the Dublin Core Metadata Initiative.

2.2 Purpose

1. To enable learners or instructors to search, evaluate, acquire, and use learning objects.
2. To enable sharing and exchanging of learning objects across any technology-supported learning system.
3. To enable developing learning objects in units that can be combined and decomposed in meaningful ways.
4. To enable computer agents to automatically and dynamically compose personalized lessons for an individual learner.
5. To complement the direct work on standards that are focused on enabling multiple Learning Objects to work together within an open, distributed, learning environment.
6. To enable documenting and recognizing the completion of existing or new learning and performance objectives associated with Learning Objects.
7. To enable a strong and growing economy for Learning Objects that supports and sustains all forms of distribution; non profit, not-for-profit and for profit.
8. To enable education, training and learning organizations, including government, public and private, to express educational content and performance standards in a standardized format that is independent of the content itself.

9. To provide researchers with standards that support collecting and sharing comparable data concerning the applicability and effectiveness of Learning Objects.
10. To define a standard that is simple yet extensible to multiple domains and jurisdictions so as to be most easily and broadly adopted and applied.
11. To support necessary security and authentication for the distribution and use of Learning Objects.

3 Overview of the Metadata Structure

3.1 Basic metadata structure

The structured approach to metadata definition implies that the actual descriptors (that together form a conventional, standardized description) of a learning resource - the learning object - are grouped into meaningful *categories*. The [Base Scheme](#) shall consist of nine such categories:

1. *General* shall group all context-independent features plus the semantic descriptors for the resource.
2. *Lifecycle* shall group the features linked to the lifecycle of the resource.
3. *Meta-metadata* shall group the features of the description itself (rather than those of the resource being described).
4. *Technical* shall group the technical features of the resource.
5. *Educational* shall group the educational and pedagogic features of the resource.
6. *Rights* shall group the features that deal with the conditions of use for the resource.
7. *Relation* shall group features of the resource that link it to other resources.
8. *Annotation* shall allow for comments on the educational use of the resource.
9. *Classification* shall group characteristics of the resource described by entries in classifications.

Taken all together, these categories form what is called here the [Base Scheme](#).

3.2 Data elements

Categories shall contain data elements. For each element, the base scheme shall define:

- *name*: shall describe how the meta-data element is called;
- *explanation*: shall define the definition of the element;
- *multiplicity*: shall define how many elements are allowed and whether their order is significant (see also [section 3.5](#));
- *domain*: shall define constraints on appropriate values for the data element - these typically refer to other standards ([section 9](#)) or vocabularies ([section 3.4](#));
- *type*: shall define whether the element's value is textual, a date or a reserved element;
- *note*: shall define additional explanations, guidelines for using the element, etc.;
- *example*

Both the multiplicity and type information can include minmax values (see [section 3.6](#)).

Some data elements contain subelements. Elements with subelements shall not have values directly; only elements with no subelements shall have values directly. Elements with subelements shall have values indirectly only, through their subelements. As an example, [1.3:General.CatalogEntry](#) has a value indirectly only, through [1.3.1:General.CatalogEntry.Catalogue](#) and [1.3.2:General.CatalogEntry.Entry](#).

3.3 List values

In some instances, a data element contains a *list of values*, rather than a single value. This list shall be one of:

- *ordered*: shall specify that the order of the values in the list is important. For example, in a list of authors of a publication, the first author is often considered the more important one.
- *unordered*: shall specify that the order of the values bears no meaning. For example, if the description of a simulation includes three short texts that describe the intended educational use in three different languages (for instance French, German and Italian), then the order of these texts not significant and they may appear in any order without loss of information.

A list of values shall contain at least one element. Implementations may use a list of zero length for internal operations, but an element with a zero length list shall not be distinguishable from an element with no value. Where a value is required, a zero-length list shall not be valid as a final value.

If an element with subelements contains a list of values, then each of these values shall be a tuple of subelements. For example, the base scheme defines that the element [1.3:General.CatalogEntry](#) contains an unordered list of values for the subelements [1.3.1:General.CatalogEntry.Catalogue](#) and [1.3.2:General.CatalogEntry.Entry](#). In other words, the Base Scheme defines that the value of the element [1.3:General.CatalogEntry](#) is an unordered list of ([1.3.1:General.CatalogEntry.Catalogue](#),[1.3.2:General.CatalogEntry.Entry](#)) elements.

3.4 Vocabularies

For some data elements, *vocabularies* are defined. A vocabulary shall be a list of appropriate values. Vocabularies shall be one of:

- *restricted*: shall specify that only the values from the list specified in this document are acceptable.
- *best practice*: A list of suggested best-practice values is provided that should be used, but other values may be used.

For data elements with associated vocabularies, the value shall be represented as a two element tuple (entry, detail). The entry shall be an item from the vocabulary. For data elements with an associated best-practice vocabulary, the entry may also be “User_defined”, or “See_classification”.

- If the entry is an item from the vocabulary, then the detail may be null. If not null, then the detail shall provide additional specialization of the entry. In this way, the detail shall further refine the vocabulary term with an additional term provided by the user.
- If the entry equals “User_defined”, then the detail element shall belong to a vocabulary not defined herein.
- If the entry equals “See_classification”, then the detail element shall belong to a vocabulary identified by an instance of the [9:Classification](#) category, whose value for [9.1:Classification.Purpose](#) shall equal the name of the element with the best practice vocabulary.

As an illustration, we give examples of the different cases for the element [5.2:Educational.LearningResourceType](#):

- The simplest case is just an item from the vocabulary, for instance “Questionnaire”. This would be represented as (“Questionnaire”,null)
- A somewhat more complex case is an item from the vocabulary, with an additional detail element, as in (“Questionnaire”,“Multiple Choice Questionnaire”).

- If the user wants to include a value that is not part of the list of [5.2:Educational.LearningResourceType](#), then the most simple case relies on the “User_defined” value for the entry, as in (“User_defined”, “MotivatingExample”).
- If the user wants to include a value from an existing classification, then he or she can do so through the “See_classification” value for the entry, as in (“See_classification”, “xxx”). In this case, there must be a value for [9:Classification](#), whose [9.1:Classification.Purpose](#) equals “LearningResourceType”. The value of [9.2.1:Classification.TaxonPath.Source](#) will then define the classification that “xxx” comes from.

A vocabulary provided by this specification is a list of “enumerated values”. The (entry) value shall be represented as an index value, as listed in the appropriate vocabulary. The (entry) value “User_defined” shall be represented as 1 and “See_classification” shall be represented as 2. Numbers of 3 and higher refer to specific vocabulary items as defined in the Base Scheme ([Section 5](#)). See also [section 8](#). The (detail) value of the tuple shall be represented as a [LangString](#).

So, the example above would be represented as:

- (“Questionnaire”, null): (5, null)
- (“Questionnaire”, “Multiple Choice Questionnaire”): (5, “Multiple Choice Questionnaire”)
- (“User_defined”, “MotivatingExample”): (1, “MotivatingExample”)
- (“See_classification”, “xxx”): (2, “xxx”)

3.5 Minmax values

In the [base scheme](#), minmax values are defined for:

- *elements with a list value*: All applications shall support at least that number of entries for the list. In other words: an application may impose a maximum on the number of entries it supports for the list value of that element, but that maximum shall not be lower than the minmax value.
- *elements with type String or [LangStringType](#)*: All applications shall support at least that length for the String value (either directly or contained in the [LangStringType](#)) of that element. In other words: an application may impose a maximum on the number of characters it supports for the string value of that element, but that maximum shall not be lower than the minmax value for the type of the element.

3.6 Character sets

This standard defines a conceptual structure for learning object metadata. It does not deal with representation issues, which will be dealt with in separate documents. Whatever decisions are made in documents that deal with representation, it is a firm expectation by the contributors of the LOM document that such decisions will be taken with a view to support multiple languages. This will have important repercussions with respect to the character sets to be supported.

3.7 Derived schemes

The metadata structure defined in this document is called the [Base Scheme](#). From the [Base Scheme](#), other schemes may be derived. Derived schemes shall *inherit* the structure from which they are derived. A derived scheme may add additional categories and data elements, but only to describe characteristics not taken into account in the Base Scheme. A common Base Scheme provides a high degree of interoperability and similarity among different derived schemes.

It is important to note that the properties described by optional data elements shall be described through these optional elements only and no overlapping data elements shall be introduced.

A particular derived scheme may be *more* restrictive. For instance, a derived scheme can restrict a vocabulary for an element to a subset of the vocabulary defined herein. But a derived scheme shall not be *less* restrictive.

3.8 Indexation

One should be aware of the fact that not all values for data elements need to be specified manually by each individual indexer or searcher. In many cases, the values could come from automated processes or templates that specify what is common for a number of objects. This implies that a user describing objects, or someone searching for appropriate material, would only be confronted with a subset of the elements in the Base Scheme.

3.9 Representation

For each of the data elements, the specification includes the data type from which it derives its values, such as [LangStringType](#) or [DateType](#), and so forth. These will be defined separately, and will be implemented in a particular way in a particular system. To maximize interoperability, future work may define a common representation for these data types. In the absence of such a common representation, an exchange format, such as XML, would allow systems with different representations to achieve interoperability through a conversion process.

4 Conformance

A metadata *instance* shall conform to this standard if it satisfies the following four requirements:

1. The metadata instance shall contain one or more LOM element(s).
2. All LOM elements in the metadata instance shall describe characteristics as defined by the LOM specification.
(For example, the user shall not abuse the title element to describe the fonts used in the document.)
3. Values for LOM elements in the metadata instance shall be structured as defined by the LOM specification and this structural information shall be carried within the instance.
(This means that the grouping in categories and subelements must be maintained. But it does not mean that representations cannot define mappings of this structure as they see fit. More specifically, an XML representation can use the lang attribute to represent the Language element of a [LangStringType](#) value.)
or
Bindings must carry equivalent information about the metadata so that conversions between bindings do not induce loss of information as defined within this standard.
4. If the instance contains extensions to the LOM structure, then extension elements shall not replace elements in the LOM structure.

5 BaseScheme

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
1	General	This category groups the general information that describes this resource as a whole. A globally unique label that identifies this resource.	single instance	-	-	-
1.1	Identifier	This is and shall not be used, because there is no specified method for the creation of a globally unique identifier. Name given to this resource.	single value	-	Reserved	-
1.2	Title	This element may be an already existing one or it may be created by the indexer ad hoc. This element shall correspond with the Dublin Core element DC.Title.	single value	-	LangStringType (1000 char)	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
This sub-category defines an entry within a catalogue (i.e., a listing identification system) assigned to this resource. This sub-category is intended to describe this resource according to some known cataloging system so that it may be externally searched for and located according to the methodology of the specified system.						
1.3	CatalogEntry	This sub-category may be used as a functional replacement for the currently reserved element 1.1:General.Identifier, as that is currently reserved. In this way, it shall be used to store the Dublin Core element DC.Identifier. One of the catalog entries can be generated automatically by the tool.	multiple unordered instance (10 items)	-	-	-
1.3.1	Catalogue	The name of the catalogue (i.e., listing identification system).	single value	ISO 10646-1	String (1000 char)	ISBN, ARIADNE
1.3.2	Entry	Actual string value of the entry within the catalogue (i.e., listing identification system).	single value	-	LangStringType (1000 char)	2-7342-0318, LEAO875
The primary human language used within this resource to communicate to the intended user.						
1.4	Language	An indexation tool may provide a useful default.	unordered list (10 items)	LanguageID = Langcode ('-' Subcode)*, with Langcode a two-letter language code as defined by ISO 639 and Subcode a country code from ISO 3166.	String (100 char)	"en", "en-GB", "de", "fr-CA", "it"
		This element shall correspond with the Dublin Core element DC.Language.		- character repertoire: ISO 646 - The approach adopted is compatible with that of the xml:lang attribute and is defined by RFC1766. ISO 639 deals with 'ancient' languages, like		

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
1.5	Description	A textual description of the content of this resource.	unordered list (10 items)	Greek and Latin.	LangStringType (2000 char)	-
		This element shall correspond with the Dublin Core element DC.Description.				
		Keywords or phrases describing this resource.				
1.6	Keywords	This element should not be used for characteristics that can be described by other elements. The span or extent of such things as time, culture, geography or region that applies to this resource.	unordered list (10 items)		LangStringType (1000 char)	-
1.7	Coverage	This element shall correspond with the Dublin Core element DC.Coverage.	unordered list (10 items)		LangStringType (1000 char)	Circa, 16th century France
1.8	Structure	Underlying organizational structure of this resource.	single value	- restricted vocabulary: 3=Collection 4=Mixed 5=Linear 6=Hierarchical 7=Networked 8=Branched 9=Parceled 10=Atomic ISO 646	VocabularyType	-
1.9	Aggregation Level	The functional granularity of this resource.	single value	0..3 ISO 646 - Level 0 shall be defined as	String (1 char)	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
				the smallest level of aggregation (e.g., raw media data or fragments).		
				- Level 1 shall be defined as a collection of atoms (e.g., an HTML document with some embedded pictures or a lesson). - Level 2 shall be defined as a collection of level 1 resources (e.g., a “web” of HTML documents, with an index page that links the pages together or a unit). - Finally, level 3 shall be defined as the largest level of granularity (e.g., a course).		
2	LifeCycle	This category describes the history and current state of this resource and those who have affected this resource during its evolution.	single instance -		-	-
2.1	Version	The edition of this resource.	single value -		LangStringType (50 3-0, 1.2.alpha, char) voorlopige versie	
2.2	Status	The state or condition of this resource.	single value	- restricted vocabulary: 3=Draft 4=Final 5=Revised 6=Unavailable ISO 646	VocabularyType	-
2.3	Contribute	This sub-category describes those people or organizations that have affected the state of this resource during its evolution (includes creation, edits and	multiple unordered instance (30 items)		-	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
		publication).				
		This sub-category is different from 3.3:MetaMetaData.Contribute .				
		Kind of contribution.				
2.3.1	Role	This element should include exactly one instance of Author.	single value	best practice list: 3=Author 4=Publisher 5=Unknown 6=Initiator 7=Terminator 8=Validator 9=Editor 10=Graphical Designer 11=Technical Implementer 12=Content Provider 13=Technical Validator 14=Educational Validator 15=Script Writer 16=Instructional Designer	VocabularyType	-
		The identification of and information about the people or organizations contributing to this resource, most relevant first.				
		If 2.3.1:LifeCycle.Contribute.Role equals Author, then the entity should be a person and this element shall correspond with the Dublin Core element DC.Creator.				
		If 2.3.1:LifeCycle.Contribute.Role equals Publisher, then the entity should be an organization and this element shall correspond with the Dublin Core element DC.Publisher.				
2.3.2	Entity	If 2.3.1:LifeCycle.Contribute.Role is not equal to Author or Publisher, then this element shall correspond with the Dublin Core element DC.Contributor.	ordered list (10 items)	vCard	String (1000 chars)	-
		If the entity is an organization, then it should be a university department, company, agency, institute, and so forth under whose				

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
2.3.3	Date	responsibility the contribution was made. The date of the contribution.	single value	-	DateType	-
3	MetaMetaData	This category describes the specific information about this metadata record itself (rather than the resource that this record describes). This category describes such things as who created this metadata record, how, when and with what references. This is <i>not</i> the information that describes the resource itself. A globally unique label that identifies this metadata record.	single instance	-	-	-
3.1	Identifier	This is and shall not be used, as there is no specified method for the creation of a globally unique identifier. This sub-category defines an entry within a catalogue (i.e., listing identification system), given to the metadata instance.	single value	-	Reserved	-
3.2	Catalog Entry	This category is intended to describe this metadata instance according to some known cataloging system so that it may be externally searched for and located according to that system. This element may be used as a functional	multiple unordered instance (10 items)	-	-	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
		replacement for the currently reserved element 3.1:MetaMetaData.Identifier .				
		One of the catalog entries may be generated automatically by the tool.				
3.2.1	Catalogue	The name of the catalogue (i.e., listing identification system). Generally system generated.	single value	ISO 10646-1	String (1000 char)	Ariadne
3.2.2	Entry	Actual string value of the entry in the catalogue. This element is usually generated by the system.	single value	-	LangStringType (1000 char)	KUL532
3.3	Contribute	This sub-category describes those people or organizations that have affected the state of this metadata instance during its evolution (includes creator and validator). This element is different from 2.3:Lifecycle.Contribute .	multiple ordered instance (10 items)	-	-	-
3.3.1	Role	Kind of contribution. Exactly one instance of creator should exist.	single value	open vocabulary with best practice list: 3=Creator 4=Validator	VocabularyType	-
3.3.2	Entity	The identification of and information about the people or organizations contributing to this metadata instance, most relevant first.	ordered list (10 items)	vCard	String (1000 char)	-
3.3.3	Date	The date of the contribution. The name and version of the authoritative specification used to create this metadata instance.	single value	-	DateType	-
3.4	Metadata Scheme	This element may be user selectable or system generated. If multiple values are provided, then the metadata instance shall conform to multiple metadata schemes.	unordered list (10 items)	ISO 646	String (30 char)	LOM-1.0
3.5	Language	Language of this metadata instance. This is the default language for all LangString values in this metadata instance.	single value	see LangStringType.Language	String (100 char)	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
4	Technical	This category describes the technical requirements and characteristics of this resource.	single instance -		-	-
		Technical data type of this resource.				
4.1	Format	This element shall be used to identify the software needed to access the resource. This element shall correspond with the Dublin Core element DC.Format. The size of the digital resource in bytes. Only the digits '0'..'9' should be used; the unit is bytes, not MBytes, GB, and so forth. This element shall refer to the actual size of this resource, and not to the size of a compressed version of this resource. A string that is used to access this resource. It may be a location (e.g., Universal Resource Locator), or a method that resolves to a location (e.g., Universal Resource Identifier). Preferable Location first.	unordered list (10 item)	MIME type or 'non-digital'	LangStringType (500 char)	video/mpeg, application/x-toolbook, text/html
4.2	Size	This element shall refer to the actual size of this resource, and not to the size of a compressed version of this resource.	single value	ISO 646 , but only the digits '0'..'9' String (30 char)	-	-
4.3	Location	This is where the learning resource described by this metadata instance is physically located. This sub-category describes the technical capabilities required in order to use this resource. If there are multiple requirements, then all are required (i.e., the logical connector is AND). The technology required to use this resource, i.e. hardware, software, network, and so forth. Name of the required technology to use this resource. The value for this element may be derived from 4.1:Technical.Format automatically,	ordered list (10 items)	ISO 10646-1	String (1000 char)	http://host/id
4.4	Requirements		multiple unordered instance (10 items)	-	-	-
4.4.1	Type		single value	open vocabulary with best practice: 3=Operating System 4=Browser if Type='Operating System', then best practice list: 3=PC-DOS 4=MS-Windows 5=MacOS	VocabularyType	-
4.4.2	Name		single value		VocabularyType	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
		e.g., “video/mpeg” implies “Multi-OS” .		6=Unix 7=Multi-OS 8=Other 9=None if Type=‘Browser’ then best practice list: 10=Any 11=Netscape Communicator 12=Microsoft Internet Explorer 13=Opera if other type then open vocabulary		
4.4.3	Minimum Version	Lowest possible version of the required technology to use this resource.	single value	ISO 646	String (30 char)	-
4.4.4	Maximum Version	Highest version of the technology known to support the use of this resource.	single value	ISO 646	String (30 char)	-
4.5	Installation Remarks	Description on how to install this resource.	single value	-	LangStringType (1000 char)	-
4.6	Other Platform Requirements	Information about other software and hardware requirements. Time a continuous resource takes when played at intended speed.	single value	-	LangStringType (1000 char)	sound card ..., runtime ...
4.7	Duration	This element is especially useful for sounds, movies or animations.	single value	-	DateType	PT1H30M, PT1M45S
5	Educational	This category describes the key educational or pedagogic characteristics of this resource. This is the pedagogical information essential to those involved in achieving a quality learning experience. The audience for this metadata includes teachers, managers, authors and learners.	single instance -		-	-
5.1	Interactivity Type	The flow of interaction between this resource and the intended user. In an <i>expositive</i> resource, the information	single value	restricted vocabulary: 3=Active 4=Expositive 5=Mixed	VocabularyType	Expositive documents include essays, video clips, all kinds of graphical

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
		flows mainly from this resource to the learner. Expositive documents are typically used for learning- by- reading. In an <i>active</i> resource, information also flows from the learner to this resource. Active documents are typically used for learning- by- doing. Activating links to navigate in hypertext documents is not considered as an information flow. Thus, hypertext documents are expositive.		6=Undefined		material and hypertext documents. Active documents include simulations, questionnaires and exercises.
5.2	Learning Resource Type	Specific kind of resource, most dominant kind first. This element shall correspond with the Dublin Core element 'Resource Type'. The vocabulary is adapted for the specific purpose of <i>learning</i> objects.	ordered list (10 items)	open vocabulary with best practice: 3=Exercise 4=Simulation 5=Questionnaire 6=Diagram 7=Figure 8=Graph 9=Index 10=Slide 11=Table 12=Narrative Text 13=Exam 14=Experiment 15=ProblemStatement 16=SelfAssesment	VocabularyType	-
5.3	Interactivity Level	This element shall define the degree of interactivity between the end user and this resource, with 0 defined as "Very Low", 1 defined as "Low", 2 defined as "Medium", 3 defined as "High", and 4 defined as "Very High". This elements shall define a subjective measure of this resource's usefulness as compared to its size or duration, with 0 defined as "Very Low", 1 defined as "Low",	single value	{0, 1, 2, 3, 4} ISO 646	String (1 char)	-
5.4	Semantic Density		single value	{0, 1, 2, 3, 4} ISO 646	String (1 char)	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
5.5	Intended end user role	2 defined as “Medium”, 3 defined as “High”, and 4 defined as “Very High”. Principal user(s) for which this resource was designed, most dominant first.	ordered list (4 items)	restricted vocabulary: 3=Teacher 4=Author 5=Learner 6=Manager	VocabularyType	-
		A learner works with a resource in order to learn something. An author creates or publishes a resource. A learner works with a resource in order to learn something. A manager manages the delivery of this resource, e.g., a university or college. The document for a manager is typically a curriculum.		Open vocabulary with best practice: 3=Primary Education 4=Secondary Education 5=Higher Education 6=University First Cycle 7=University Second Cycle 8=University Postgrade 9=Technical School First Cycle 10=Technical School Second Cycle 11=Professional Formation 12=Continuous Formation 13=Vocational Training 14=Other		
5.6	Context	The principal environment within which the learning and use of this resource is intended to take place.	unordered list (10 items)		VocabularyType	-
5.7	Typical Age Range	Age of the typical intended user.	unordered list (5 items)		LangStringType (1000 chars)	7-9, 0-5, 15, 18-, suitable for children over 7, adults only
		This element shall refer to developmental age, if that would be different from chronological age. The age of the learner is important for finding resources, especially for school age learners and their teachers. When applicable, the string should be formatted as minAge-maxAge or minAge-. (This is a compromise between adding three subfields (minAge, maxAge and description) and having just a free text field.)				

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
Various reading age schemes, IQ's or developmental age measures should be represented through the 9:Classification category This element defines how hard it is to work through this resource for the typical target audience, with 0 defined as "Very Easy", 1 defined as "Easy", 2 defined as "Medium", 3 defined as "Difficult", and 4 defined as "Very Difficult".						
5.8	Difficulty		single value	{0, 1, 2, 3, 4} ISO 646	String (1 char)	-
5.9	Typical Learning Time	Approximate or typical time it takes to work with this resource.	single value		DateType	PT1H30M, PT1M45S
5.10	Description	Comments on how this resource is to be used.	single value		LangStringType (1000 char)	Teacher guidelines that come with a textbook.
- LanguageID = Langcode('Subcode)*, with Langcode a two-letter language code as defined by ISO639 and Subcode a country code from ISO 3166. ISO 646 - This approach is compatible with that of the xml:lang attribute and is defined by RFC 1766. ISO 639 deals with 'ancient' languages, like Greek and Latin. - The language code should be given in lower case and the country code (if any) in upper case. However, the values are case insensitive.						
5.11	Language	The human language used by the typical intended user of this resource.	unordered list (10 items)		String (100 char)	"en", "en-GB", "de", "fr-CA", "it"

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
----	------	-------------	--------------	--------	------	---------

6	Rights	This category describes the intellectual property rights and conditions of use for this resource.	single instance -	-	-	•
		The intent is to reuse results of ongoing work in the Intellectual Property Right and e-commerce communities. This category currently provides the absolute minimum level of detail only.				
		Whether use of this resource requires payment.				
		Whether copyright or other restrictions apply to the use of this resource.				
6.1	Cost	Comments on the conditions of use of this resource.	single value	restricted vocabulary: 3=yes 4=no	VocabularyType	-
6.2	Copyright and Other Restrictions	Whether copyright or other restrictions apply to the use of this resource.	single instance	restricted vocabulary: 3=yes 4=no	VocabularyType	-
6.3	Description	Comments on the conditions of use of this resource.	single value	-	LangStringType (1000 char)	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
7	Relation	This category defines the relationship between this resource and other targeted resources, if any. To define multiple relationships there may be unordered multiple instances of this category. If there is instance (30 more than one target resource, then each target is covered by a new relationship instance.	multiple		-	-
7.1	Kind	Nature of the relationship between this resource and the target resource, identified by 7.2:Relation.Resource. This element shall correspond with the Dublin Core element DC.Relation.	single value	best practice list from Dublin Core: 3=IsPartOf 4=HasPart 5=IsVersionOf 6=HasVersion 7=IsFormatOf 8=HasFormat 9=References 10=IsReferencedBy 11=IsBasedOn 12=IsBasisFor 13=Requires 14=IsRequiredBy	VocabularyType	-
7.2	Resource	The target resource that this relationship references.	single instance	-	-	-
7.2.1	Identifier	Unique Identifier of the target resource. This is and shall not be used.	single value	-	Reserved	-
7.2.2	Description	Description of the target resource.	single value	-	LangStringType (1000 char)	-
8	Annotation	This category provides comments on the educational use of this resource, who created this annotation and when. When multiple annotations are needed, multiple instances of this category may be used.	multiple unordered instance (30 items)	-	-	-
8.1	Person	The person who created this annotation.	single value	vCard	String (1000 char)	-
8.2	Date	Date that this annotation was created.	single value	-	DateType	-
8.3	Description	The content of this annotation.	single value	-	LangStringType (1000 char)	-
9	Classification	This category describes where this resource is placed within a particular classification	multiple unordered	-	-	-

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
		system. To define multiple classifications, there may be multiple instances of this category. If 9.1:Classification Purpose equals Discipline, then this category shall correspond with the Dublin Core element DC.Subject.	instance (10 items)			
9.1	Purpose	The purpose of classifying this resource.	single value	open vocabulary with best practice: 3=Discipline 4=Idea 5=Prerequisite 6=Educational Objective 7=Accessibility Restrictions 8=Educational Level 9=Skill Level 10=Security Level	VocabularyType	-
9.2	TaxonPath	This sub-category describes a taxonomic path in a specific classification system. Each succeeding level is a refinement in the definition of the higher level. There may be different paths, in the same or different classifications, that describe the same characteristic. A taxonomy is a hierarchy of terms or phrases that are taxons. The name of the classification system. This element may use any recognized “official” taxonomy, any user-defined taxonomy. An indexation or query tool may provide the top-level entries of a well-established classification (LOC, UDC, DDC, etc.). This sub-category describes a particular term within a hierarchical classification system or taxonomy. A taxon is a node that has a defined label or term. A taxon may also have an alphanumeric designation or identifier for	multiple unordered instance (15 items)		-	-
9.2.1	Source		single value	ISO 10646-1	String (1000 char)	ACM, MESH, ARIADNE
9.2.2	Taxon		multiple ordered instance (15 items)		-	Physics/ Acoustics/ Instruments/ Stethoscope Medicine/ Diagnostics/

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
		standardized reference. Either or both the label and the entry may be used to designate a particular taxon.				Instruments/ Stethoscope
		An ordered list of Taxons creates a taxonomic path, i.e. “taxonomic stairway”: this is a path from a more general to more specific entry in a classification.				
		A TaxonPath shall have a depth from 1 to 9. Normal values should be defined as values between 2 and 4.				
9.2.2.1	Id	The identifier of the taxon, such as a number or letter combination provided by the source of the taxonomy.	single value	ISO 10646-1	StringType (100 char)	320 4.3.2 BF180
9.2.2.2	Entry	The textual label of the taxon.	single value	-	LangStringType (500 char)	Medical Sciences
9.3	Description	This is the description of the resource relative to the stated 9.1:Classification.Purpose of this specific classification, such as discipline, idea, skill level, educational objective, etc.. These are the keywords and phrases descriptive of the resource relative to the stated 9.1:Classification.Purpose of this specific classification, such as accessibility, security level, etc., most relevant first.	single value	-	LangStringType (2000 char)	A medical instrument for listening called a stethoscope.
9.4	Keywords		ordered list (10 items)	-	LangStringType (1000 char)	-

6 LangStringType

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
1	LangString	String in one or more human languages.	multiple unordered instance (10 items) -		-	-
1.1	Language	Human language in which the string is expressed. Indexation tool should provide useful default.	single value	• LanguageID = Langcode(‘-’Subcode)*, with Langcode a two-letter language code as defined by ISO 639 and Subcode a country code from ISO 3166. • ISO 646 • This approach is compatible with that of the xml:lang attribute and is defined by RFC 1766. • ISO 639 deals with ‘ancient’ languages, like Greek and Latin. • The language code should be given in lower case and the country code (if any) in upper case. However, the values are case insensitive. • If no Language is specified, then LangString.String should be interpreted as a string in 3.5:MetaMetaData.Language.	String (100 char)	“en”, “en-GB”, “de”, “fr-CA”, “it”
1.2	String	Actual string value. A string shall contain at least one letter. Implementations may use a string of zero length for internal operations, but an element with a zero length string shall not be distinguishable from an element with no value. Where a value is required, a zero length string shall not be valid as a final value.	single value	ISO 10646-1	String	-

7 DateType

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
Date expressed as per ISO 8601 standard.						
1	DateTime	All occurrences of this type shall correspond with the Dublin Core element DC.Date.	single value	ISO 8601	String (200 char)	1999-06-11
2	Description	Description of the date.	single value	-	LangStringType (1000 char)	circa 1300 BC, Fall Semester 1999

8 VocabularyType

Nr	Name	Explanation	Multiplicity	Domain	Type	Example
1	Entry	“User_defined”, “See_classification”, or an entry from the vocabulary of the data element.	single value	ISO 646	- String (2 char) 1=User_defined 2=See_classification	-
2	Detail	Additional detail on the vocabulary entry. See section 3.4 .	single value	-	LangStringType (1000 char)	-

9 References to Other Standards

9.1 Complete Dublin Core Mapping

The Dublin Core defines 15 fields of meta data information. These (unqualified) fields map directly to data elements in the above structure.

DC.Identifier	1.3:General.CatalogEntry . 1.1:General.Identifier is currently a reserved term, as there is no specified method for the creation of a globally unique identifier.
DC.Title	1.2:General.Title
DC.Language	1.4:General.Language
DC.Description	1.5:General.Description
DC.Subject	1.6:General.Keywords or 9:Classification with 9.1:Classification.Purpose equals “Discipline” or “Idea”.
DC.Coverage	1.7:General.Coverage
DC.Type	5.2:Educational.LearningResourceType
DC.Date	2.3.3:LifeCycle.Contribute.Date when 2.3.1:LifeCycle.Contribute.Role has a value of “Publisher”.
DC.Creator	2.3.2:LifeCycle.Contribute.Entity when 2.3.1:LifeCycle.Contribute.Role has a value of “Author”.
DC.OtherContributor	2.3.2:LifeCycle.Contribute.Entity with the type of contribution specified in 2.3.1:LifeCycle.Contribute.Role .
DC.Publisher	2.3.2:LifeCycle.Contribute.Entity when 2.3.1:LifeCycle.Contribute.Role has a value of “Publisher”.
DC.Format	4.1:Technical.Format
DC.Rights	6:Rights
DC.Relation	7:Relation
DC.Source	7.2:Relation.Resource when the value of 7.1:Relation.Kind is “IsBasedOn”.

9.2 Miscellaneous

- Dublin Core: The Dublin Core is a metadata element set intended to facilitate discovery of electronic resources. <<http://purl.org/dc/>>
- ISO 639: This is an international standard for the representation of languages. Version 1 uses two-letter language codes, e.g. 'en' for English, 'fr' for French, 'nl' for Dutch, and so forth. These language codes are a basis for the IETF registry of language tags, as specified in RFC 1766: Tags for the identification of languages.
- ISO 646: This is an international standard that defines the ASCII character set.
- ISO 3166: This is an international standard for the representation of country names, e.g. 'BE' for Belgium, 'CA' for Canada, 'FR' for France, 'GB' for United Kingdom, 'US' for United States, etc. <<http://www.din.de/gremien/nas/nabd/iso3166ma/codlstp1.html>>
- ISO 8601: This is an international Standard that specifies numeric representations of date and time. The basic notation is YYYY-MM-DD where YYYY is the year in the usual Gregorian calendar, MM is the month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 31. <<http://www.cl.cam.ac.uk/~mgk25/iso-time.html>>
- ISO 10646-1: This is an international Standard that specifies a character set that relies on 32 bits, includes approximately 4 billion characters, of which the first 65536 are Unicode, the first 256 are ISO 8859-1, and the first 128 are ASCII.
- MIME type: Multipurpose Internet Mail Extensions extends the format of Internet mail to allow non-US-ASCII textual messages, non-textual messages, multipart message bodies, and non-US-ASCII information in message headers. <<http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html>>
- RFC 1766: This Internet standard defines a language tag, referring to [ISO 639](#) for the language, and to [ISO 3166](#) for the country code. <<http://ds.internic.net/rfc/rfc1766.txt>>
- vCard: <<http://www.imc.org/pdi/>>: This standard defines how contact details for people and organizations can be represented. Appendix D – IMS Learning Resource Metadata XML Binding Specification

The following specification is available from www.imsproject.org.

Appendix D – IMS Learning Resource Metadata XML Binding Specification

IMS Learning Resource Metadata XML Binding Specification

Version 1.0

Copyright © 1999 by EDUCAUSE

[About This Document](#)

[Table of Contents](#)

[**Introduction**](#)

[**XML Basics**](#)

[Elements](#)

[*Element Contents*](#)

[*Element Attributes*](#)

[*Element Names*](#)

[Document Type Definitions \(DTD\)](#)

[*Declaring Element Contents*](#)

[*Declaring Element Attributes*](#)

[Use of Attributes](#)

[Lists](#)

[*Ordered Lists*](#)

[*Unordered Lists*](#)

[Namespaces](#)

[**Special Handling Requirements for Meta-data Elements**](#)

[LangStringType](#)

[DateType](#)

[Language elements](#)

[TaxonPath elements](#)

[vCard elements](#)

[Keywords](#)

[**Extensibility**](#)

[**Using the vCard Specification**](#)

[**Sample Meta-data Record**](#)

[**Appendix**](#)

[*Additional resources*](#)

Introduction

This document describes the XML binding for the IMS Learning Resource Meta-data Information Model. The model is based on the IEEE Learning Technology Standards Committee (LTSC) Learning Object Meta-data base document, plus modifications approved by the IMS Technical Board and submitted to IEEE. For links to the related IEEE documents, please see <http://www.imsproject.org/metadata/mdinfo01.html>

XML Basics

The IEEE conceptual model for metadata definitions is a hierarchy. Hierarchical models are convenient for representing data consisting of many elements and subelements. XML is perfectly suited for representing hierarchical models such as the IEEE LOM Base Document. An XML document is a hierarchy comprised of **elements** that have **contents** and **attributes**.

Elements

An element is a component of a document that has been identified in a way a computer can understand. Each element has a **tag name**. When a tag name is shown as “<TAGNAME>“, with less-than and greater-than symbols before and after the tag name, it serves as the **start-tag** to mark the beginning of an element. When that same tag name has a forward slash “/” added, it serves as an **end-tag** such as “</TAGNAME>“. An element may have contents between its start and end-tags, and may have one or more **attributes**. When an XML element has a start and end-tag (also called an **opening** and **closing tag**) with a common name, it is considered to be “well-formed” XML. The contents of an element are placed between the start and end-tags as shown below:

<TAGNAME>contents</TAGNAME>

Element Contents

An element may contain other elements, Parsed Character Data (PCDATA), Character Data (CDATA), or a mixture of PCDATA and elements. The allowable contents of an element are its content model. PCDATA really means any character string that does not contain elements. PCDATA is what the bulk of elements will use between their start and end-tags. CDATA is different in that it is a method for adding any character data that should not be processed. For example you could add some JavaScript code instructions using a CDATA section. A CDATA section tells the parser not to look for any markup until after it locates the end of the CDATA section.

Element Attributes

An attribute provides additional information about an element. Attributes are a way of attaching characteristics or properties to the elements of a document. An element may have more than one attribute and are contained within the start tag of an element. Attributes are represented by an attribute name followed by an equal sign and the attribute value in quotation marks:

**<TITLE> <LANGSTRING lang="en-US">Sniffy the Virtual
Rat</LANGSTRING> </TITLE>**

In this example, the TITLE element contains another element, the LANGSTRING element. The LANGSTRING element has one attribute “lang”, with the value “en-US” and the contents of the element being the string “Sniffy the Virtual Rat”.

Element Names

Each element has a unique name, the tag name. XML is case-sensitive in its processing of tag names. The IMS Learning Resource Meta-data XML binding specification adheres to the following tag name rules:

- All tag names will conform to the rules for element naming as given within the [XML Version 1.0 specification](#).
- Names beginning in “xml” in any case or mix of cases are not permitted.
- The IMS binding will use only **upper case** tag names. All element names in the IMS XML binding are to be in **capitals**. This will allow uniform machine conversion to a single case should the need arise.
- Element names may not include words reserved by the XML specification. These include:

DOCTYPE

ELEMENT

ATTLIST

ENTITY

- Tag names defined by the IMS binding may not be redefined.

Document Type Definitions (DTD)

The *tag name*, *content model*, and *attributes* of elements are defined in a **Document Type Definition** (DTD) statement. This may exist as an external file or a block of text internal to an XML document. Internal DTDs are used to override elements defined in external DTD files, so an internal DTD should be used with care. The DTD defines the elements that may be used, and may define the contents of the elements.

This specification defines external DTDs with defined file names, specifically **IMS-MD01.dtd** and **IMSCOR01.dtd**. These file names represent the 1.0 version of the IMS Meta-data and the version 1.0 of the IMS CORE metadata respectively. Some XML editors may make use of a DTD to help guide the developer in creating the proper elements at the proper locations in an XML file. Other developers will make use of DTDs to validate their XML documents to ensure their document is consistent with all of the element names and locations defined in the DTD. An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it. Details of the construction of DTDs are outside the scope of this document, but links to the [XML Version 1.0](#) specification and the IMS-MD01.dtd are included in the Appendices section of this document.

Declaring Element Contents

The information specifying the order and usage of allowable contents for an element are its content model. The content model is declared in a Document Type Definition or DTD (see below). The declaration of the content model is of the general form:

<!ELEMENT TAGNAME (Content Model)>

The DATETIME element can again serve as an example of how an element is declared with its content model:

<!ELEMENT DATETIME (#PCDATA|EXTENSION)*>

The vertical bar character “|” indicates that the metadata author may choose between the elements. The asterisk “*” after the content model means that the #PCDATA element and the EXTENSION element may be mixed or optionally interspersed with subelements. This definition of the DATETIME element’s content model allows the following XML fragment to exist:

```
<DATETIME>1999-07-23
  <EXTENSION> <LANGSTRING lang="en">circa</LANGSTRING>
</EXTENSION>
</DATETIME>
```

Notice that the EXTENSION element is optional and was used in the example above. The [XML specification](#) provides more information about the details for creating and interpreting content models.

Declaring Element Attributes

An example of how the attributes for the element LANGSTRING are declared in a DTD is found below:

<!ELEMENT LANGSTRING (#PCDATA|EXTENSION)*>
<!ATTLIST LANGSTRING
lang CDATA #IMPLIED>

The first line declares that there is an element named “LANGSTRING” which is allowed to have PCDATA and EXTENSION elements as its contents. The second line begins with “!ATTLIST” to start an attribute list declaration for the LANGSTRING element. The word “lang” will serve as the attribute’s name. The allowable value for this attribute must be of type CDATA.

At the end of the example above is the term #IMPLIED. It is at this location in the attribute declaration, where a default value for an attribute may be specified. It is also possible to use the keyword #REQUIRED which would force a lang value to be supplied and there would be no default value. In the example above, the #IMPLIED designation means that the DTD designer wants to allow users to omit the value for the attribute without forcing a particular default value.

Use of Attributes

Within the IMS XML binding, the use of attributes is reserved for information about the structure of, and source of terms in, the metadata record. It is recommended that attributes not be used for information about the resource. This IMS XML binding specification uses only two element attributes (the “lang” attribute and the “type” attribute) in particular ways and for particular purposes.

lang:

This attribute specifies the human language of the contents of the element. It is only used as an attribute of the LANGSTRING element. The lang attribute may contain a two-character language code followed by a two-character country code. For example:

```
<OTHERPLATFORMREQUIREMENTS>
  <LANGSTRING lang="en-US">Will not run in
browser.</LANGSTRING>
</OTHERPLATFORMREQUIREMENTS>
```

The codes for languages and countries are enumerated in the [XML specification](#).

type:

This attribute specifies the type of string that may be used to identify the location of a learning resource as used in the Location element. The type attribute may be assigned the value of either “URI” or “TEXT”. These values indicate whether the string used will be a simple textual description of where a resource is located or whether the string represents a resource available on the Internet with a specific address such as a URL. For example:

```
<TECHNICAL>
  <FORMAT/>
  <SIZE>1032353</SIZE>
  <LOCATION
type="URI">http://www.brookscole.com</LOCATION>
</TECHNICAL>
```

The codes for languages and countries are enumerated in the [XML specification](#).

Lists

The metadata specification uses **listing** at multiple levels in the hierarchy. A list is a repetition of the contents of an element. In XML, this is accomplished by repeating the containing element:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RECORD [
  <!ELEMENT RECORD (GREETING*)>
  <!ELEMENT GREETING (#PCDATA)>
]>
<RECORD>
  <GREETING>Hello, world!</GREETING>
  <GREETING>How are you?</GREETING>
</RECORD>
```

In this example, the element “GREETING” is repeated. Thus GREETING is the

containing element for the **repeated contents** of “Hello, world!” and “How are you?” The notation for repetitions of an element in a content model follows the [XML specification](#). An asterisk (*) specifies that none or more repetitions of the element may be included in the XML instantiation. There are two main types of lists: **ordered** and **unordered**.

Ordered Lists

Repeating the listed element at its specific location in the XML structure creates an ordered list of contents. The order of the elements has significance as their placement in the XML file determines this. The following is an example of an XML fragment in which the EDUCATIONAL element contains an ordered list of LEARNINGRESOURCETYPE (learning resource type) elements:

```
<EDUCATIONAL>
  <LEARNINGRESOURCETYPE>
    <LANGSTRING lang="en">Simulation</LANGSTRING>
  </LEARNINGRESOURCETYPE>
  <LEARNINGRESOURCETYPE>
    <LANGSTRING lang="en">Assessment</LANGSTRING>
  </LEARNINGRESOURCETYPE>
</EDUCATIONAL>
```

Unordered Lists

Repeating the containing element at its specific location in the XML structure creates an unordered list of contents. The order of the repetitions has no significance. For example:

```
<GENERAL>
  <LANGUAGE>en_US</LANGUAGE>
  <LANGUAGE>fr_FR</LANGUAGE>
</GENERAL>
```

In this example, each new instance of a definition of a language requires that the LANGUAGE element be repeated. Whether an element list should be treated as ordered or unordered is specified by the IEEE Learning Object Meta-data (LOM) specification.

Namespaces

XML is designed to allow individuals to create their own element tag names. It soon became apparent that there could be problems if different DTDs were used in the same document and those DTDs had elements using the same name. The XML Namespace recommendation proposal specifies a way to ensure that names from different DTDs can be safely combined in a single document.

XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URL references. The XML binding document uses a default document IMS namespace of <http://www.imsproject.org/metadata/>. An example namespace declaration for the IMS is shown below.

Namespace declaration:

<RECORD xmlns="http://www.imsproject.org/metadata/">

The XML Namespace document provides more information about the flexible capabilities of namespaces. Namespaces use their own DTD documents for validation. In order for any documents that are based upon this XML binding to find an associated DTD such as "IMS-MD01.dtd", the IMS-MD01.dtd file and the actual metadata record must be placed in the same directory or the DTD must be found at the specified URL.

Always make sure to place the IMS-MD01.dtd and the metadata record in the same directory if you wish to validate the record locally.

Special Handling Requirements for Meta-data Elements

There are some common structures that are used more than once within the metadata. The use of these common structures may facilitate the creation of common data storage structures in implementations. These structures have the suffix of "TYPE". INTERACTIVITYTYPE is not a common structure, even though it ends in "TYPE". The types defined in the LOM and encoded in the XML are:

- LangStringType
- DateType

LangStringType

LangStringType denotes a string that is encoded for a specific language or other interpretable type. It is of the logical form:

**LangStringType
LangString
Language
String**

It is important to note how the logical form specified by the IEEE LOM appears to be different from the XML binding suggested in this document. In the suggested binding of this document, LangStringType is not an XML element. Rather LangString is an element with a "lang" attribute which is used to define the language of a string value:

<LANGSTRING lang="en">string value</LANGSTRING>

The lang attribute can contain both language and country codes as defined in the XML specification. Any element that contains a LANGSTRING element may contain multiple LANGSTRING elements with each one representing a different language

Each LANGSTRING within an element is considered to contain the same information, differing by language.

The XML 1.0 specification also allows the lang attribute to be assigned an arbitrary value that is agreed upon by parties in private use. These attributes are identified by the prefix

“x-” or “X-”. This practice is strongly discouraged for IMS metadata records.

DateType

DateType is a formatted date. There are two subelements representing the two different date formats used. Precise date and time information is formatted according to the ISO 8601 specification. Point in time and time duration information is captured in the DateTime element. More general date and time information is captured using the Description element. A DateType may contain values for both a DateTime and Description.

DateType has the logical structure of:

```
DateType
  DateTime
  Description
    LangStringType
      LangString
        Language
          String
```

It is important to note that, just as with the LangStringType, the logical form specified by the IEEE LOM for DateType appears to be different from the XML binding suggested in this document. In this binding, DateType is not an XML element. Rather Date is an element with two subelements: DateTime and Description.

The DateTime element makes use of the format dictated by the ISO8601 specification. Dates are captured using the CCYY-MM-DD form while Time elements are specified as hh:mm:ss. There is also the ability to specify Time Zone Determination information by adding “+hh:mm” to indicate differences in time zones. Both the date and time value are combined using the capital “T” character to separate them. Three examples of this are found below:

Use DATETIME for precise dates and times. Use DESCRIPTION for more general date time information.

```
<DATETIME>1999-07-26<DATETIME>
<DATETIME>1999-07-26T12:15:35<DATETIME>
<DATETIME>1999-07-26T12:15:35+01:00<DATETIME>
```

There is a version of the ISO8601 specification that is available through the W3C which also specifies how a date range is represented and how date and time duration is accurately represented. Readers may refer to the ISO8601 specification available at: <http://www.w3.org/TR/NOTE-datetime> for more detailed information on the usage of date and time values.

Language elements

Meta-data authors may specify a language that will be used as the default language for any LANGSTRING elements that are encountered. This is done by providing a value for the LANGSTRING element that is contained by the METAMETADATA element. Each

individual occurrence of LANGSTRING may override this default value by declaring a language and country code using the “lang” attribute.

The default language for a record can be specified in the MetaMetaData category. Use individual LANGSTRING elements to override the default language.

TaxonPath elements

In most cases, the value of using the TAXONPATH element lies in the ability to locate the source of the taxonomy. If the source for a TAXONPATH is not provided or doesn't map to an existing, logical source then the element should contain something useful regarding how to location information about the taxonomy.

Always try to provide a SOURCE element when using the TAXONPATH element.

vCard elements

There are at least two elements in the IEEE LOM that require contributing entity information; elements LifeCycle.Contribute.Entity and MetaMetaData.Contribute.Entity. Both specify the vCard specification as the source for representing these elements' data.

When using only IMS Core elements, the **formatted name** or “FN” element from the vCard specification should contain the name of the individual contributing to the learning resource of metametadata of the resource. If a company, rather than an individual, contributed to the resource or resource metametadata, the **organization** or “ORG” element from the vCard specification should be used. This is illustrated below:

```
<CENTITY>
  <VCARD>
    BEGIN:vCard
    FN:Lotta Data
    END:vCard
  </VCARD>
</CENTITY>
```

As far as most XML parsers are concerned, the information between the <VCARD> and </VCARD> tags is just a bunch of text. It is up to metadata implementers to individually determine how they will process vCard information. The vCard specification allows for a rich set of information to be captured as the example below illustrates. The reader is directed to the “Using the vCard Specification” portion of this document and the vCard specification itself for more details regarding its usage.

Keywords

The elements General.Keywords and Classification.Keywords are found in the IMS metadata set. It is expected that the keywords describing a learning resource are likely to be provided in multiple languages. To accommodate this, the IMS XML binding suggests that keywords and short, keyword phrases be represented as separate LANGSTRING elements rather than a comma-delimited text string as in the example below:

Use multiple LANGSTRING elements to represent keywords and keyword phrases.

```
<KEYWORDS>
  <LANGSTRING lang="en">operant
conditioning</LANGSTRING>
  <LANGSTRING lang="en">psychology</LANGSTRING>
  <LANGSTRING lang="en">simulation</LANGSTRING>
  <LANGSTRING lang="en">program</LANGSTRING>
  <LANGSTRING lang="en">shaping</LANGSTRING>
  <LANGSTRING lang="en">mouse</LANGSTRING>
  <LANGSTRING lang="en">learn by doing</LANGSTRING>
</KEYWORDS>
```

Extensibility

Some metadata providers will find the current element set defined in the IMS metadata as too restrictive to accomplish their metadata purposes. To ensure metadata extensibility, the IEEE LOM Base Document requires that there be no limit on potential extensions to the metadata as long as the integrity of the specified metadata is not impaired. An extension is the addition of information to an existing metadata XML structure. There are two general ways to extend IMS metadata:

1. One or more elements may be added to the structure using elements defined in the IEEE LOM Base Document; and
2. One or more elements may be added to the structure using elements that are not defined in the LOM specification.

These two types of extension are defined as:

1. Use of elements from the LOM: The LOM specification contains some elements with definitions that are not specific for any particular context. The context in which an element is placed provides specificity for its definition. These elements may be reused as long as the definition is not changed.
2. Use of elements not contained in the LOM specification: New elements may be introduced and used to extend the structure.

The XML binding does not inhibit either of these types of metadata extension. The XML binding defines the Extension element as the element used for indicating where a set of extension elements can be found in the metadata structure. In the IMSMD01.dtd file, the EXTENSION element exists in every element's content model allowing every element to contain one or more EXTENSION elements. The only element without an extension capability is IDENTIFIER, as it is a reserved word. The EXTENSION element's DTD declaration is:

Use the EXTENSION element to extend the metadata structure.

<!ELEMENT EXTENSION ANY>

An example of the inclusion of EXTENSION in the content model of element COVERAGE is:

```
<!ELEMENT COVERAGE (LANGSTRING*, EXTENSION?)>
```

The use of the EXTENSION element is illustrated as follows:

```
<COVERAGE>  
<LANGSTRING>1880-1900</LANGSTRING>  
<EXTENSION>  
  <ROLE>Date Range</ROLE>  
</EXTENSION>  
</COVERAGE>
```

The contents, but not a content model, of an extension must be declared in an internal or external DTD. Many extensions can be created through the use of existing elements. Care must be used with internal DTDs, as they override external DTD declarations. The contents of an extension must obey the attribute and content models of the elements employed. New elements that duplicate the definitions of existing elements should not be introduced.

Prefacing the EXTENSION element with an appropriate namespace may usefully reference descriptions of extensions. For example, a group such as the Advanced Distributed Learning (ADL) initiative may wish to add the “adl” prefix to an extension element to uniquely identify ADL extensions. An example of this is show below:

```
<LEARNINGCONTEXT>  
  <LANGSTRING lang="en">Military Training</LANGSTRING>  
    <EXTENSION adl:type="Topic">  
      <TITLE>Roman military tactics</TITLE>  
      <LANGSTRING lang="en">This example discusses how  
the Romans  
      defined many aspects of modern warfare.  
    </LANGSTRING>  
  </EXTENSION>  
</LEARNINGCONTEXT>
```

This serves to note the entire extension structure. Extensions should always be added at the lowest point (farthest from the root element) in the hierarchy possible, to the degree that the structure defines the meaning of the extension.

Using the vCard Specification

The IMS XML binding uses the vCard specification wherever the Entity element is defined. An Entity, as far as IMS metadata is concerned, represents a person or

organization. The IMS binding uses the clear text form of the vCard specification. The vCard specification defines the **clear text** form as a “Simplegram”. This is not intended as a complete description of the vCard coding; it is intended to provide some guidelines for simple cases. The vCard specification is located at <http://www.imc.org/pdi/>.

The vCard specification defines a set of **properties** that contain the information about an entity. The default character set encoding for the vCard is 7-bit US-ASCII. The default character set can be overridden for an individual property using the “CHARSET” property parameter. For example, the ISO 8859-8 or Latin/Hebrew character set used in an address is specified by:

ADR;CHARSET=ISO-8859-8:...

It is also possible to set the encoding for the entire record to another encoding. See the vCard specification for further instructions. The default language is “en-US”, which may similarly be overridden for a property using the “LANGUAGE” property parameter. Property names are case insensitive.

The general form of the Simplegram vCard encoding is:

BEGIN:VCARD
Items
END:VCARD

Items is a list of items separated by a any valid **line ending protocol**. For example, in 7-bit ASCII, the Carriage Return (CR) character (ASCII decimal 13), the Line Feed character (LF) (ASCII decimal 10), the Carriage Return character followed by a Line Feed character (CRLF), or the Property Delimiter are line ending protocols. Property parameter substrings are delimited by the Field Delimiter, specified by the Semi-Colon (;) character (ASCII decimal 59). Each item is of the general form:

name:value A;value B CRLF

An example of an item with no substrings is the **formatted name** property, **FN**. **FN** is the full formatted name of a person:

FN:Mr. James Q. Smith, Jr.

Some items may have multiple properties or **parameter substrings**. For example, a person’s name, **N**, can contain a Family Name, a Given Name, an Other Names, Prefix and a Suffix. The property value is a concatenation of the Family Name (first field), Given Name (second field), Additional Names (third field), Name Prefix (fourth field), and Name Suffix (fifth field) strings separated by the Field Delimiter “;”.

N:Smith;James;Q.;Mr.;Jr

An unused substring, if not at the end of the list of substrings, is represented with a semicolon only:

N:Smith;James;Q.;;Jr

Vcards may be organized to contain groups. The grouping of a comment property with a telephone property is shown in the following example:

```
A.TEL;Home:+1-213-555-1234  
A.NOTE:This is my vacation home.
```

Below are some commonly used vCard properties, with substrings. Separate lines should not be used for field substrings, but are used here for clarity:

Formatted Name:

FN:Text Value

Example:

```
FN:Dr. Thomas D. Wason, Sr.
```

Name:

```
N:Family (Sur) Name;  
First (Given) Name;  
Other Names;  
Prefix;  
Suffix CRLF
```

Example:

```
N:Wason;Thomas;D.;Dr.;Sr
```

Address:

The property value is a concatenation of the Post Office Address (first field), Extended Address (second field), Street (third field), Locality (fourth field, e.g., City), Region (fifth field, e.g., State), Postal (Zip) Code (six field), and Country (seventh field) strings:

```
ADR:P.O.Box;  
: Extended Address  
Street;  
Locality;  
Region;  
Postal Code;  
Country CRLF
```

Example:

```
ADR;;IMS Project;1421 Park Drive;Raleigh;North  
Carolina;27605-1727;United States of America
```

Address Delivery Label:

LABEL: Text

Example:

```
LABEL;QUOTED-PRINTABLE:IMS Project=0A=
1421 Park Drive=0A=
Raleigh, NC 27605-1727=0A=
```

Note the use of the escaped line feed (=0A=). The property parameters are preceded by a semicolon (;) after the property name. They are optional, defining the uses of the Delivery Label.

Organization:

The property value is a concatenation of the Organization Name (first field), Organizational Unit (second field) strings. Additional positional fields, if specified, contain additional Organizational Units:

```
ORG:Organization Name;
Organizational Unit[;
Organizational Unit] CRLF
```

Example:

```
ORG:IMS Project;Meta Data Team
```

Electronic Mail:

```
EMAIL; Electronic Mail Type:email
```

Example:

```
EMAIL;INTERNET:twason@imsproject.org
```

Telephone:

```
TEL:telephone number
```

Example:

```
TEL:+1 919.839.8187
```

All of these previously described properties are included in the following example:

```
BEGIN:VCARD
N:Wason;Thomas;D.;Dr.;Sr.
FN:Thomas D. Wason, Ph.D.
ORG:IMS Project;Meta Data Team
ADR;;IMS Project;1421 Park Drive;Raleigh;North
Carolina;27605-1727;United States of America
TEL:+1 919.839.8187
EMAIL;INTERNET:twason@imsproject.org
LABEL;QUOTED-PRINTABLE:IMS Project=0A=
1421 Park Drive=0A=
Raleigh, NC 27605-1727=0A=
USA
END:VCARD
```

Sample Meta-data Record

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE RECORD SYSTEM "http://www.imsproject.org/XML/IMS-MD01.dtd">
<!-- Reference to master DTD at IMS site. -->
<!--DOCTYPE RECORD SYSTEM "IMS-MD01.dtd" -->
<!-- Use this doc type declaration for references to a local dtd. -->
<RECORD xmlns="http://www.imsproject.org/metadata/">
  <!-- 1999-08-18, Thomas D. Wason: Sniffy01.xml. A complete meta-data set. A few empty fields.
  Located at http://www.imsproject.org/xml/Sniffy01.xml -->
  <!-- The full IEEE - IMS Learning Object Meta-data in XML. -->
  <METAMETADATA>
    <CATALOGENTRY>
      <CATALOGUE>IMS-Test</CATALOGUE>
      <ENTRY>
        <LANGSTRING>1999.000002</LANGSTRING>
      </ENTRY>
    </CATALOGENTRY>
    <CONTRIBUTE>
      <ROLE>
        <LANGSTRING lang="en">Author</LANGSTRING>
      </ROLE>
      <CENTITY>
        <!-- Simple vCard example -->
        <VCARD>
BEGIN:VCARD
N:Wason;Thomas;D.;Dr.;Sr.
FN:Thomas D. Wason, Ph.D.
ORG:IMS Project;Meta Data Team
ADR:;IMS Project;1421 Park Drive;Raleigh;North Carolina;27605-1727;United States of America
TEL:+1 919.839.8187
EMAIL;INTERNET:twason@imsproject.org
LABEL;QUOTED-PRINTABLE:IMS Project=0A=
1421 Park Drive=0A=
Raleigh, NC 27605-1727=0A=
USA
END:VCARD
</VCARD>
        </CENTITY>
        <DATE>
          <DATETIME>1999-08-05</DATETIME>
        </DATE>
      </CONTRIBUTE>
      <METADATAScheme>IEEELOM:1.0</METADATAScheme>
      <LANGUAGE>en-US</LANGUAGE>
      <!-- English as default meta-data language. -->
    </METAMETA-DATA>
    <GENERAL>
      <TITLE>
        <LANGSTRING lang="en-US">Sniffy The Virtual Rat</LANGSTRING>
        <LANGSTRING lang="fr">Sniffy Virtuel le Rat</LANGSTRING>
      </TITLE>
      <CATALOGENTRY>
        <CATALOGUE>ISBN</CATALOGUE>
      <ENTRY>
```

```

    <LANGSTRING>0-534-26702-5</LANGSTRING>
  </ENTRY>
</CATALOGENTRY>
<LANGUAGE>en-US</LANGUAGE>
<DESCRIPTION>
  <!--English description-->
  <LANGSTRING lang="en-US">A computer program that enables students to explore the principles of
shaping and partial reinforcement in operant conditioning, using a "virtual rat" named Sniffy. Each student
learns by doing-conditioning his or her own rat-and experiences many benefits of animal experimentation
but none of the drawbacks associated with using live animals.</LANGSTRING>
  <!--French Description -->
  <LANGSTRING lang="fr">Un programme machine qui permet &#224; des &#233;tudiants d'explorer
les principes de la formation et du renfort partiel dans l'op&#233;rateur conditionnant, utilisant " un rat
virtuel " a nomm&#233; Sniffy. Chaque &#233;tudiant apprend par le faire-conditioning ses propres rat-et
&#233;prouve beaucoup d'avantages l'exp&#233;rimentation animale de mais aucune des
inconv&#233;nients associ&#233;s &#224; utiliser les animaux vivants. </LANGSTRING>
</DESCRIPTION>
<KEYWORDS>
  <!--English Keywords, unordered list-->
  <LANGSTRING lang="en">operant conditioning</LANGSTRING>
  <LANGSTRING lang="en">psychology</LANGSTRING>
  <LANGSTRING lang="en">simulation</LANGSTRING>
  <LANGSTRING lang="en">program</LANGSTRING>
  <LANGSTRING lang="en">shaping</LANGSTRING>
  <LANGSTRING lang="en">rat</LANGSTRING>
  <LANGSTRING lang="en">learn by doing</LANGSTRING>
</KEYWORDS>
<KEYWORDS>
  <!--French keywords, unordered list-->
  <LANGSTRING lang="fr">traitement d'op&#233;rateur</LANGSTRING>
  <LANGSTRING lang="fr">psychologie </LANGSTRING>
  <LANGSTRING lang="fr">simulation</LANGSTRING>
  <LANGSTRING lang="fr">programme</LANGSTRING>
  <LANGSTRING lang="fr">formation</LANGSTRING>
  <LANGSTRING lang="fr">rat</LANGSTRING>
  <LANGSTRING lang="fr">apprenez en faisant</LANGSTRING>
</KEYWORDS>
<COVERAGE>
  <LANGSTRING/>
</COVERAGE>
<STRUCTURE>
  <LANGSTRING>Hierarchical</LANGSTRING>
</STRUCTURE>
<AGGREGATIONLEVEL>2</AGGREGATIONLEVEL>
</GENERAL>
<LIFECYCLE>
  <VERSION>
    <LANGSTRING>4.5</LANGSTRING>
  </VERSION>
  <STATUS>
    <LANGSTRING lang="en">Final</LANGSTRING>
  </STATUS>
  <!--Contains an unordered list of CONTRIBUTE-->
  <CONTRIBUTE>
    <ROLE>
      <LANGSTRING lang="en">Author</LANGSTRING>

```

```

        </ROLE>
        <!--Contains an ordered list of CENTITY-->
        <CENTITY>
        <VCARD>
BEGIN:vCard
FN:Lester Krames
END:vCard
</VCARD>
    </CENTITY>
    <CENTITY>
    <VCARD>
BEGIN:vCard
FN:Jeff Graham
END:vCard
</VCARD>
    </CENTITY>
    <CENTITY>
    <VCARD>
BEGIN:vCard
FN:Tom Alloway
END:vCard
</VCARD>
    </CENTITY>
    <DATE>
    <DATETIME>1995</DATETIME>
    </DATE>
    </CONTRIBUTE>
    <CONTRIBUTE>
    <ROLE>
    <LANGSTRING lang="en">Technical Implementer</LANGSTRING>
    </ROLE>
    <CENTITY>
    <VCARD>
BEGIN:vCard
FN:Greg Wilson
END:vCard
</VCARD>
    </CENTITY>
    </CONTRIBUTE>
    <CONTRIBUTE>
    <ROLE>
    <LANGSTRING lang="en">Publisher</LANGSTRING>
    </ROLE>
    <CENTITY>
    <VCARD>
BEGIN:vCard
ORG:Brooks/Cole publishing;International Thomson Publishing Company
END:vCard
</VCARD>
    </CENTITY>
    <DATE>
    <DATETIME>1995</DATETIME>
    </DATE>
    </CONTRIBUTE>
    </LIFECYCLE>
    <TECHNICAL>

```

```

<FORMAT/>
<SIZE>1032353</SIZE>
<LOCATION type="URI">http://www.brookscole.com</LOCATION>
<REQUIREMENTS>
  <TYPE>
    <LANGSTRING lang="en">Operating System</LANGSTRING>
  </TYPE>
  <NAME>
    <LANGSTRING lang="en">MS-DOS</LANGSTRING>
  </NAME>
  <MINIMUMVERSION>5.0</MINIMUMVERSION>
  <MAXIMUMVERSION/>
</REQUIREMENTS>
<INSTALLATIONREMARKS>
  <LANGSTRING lang="en">Load from diskette</LANGSTRING>
</INSTALLATIONREMARKS>
<OTHERPLATFORMREQUIREMENTS>
  <LANGSTRING lang="en">Will not run in browser.</LANGSTRING>
</OTHERPLATFORMREQUIREMENTS>
<DURATION>
  <DATETIME>0000-00-00T01:20</DATETIME>
</DURATION>
</TECHNICAL>
<EDUCATIONAL>
  <INTERACTIVITYTYPE>
    <LANGSTRING>Active</LANGSTRING>
  </INTERACTIVITYTYPE>
  <LEARNINGRESOURCETYPE>
    <LANGSTRING lang="en">Simulation</LANGSTRING>
  </LEARNINGRESOURCETYPE>
  <INTERACTIVITYLEVEL>3</INTERACTIVITYLEVEL>
  <SEMANTICDENSITY>2</SEMANTICDENSITY>
  <INTENDEDENDUSERROLE>
    <LANGSTRING lang="en">Learner</LANGSTRING>
  </INTENDEDENDUSERROLE>
  <!--May be an ordered list of intended user roles with the most relevant first.-->
  <LEARNINGCONTEXT>
    <LANGSTRING lang="en">Secondary Education</LANGSTRING>
  </LEARNINGCONTEXT>
  <TYPICALAGERANGE>
    <LANGSTRING>12-99</LANGSTRING>
  </TYPICALAGERANGE>
  <DIFFICULTY>2</DIFFICULTY>
  <TYPICALLEARNINGTIME>
    <DATETIME>0000-00-00T03:00</DATETIME>
  </TYPICALLEARNINGTIME>
  <DESCRIPTION>
    <LANGSTRING lang="en">Interactive teaching of the concepts of operant
conditioning.</LANGSTRING>
  </DESCRIPTION>
  <LANGUAGE>en_US</LANGUAGE>
</EDUCATIONAL>
<RIGHTS>
  <COST>
    <LANGSTRING lang="en">yes</LANGSTRING>
  </COST>

```

```

<COPYRIGHTOROTHERRESTRICTIONS>
  <LANGSTRING>yes</LANGSTRING>
</COPYRIGHTOROTHERRESTRICTIONS>
<DESCRIPTION>
  <LANGSTRING lang="en">Copyright 1995 Brooks Cole Publishing</LANGSTRING>
  <LANGSTRING lang="en">Contact publisher to purchase</LANGSTRING>
</DESCRIPTION>
</RIGHTS>
<RELATION>
  <KIND>
    <LANGSTRING lang="en">IsReferencedBy</LANGSTRING>
  </KIND>
  <RESOURCE>
    <!--IDENTIFIER reserved for later use. Do not use.-->
    <DESCRIPTION>
      <LANGSTRING lang="en">Companion book of the same name.</LANGSTRING>
    </DESCRIPTION>
  </RESOURCE>
</RELATION>
<ANNOTATION>
  <CENTITY>
    <VCARD>
BEGIN:VCARD
FN:Stuart Sutton
ORG:GEM;Syracuse University
END:VCARD</VCARD>
    </CENTITY>
    <DATE>
      <DATETIME>1999-06-10</DATETIME>
    </DATE>
    <DESCRIPTION>
      <LANGSTRING lang="en-US">A useful simulation for the student new to concepts of behavioral
science.</LANGSTRING>
    </DESCRIPTION>
  </ANNOTATION>
<CLASSIFICATION>
  <PURPOSE>
    <LANGSTRING lang="en">Discipline</LANGSTRING>
  </PURPOSE>
  <TAXONPATH>
    <SOURCE>Dewey</SOURCE>
    <!--Ordered list of Taxons-->
    <TAXON>
      <ID>300</ID>
      <ENTRY>
        <LANGSTRING lang="en">Social Sciences</LANGSTRING>
      </ENTRY>
    </TAXON>
    <TAXON>
      <ID>320</ID>
      <ENTRY>
        <LANGSTRING lang="en">Political Science</LANGSTRING>
      </ENTRY>
    </TAXON>
  </TAXONPATH>
</TAXONPATH>

```

```

<SOURCE>LCSH</SOURCE>
<!--Ordered list of Taxons-->
<TAXON>
  <ID>B</ID>
  <ENTRY>
    <LANGSTRING lang="en">Philosophy, Psychology, Religion</LANGSTRING>
  </ENTRY>
</TAXON>
<TAXON>
  <ID>F</ID>
  <ENTRY>
    <LANGSTRING lang="en">Psychology</LANGSTRING>
  </ENTRY>
</TAXON>
<TAXON>
  <ID>180</ID>
  <ENTRY>
    <LANGSTRING lang="en">Experimental Psychology</LANGSTRING>
  </ENTRY>
</TAXON>
</TAXONPATH>
<DESCRIPTION>
  <LANGSTRING lang="en-US">principles of operant conditioning</LANGSTRING>
</DESCRIPTION>
<KEYWORDS>
  <LANGSTRING lang="en">operant conditioning</LANGSTRING>
  <LANGSTRING lang="en">psychology</LANGSTRING>
</KEYWORDS>
</CLASSIFICATION>
</RECORD>

```

Appendix

Additional Resources

IMS XML Documents

IMS-MD01.dtd is located at: <http://www.imsproject.org/xml/IMS-MD01.dtd>

IMS-MD01.xml is located at: <http://www.imsproject.org/xml/IMS-MD01.xml>

IMS Meta-data Documents

The IMS Resource Meta-data Best Practice and Implementation Guide can be found at: <http://www.imsproject.org/metadata/mdbest01.html>

The IMS Learning Resource Meta-data Information Model document can be found at: <http://www.imsproject.org/metadata/mdinfo01.html>

ISO/IEC 10646

ISO (International Organization for Standardization). ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

Unicode

The Unicode Consortium. The Unicode Standard, Version 2.0. Reading, Mass.: Addison-Wesley Developers Press, 1996.

VCard

vCard Specification <http://www.imc.org/pdi/>

XML

XML Version 1.0 specification of the W3C: <http://www.w3.org/TR/1998/REC-xml-19980210>

XML Namespace Recommendation of W3C: <http://www.w3.org/TR/1999/REC-xml-names-19990114>

About This Document

Title	IMS Learning Resources Meta-data XML Binding Specification
Authors	Thomas D.Wason, Thor Anderson, Steve Griffin
Version	1.0
Version Date	August 20, 1999
Status	Final
Summary	This document describes the XML binding for the IMS Learning Resource Meta-data Information Model. The model is based on the IEEE Learning Technology Standards Committee (LTSC) Learning Object Meta-data base document, plus modifications approved by the IMS Technical Board and submitted to IEEE. For links to the related IEEE documents, please see http://www.imsproject.org/metadata/mdinfo01.html
Revision Information	Last revised August 20, 1999
Document Location	http://www.imsproject.org/metadata/mdbind01.html

Appendix E – Reference Material

AICC Learning Model Definitions⁶

AICC CMI TERM	AICC “Hierarchy of CBT Components”	
	Curriculum	
COURSE	Course	A complete unit of training. A course generally represents what a student needs to know in order to perform a set of related skills or master a related body of knowledge. Course structure provides a method to group lessons into sequences for assignment. This entails support for lesson hierarchies which allow the course developer to define predecessor and successor relationships.”
BLOCK		An arbitrarily defined grouping of course components. Blocks are composed of related assignable units or other blocks. This is a term used in the AICC document CMI Guidelines for Interoperability. A block may correspond to any level of the AICC instructional hierarchy above lesson, up to and including course.
	Chapter	
	Sub-Chapter	
	Module	
Assignable Unit (AU) (aka: lesson)	Lesson	The smallest element of instruction or testing to which a student may be routed by a CMI system. It is the smallest unit the CMI system assigns and tracks. A program or lesson launched by the CMI system. Lesson: A meaningful division of learning that is accomplished by a student in a continuous effort -- that is at one sitting. That part of the learning that is between designed breaks. Frequently requires approximately 20 minutes to an hour. OR A grouping of instruction that is controlled by a single executable computer program. Or A unit of training that is a logical division of a subchapter, chapter, or course.
	Topic	
	Sequence	
	Frame	
	Object	

⁶ Excerpted from: DOCUMENT NO. CMI001 - **CMI Guidelines for Interoperability**, AICC ORIGINAL RELEASE DATE 25-Oct-93 Revision 2.1 release 18 Jun 98

Army Learning Structure Definitions

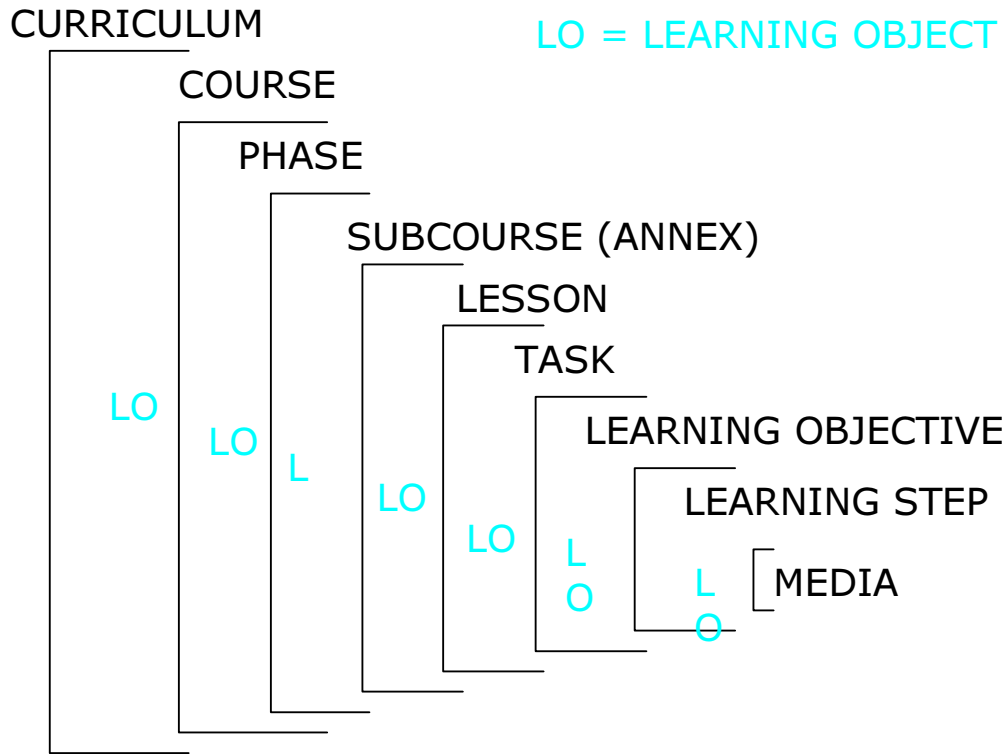
COURSE	A complete series of instructional units(phases, modules and lessons) identified by a common title or number.
MODULE	A- group of lessons in an approved course of instruction; it could consist of a single lesson (e.g., for distance learning). Synonymous with annex and sub-course. A module includes one or more training media/methods or combination thereof.
LESSON	The basic building block of all training. The level at which training is designed in detail. The lesson is structured to facilitate learning. A lesson normally includes telling or showing the soldier what to do and how to do it, an opportunity for the soldiers to practice, and providing the soldiers feedback concerning their performance. A lesson may take the form of an instructor presented lesson, a SGI presented lesson, or a self-paced lesson, such as a correspondence course or CBI lesson. - An instructor presented lesson or SGI presented lesson is documented as a lesson plan. - A self-paced lesson must be of sufficient detail that the student can learn the material to the established learning objective standard on his own. - An extension training lesson is a self paced instructional program developed, reproduced, and packaged for distribution to soldiers in the field. these lessons consist of a terminal learning objective, instructional text, practice, and immediate feedback to the soldier.
LEARNING OBJECTIVE	A precise three-part statement describing what the student is to be capable of accomplishing in terms of the expected student performance under specific conditions to accepted standards. Learning objectives clearly and concisely describe student performance required to demonstrate competency in the material being taught. Los focus the training development on what needs to be trained and focuses student learning on what needs to be learned. Both terminal and enabling objectives are learning objectives. Criterion-referenced Instruction (CRI) - the instruction aimed at training students to perform established learning objectives (performance criteria) to the prescribed standard. CRI is the selected instructional methodology for training within the Army.
LEARNING STEPS	A student activity that leads toward achievement of a learning objective. Learning steps are determined when the objective is broken down into its component parts. Often an explicit hierarchical relationship consisting of terminal learning objective, enabling learning objective, and learning step in maintained. Learning steps are identified and delineated in the lesson, training support package, or Army Correspondence Course Program outline during the design phase. It should be performance oriented.
MEDIA	Media - word document, PowerPoint slide or presentation, avi file that is used to assist the training process.

Air Force Shared Content Object Model

CATEGORY	DEFINITIONS	EXAMPLE/CONTENT
Course	A complete, organized series of instructional elements (modules, lessons, learning objectives) identified by a title and/or course version or number.	<u>Course Name - Labor-Management Relations (LRM)</u> <u>Course Number - OC233M</u> Who can take the course - This course is designed for civilian and military personnel responsible for any aspect of Labor-Management Relations. This course is meant as a basic introduction for personnel working with Labor-Management Relations. It is made up of eight modules. Date Activated - 06-30-99
Block/Module	A series of lessons that cover a general subject. A teaching unit in an approved course of instruction, consisting of one or more lessons with an interlacing theme of function or notion. "Block" is typically used in technical training courses and usually represents a testing milestone. "Module" is associated with education courses.	<u>Module Name - Bargaining Principles and Practices</u> Fifth module of this course, explains some of the different types of bargaining methods, ground rules, bargaining principles and tactics, and some of the various pitfalls. this module is composed of three lessons
Lesson	An aggregation of related topics (the smallest unit of teaching covering one subject only). A division or exercise describing what activity or steps are required to achieve an objective. A single continuous session of formal instruction in terms of the expected student performance under specific conditions.	<u>Lesson Name - Bargaining Preparation</u> The first lesson of Bargaining Principles and Practices. This lesson covers the preparation that needs to take place prior to bargaining with labor or management. This is the lowest level that can effectively stand by itself. Contains topics - <u>Topic Name - Selecting the Negotiation Team</u> One of the topics covered in the lesson Bargaining Preparation is Selecting the Negotiation Team. This topic discusses how to select the right people to represent management during negotiations. A topic is a complete thought, but is not capable of standing alone. Most topics could easily be incorporated into a similar section in some other course
Learning Objective	Defines learner performance expectations. Includes terminal objectives and enabling objectives. Objectives require action, condition, standard. Objectives are the basis for student measurement	<u>Action - Selecting the Negotiation Team</u> <u>Condition - You need to form a team to represent management during negotiations.</u> Standard - proper planning in selecting team members is crucial. Team members should be selected by desirable personal characteristics to include being a team player, a contributor, even-tempered, articulate, analytical, open-minded, and positive.

Air University has chosen to use the model of Course, Module, Learning Objective, and Lesson which accommodates cognitive levels of instruction and testing.

Marine Corps Learning Object Taxonomy



Marine Corps Learning Object

ADL Learning Taxonomy Mapping

Reference Model Level	Function	AICC	Oracle	Asymmetri x	Macromedia	DoD Enterprise Model	Army Learning Model	²	³	Marine Corps	Canadian SCO
COURSE "Packaged Collections"	Outer Container	Course	Course	Course (Block)	Course	Course	Course			Course	Course
BLOCK	Nesting Container	Block	Topic Group	Block	Block	Module	Module			Instructional Unit	Performance Objective
"	Nesting Container (N levels deep)	Block	Topic Group	Module	Block	Instructional Unit (Lesson)	Lesson			(n/a)	
BLOCK "A Collection of Atoms"	Nesting Container (N levels deep)	Block	Topic	Lesson	Block	Learning Objective	Learning Objective			Lesson	Enabling Object
AU "Atomic"	Reusable Content	Assignable Unit	Activity	Page	Assignable Unit	??	Learning Step			Learning Objective	Teaching Point
Raw Media (No Metadata)	Interaction	Interaction			Knowledge Object		Raw Media (no metadata)				

² Air Force (to be added)

³ Navy (to be added)

Appendix F – Document Change Summary

From 0.9.x.4 to 0.9.x.5 (edits 10-25,26-99)

- Added section 7.6 – examples of SCORM XML Metadata records derived from the IMS DTD (illustrates some prospective aspects of conformance testing criteria)
- Removed “old 0.7.3” metadata column in section 7.3 (now very obsolete)
- Many small edits/cleanups throughout, (more to be done here)
- Made elements in section 7.3 the same for content and courses since they seemed so close otherwise. This may change. See section 7.6.3.
- Revamped section 7.3 updating to IEEE LOM 3.7 and adding all missing element categories
- Added new references/links for related documents, especially in appendix D; combined appendix E with D since they are all Metadata related.
- Made paramString under au.launch in CSF DTD optional (thanks Tyde!)
- Added citations in section 2
- Added examples to CSF format, sections 5.8.1+

From 0.9.x.5 to 0.9.x.6 (edits 10-27-99)

- Miscellaneous wordsmithing in section 5
- Added missing text to 5.7 *Extensibility*
- Added missing text to 5.8 *Conformance Testing*
- Added missing text to 6.6 *Conformance*
- Added missing text to 7.5 *Conformance*
- Updated Appendix B

From 0.9.x.6 to 0.9.x.7 (edits 10-30-99)

- Very minor wordsmithing sections 2, 3
- Changed section 4 title to Definitions
- Changed Compliance with conformance throughout
- Fixed up numbering in section 5
- Added new section 6.3 as a placeholder for API adapter discussion; other sections renumbered up one.

From 0.9.x.7 to 0.9.x.8 (edits 11-11-99)

- Minor edits throughout
- Added section 8 Acronyms
- Removed JSIMS Example (Appendix A) – deemed obsolete
- Removed Section 7.4 (Dictionary) since that has been incorporated within IEEE’s table update (3.8) Renumbered sections 7.5+ accordingly
- Updated Section 7.3 (table) to reflect explanation changes in IEEE LOM version 3.8

From 0.9.x.8 to 0.9.x.9 (edits 12-6-99)

- Revised Section 5 as follows:
 1. Removed “assignments” from the CSF and replaced it with “block” – this was because it appears that the root node (formerly “assignments”) needs mostly all of the same elements as “block”. Since “blocks” nest n deep, it seemed to make sense to not differentiate between root and nodes; i.e., they should be the same thing. Note that this does not appear to be the case with nodes (blocks) and leaves (aus) since there are differences.
 2. Removed “content” element (and its sub-elements) on the basis that the data that would be contained in these elements should be available within externally referenced, LOM-based metadata records.

From 0.9.x.9 to 0.9.x.10 (edits 12-7-99)

- Section 5.6, 5.10 and figure 5.4.2a: renamed element “score” to “masteryScore” and eliminated sub-elements “mastery”, “maximum”, and “minimum” since these elements don’t belong here (they are in the CMI data model and have no meaning here; however, “masteryScore” does have context-specific value. This element defines the mastery score the student must achieve *in this course* context, which overrides whatever default mastery value an *au* might have defined.

From 0.9.x.11 to 1.0

- Miscellaneous minor edits throughout
- Added sections 1.1, 1.2, 1.3
- Section 6.1 modified figure 6.1.a showing content/au
- Added examples to 6.3, 6.4
- Added Section 6.6 – further defining content as Assignable Units (renumbered 6.7+)
- Added data model tables
- Added section 7.4 Stand-Alone XML Metadata Records
- Added section 7.5 XML Schema, Namespaces and Extensibility
- Renumbered old 7.4+ to 7.6+
- Changed appendices sequence; moved dtd to appendix A
- Minor edits throughout sections 1 and 2
- Incorporated 3.5 into 3.4
- Added section 8 examples
- Added examples and ref tables throughout
- General editing
- Editing Throughout.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE July 2000		2. REPORT TYPE Final		3. DATES COVERED (From-To) July 1999-December 1999	
4. TITLE AND SUBTITLE Sharable Courseware Object Reference Model (SCORM), Version 1.0				5a. CONTRACT NUMBER DAS W01 98 C 0067	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Ron Ball, Richard Burke, Dexter Fletcher, Alan Hoberney, Paul Jesukiewicz, Philip Dodds (Editor)				5d. PROJECT NUMBER	
				5e. TASK NUMBER AK-2-1800	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Institute for Defense Analyses 4850 Mark Center Drive Alexandria, VA 22311-1882				8. PERFORMING ORGANIZATION REPORT NUMBER IDA Document D-2461	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) USD (P&R) Readiness and Training Room 1C757, The Pentagon Washington, DC 20301				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Open Publication; unlimited distribution (October 23, 2002).					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The Department of Defense (DoD), in cooperation with the Office of Science and Technology, established the Advanced Distributed Learning (ADL) initiative to develop a strategy and process for using learning and information technologies together to modernize education and training in the DoD and across federal agencies. The ADL initiative has defined high-level requirements for learning content or learning content "objects." These requirements include accessibility, reusability, durability, and interoperability. To meet these requirements, content objects must be developed in accord with specifications that are widely agreed upon, accepted, and used. The Sharable Courseware Object Reference Model (SCORM) was developed to articulate clear and implementable specifications for the development of Web-based learning content objects. Version 1.0 addresses three major issues: Course Structure Format, Runtime Environment, and Metadata. Later versions will address these and other issues and needed to develop the specification.					
15. SUBJECT TERMS distance learning, distributed learning, education, instructional objects, learning management systems, metadata, object-oriented development, reusable content, sharable content, technology-based instruction, Web-based instruction					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT SAR	18. NUMBER OF PAGES 208	19a. NAME OF RESPONSIBLE PERSON Dr. Robert Wisher
a. REPORT Uncl.	b. ABSTRACT Uncl.	c. THIS PAGE Uncl.			19b. TELEPHONE NUMBER (include area code) 703-695-2618

