DOCUMENT NO. CMI001

CMI Guidelines for Interoperability
AICC

ORIGINAL RELEASE DATE 25-Oct-93
Revision 4.0 release 16-Aug-2004

THIS DOCUMENT IS CONTROLLED BY:

AICC CMI Subcommittee

ALL REVISIONS TO THE DOCUMENT SHALL BE APPROVED
BY THE ABOVE ORGANIZATION PRIOR TO RELEASE.

POINT OF CONTACT:

Scott Bergstrom
AICC Administrator
P.O. Box 472
Sugar City, ID 83448-0472

Telephone: (208) 496-1136
E-mail address: admin@aicc.org

PREPARED ON PC FILED UNDER CMI001v4.doc

Contributing Editors

William A. McDonald – Alteon (A Boeing Company), AICC CMI Subcommittee Chair
Jack Hyde - AICC Technical Advisor
Ann Montgomery - AICC Technical Coordinator

Partial List of Contributors

Mark Schupp – Integrity eLearning
Jacques Talvard – Airbus Industrie
Brett Watters – Geometrix
Bradley K. Weage – Learn.Net
Greg Tobin – Heathkit Educational Systems
Nathan Summers – FutureMedia
Tom King - Macromedia
Bernard Bouyt – Airbus Industrie
Paul Bishop – Plan Three Solutions
Ed Cohen – Plateau Systems
Jon Conradt – Gallup
Jonathan Zemple - IBM
Paul Roberts – Question Mark, Ltd.
John Kleeman – Question Mark, Ltd.

Revision History

REV 4.0 (June 2004) Complete rewrite and reorganization of all sections. See CMI001 version 3.5 for a complete revision history back to version 1.0. This revision is intended to be functionally equivalent to CMI001 version 3.5

.Major changes include:

- All definitions were narrowed and clarified.
- Conflicting rules and statements clarified/resolved.
- Structured notation was added for every data element to define data types, range of data, and data vocabularies.
- Communication and course structure data models separated from individual bindings (methods of implementation). All bindings were mapped to the data models in separate sections.
- The content of all appendices (Appendix A, and Appendix B) were merged into the main body of the document.

Table of Contents

1.0 Overview

1.1 Purpose

The purpose of this document is to define interfaces and rules that allow CBT (Computer-Based Training) content from a variety of sources to interoperate with CMI (Computer Managed Instruction) systems.

1.2 Scope

This document defines the following:

- The mechanism used by the CMI to launch CBT content
- Common mechanisms and data for CMI/CBT communication
- A common definition for organization and sequencing of CBT content in a course.

Following items are outside the scope of this document:

- User interface appearance
- Pedagogy

1.3 Document Organization

Document Section	Description
1.0 Overview	General Description and Overview
2.0 Communication Data Model	Describes all data used for communication between CBT assignable units and the CMI system. Each data element is cross-referenced to all relevant bindings.
3.0 Course Structure Data Model	A description of all data used to define a course structure. Each data element is cross-referenced to all relevant bindings
4.0 Assignable Unit Sequencing within a Course	A detailed explanation of how sequencing rules in a course are used.
5.0 Communicating via Files (The File Binding)	Defines the requirements for implementing the communication data model using files.
6.0 Communicating via HTTP (The HACP Binding)	Defines the requirements for implementing the communication data model using HTTP messages.
7.0 Communicating via API (The API Binding)	Defines the requirements for implementing the communication data model using a JavaScript API.
8.0 Course Structure Definition (File Binding)	Defines the requirements for implementing the course structure data model using files.
9.0 Data Types	Definition and format of data types used by the various data models and their bindings.
10.0 Augmented Backus-Naur Form (BNF) Notation	The structured notation used to describe the formatting of data types in this document
11.0 Glossary	Definition of terms used in this document.
12.0 References	List of external documents referenced.

1.4 Conformance Requirements

This specification defines interoperability for the following environments:

- File-based (Local file system and program execution.)
- Web-based (Using a web browser)

The conformance requirements for each environment are described in the following sections.

1.4.1 File-based Environments

A conforming CMI system in the file-based environment must meet all conformance requirements described in the following sections:

5.0 Communicating via Files (The File Binding)
8.0 Course Structure Definition (File Binding)

A conforming Assignable Unit (AU) in the file-based environment must meet all conformance requirements described in the following section:

5.0 Communicating via Files (The File Binding)

1.4.2 Web-based Environments

A conforming CMI system in the web-based environment must meet all of the conformance requirements described in the following sections:

6.0 Communicating via HTTP (The HACP Binding)
7.0 Communicating via API (The API Binding)
8.0 Course Structure Definition (File Binding)

A conforming Assignable Unit (AU) in the web-based environment must meet all of the conformance requirements described in ***either*** of the following sections:

6.0 Communicating via HTTP (The HACP Binding)
7.0 Communicating via API (The API Binding)

2.0 Communication Data Model

This section covers all that data that may be communicated between the CMI and the AU.
Each data element is this model may appear in one or more of the following bindings:

File-Based	A text-file binding for use in LAN/CD-ROM based systems. (See section 5.0.)
HACP	An HTTP-based binding which may be used for Web implementations. (See section 6.0)
API	A JavaScript API binding which may also be used for Web implementations. (See section 7.0)

In general, a data element is used in the same manner across all bindings, but there are some important distinctions to be made for each binding:

1. Each binding has different rules for formatting data
2. Each binding also operates in a different environment with different transport mechanisms.
3. Some data elements may be specific only to a particular binding.

The data elements in this model are arranged hierarchically (in a "parent/child" relationship). Hierarchy levels are delimited by period (".")s in the data element name. Any item to the right of the period delimiter is the "child" of preceding item (e.g. in "Core.Score" , "Core.Score" is a child of "Core" and "Core" is the parent of "Core.Score")

The table below list all elements in this data model. Each element is described in the section indicated.

Table Legend:

Name	Indicates the name of the data element.
Section	Indicates where in this document a definition of the data element is found.
CMI Obligation	This indicates whether the data element is required or optional for a CMI system.

Data Element	Section	CMI Obligation
Core	2.1	Mandatory
Core.Student Id	2.1.1	Mandatory
Core.Student Name	2.1.2	Mandatory
Core.Output File	2.1.3	Mandatory
Core.Lesson Location	2.1.4	Mandatory
Core.Credit	2.1.5	Mandatory
Core.Lesson Status	2.1.6	Mandatory
Core.Exit	2.1.7	Mandatory
Core.Entry	2.1.8	Mandatory
Core.File Path	2.1.9	Mandatory
Core.Score	2.1.10	Mandatory
Core.Session Time	2.1.11	Mandatory
Core.Total Time	2.1.12	Mandatory
Core.Lesson Mode	2.1.13	Optional
Suspend Data	2.2	Mandatory
Launch Data	2.3	Mandatory
Comments From Learner	2.4	Optional
Itemized Comments From Learner	2.5	Optional
Itemized Comments From Learner.Content	2.5.1	Optional
Itemized Comments From Learner.Date	2.5.2	Optional
Itemized Comments From Learner.Location	2.5.4	Optional
Itemized Comments From Learner.Time	2.5.5	Optional
Comments From LMS	2.6	Optional
Evaluation	2.7	Optional

Data Element	Section	CMI Obligation
Evaluation.Comments_File	2.7.1	Optional
Evaluation.Course_Id	2.7.2	Optional
Evaluation.Interactions_File	2.7.3	Optional
Evaluation.Objective_Status_File	2.7.4	Optional
Evaluation.Path_File	2.7.5	Optional
Evaluation.Performance_File	2.7.6	Optional
Objectives	2.8	Optional
Objectives.ID	2.8.1	Optional
Objectives.Score	2.8.2	Optional
Objectives.Status	2.8.3	Optional
Objectives.Date	2.8.4	Optional
Objectives.Time	2.8.5	Optional
Objectives.Mastery Time	2.8.6	Optional
Student Data	2.9	Optional
Student Data.Attempt Number	2.9.1	Optional
Student Data.Tries	2.9.2	Optional
Student Data.Tries.Score	2.9.2.1	Optional
Student Data.Tries.Status	2.9.2.2	Optional
Student Data.Tries.Time	2.9.2.3	Optional
Student Data.Mastery Score	2.9.3	Optional
Student Data.Max Time Allowed	2.9.4	Optional
Student Data.Time Limit Action	2.9.5	Optional
Student Data.Tries During Lesson	2.9.6	Optional
Student Data.Sessions Journal	2.9.7	Optional
Student Data.Sessions Journal.Lesson Score	2.9.7.1	Optional
Student Data.Sessions Journal.Lesson Status	2.9.7.2	Optional
Student Preference	2.10	Optional
Student Preference.Audio	2.10.1	Optional
Student Preference.Language	2.10.2	Optional
Student Preference.Lesson Type	2.10.3	Optional
Student Preference.Speed	2.10.4	Optional
Student Preference.Text	2.10.5	Optional
Student Preference.Text Color	2.10.6	Optional
Student Preference.Text Location	2.10.7	Optional
Student Preference.Text Size	2.10.8	Optional
Student Preference.Video	2.10.9	Optional
Student Preference.Windows	2.10.10	Optional
Interactions	2.11	Optional
Interactions.ID	2.11.1	Optional
Interactions.Objectives	2.11.2	Optional
Interactions.Date	2.11.3	Optional
Interactions.Time	2.11.4	Optional
Interactions.Type	2.11.5	Optional
Interactions.Correct Responses	2.11.6	Optional
Interactions.Weighting	2.11.7	Optional
Interactions.Student Response	2.11.8	Optional
Interactions.Result	2.11.9	Optional
Interactions.Latency	2.11.10	Optional
Paths	2.12	Optional
Paths.Location Id	2.12.1	Optional
Paths.Date	2.12.2	Optional
Paths.Time	2.12.3	Optional
Paths.Status	2.12.4	Optional
Paths.Why Left	2.12.5	Optional
Paths.Time In Element	2.12.6	Optional
Student Demographics	2.13	Optional
Student Demographics.City	2.13.1	Optional
Student Demographics.Class	2.13.2	Optional
Student Demographics.Company	2.13.3	Optional
Student Demographics.Country	2.13.4	Optional
Student Demographics.Experience	2.13.5	Optional
Student Demographics.Familiar Name	2.13.6	Optional
Student Demographics.Instructor Name	2.13.7	Optional
Student Demographics.Native Language	2.13.8	Optional

Data Element	Section	CMI Obligation
Student Demographics.State	2.13.9	Optional
Student Demographics.Street Address	2.13.10	Optional
Student Demographics.Telephone	2.13.11	Optional
Student Demographics.Title	2.13.12	Optional
Student Demographics.Years Experience	2.13.13	Optional

Each element in this data model is described in tables in the following sections. The fields for each of these tables are as follows:

Data Element Name
The data elements in this model are arranged hierarchically (in a "parent/child" relationship). Hierarchy levels are delimited by period (".")s in the data element name. Any item to the right of the period delimiter is the "child" of preceding item (e.g. in "Core.Score" , "Core.Score" is a child of "Core" and "Core" is the parent of "Core.Score").

Definition
A description of the data element.

Usage
What the data element is used for, and rules for its usage.

CMI Behavior Notes
A description of the expected or recommended CMI behavior when using the data element. (This field augments "Usage")

AU Behavior Notes
A description of the expected or recommended AU behavior when using the data element. (This field augments "Usage")

File Binding: Name
Name used for the data element in the file binding.

File Binding: Files and Obligations
The requirement for CMI or AU to read/write the data element in the files.

File Binding: Name Format
Formatting for the Name of the data element written in the files.

File Binding: Value Format
This field adds additional explanation for valid values that a field may have (in addition to the definition that *data type* provides).

File Binding: DataType
Each data element binding is assigned a "data type". The data type defines the size of data element and the valid ranges of values. See *section 10. Data Types*

File Binding: Examples
Examples of how data element is represented in files.

HACP Binding: Name
Name used for the data element in the HACP binding.

HACP Binding: HACP Message(s) and Obligations
The requirement for CMI or AU to read/write the data element in the HACP messages specified.

HACP Binding: Name Format
Formatting for the name of the data element included in the HACP messages.

HACP Binding: Value Format
This field adds additional explanation for valid values that a field may have (in addition to the definition that *data type* provides).

HACP Binding: Data type
Each data element binding is assigned a "data type". The data type defines the size of data element and the valid ranges of values. See *section 10. Data Types*

HACP Binding: Examples
Examples of how the data element is represented in HACP messages.

API Binding: Name
Name used for the data element in the API binding.

API Binding: API's and Obligations
The requirement for CMI or AU to read/write the data element using the API.

API Binding: Name Format
Formatting for the name of the data element when using the API

API Binding: Value Format
This field adds additional explanation for valid values that a field may have (in addition to the definition that *data type* provides).

API Binding: Data type
Each data element binding is assigned a "data type". The data type defines the size of data element and the valid ranges of values. See *section 10. Data Types*

API Binding: Examples
Examples of how the data element is represented in API calls.

2.1 Core

Data Element Name	Core
Definition	A grouping for a variety of important data elements.
Usage	Most data elements in this category are required to be furnished by all CMI systems. Mandatory members of this group are what all AU's may depend upon at start up. Individual members of group are not necessarily all mandatory. (See individual member data elements for obligations)
Membership	Core.Student ID Core.Student Name Core.Output File Core.Lesson Location Core.Credit Core.Lesson Status Core.Exit Core.Entry Core.File Path Core.Score Core.Session Time Core.Total Time Core.Lesson Mode

2.1.1 Core.Student ID

Data Element Name	Core.Student ID	
Definition	Unique alpha-numeric code/identifier that refers to a single user of the CMI system.	
Usage	Used to uniquely identify a student. The AU obtains this element on startup in order to associate *Core.Student ID* with other optional data elements (in the file and HACP bindings). The AU may also use *Core.Student ID* for display purposes in all bindings.	
CMI Behavior Notes	This element may be associated with the CMI system's login name for a given student. (But is not required to be equivalent)	
AU Behavior Notes		
File Binding		
Name	Student_ID	
Files & Obligations	Startup: CMI Mandatory Comments: If file exists, AU Mandatory Interactions: If file exists, AU Mandatory Objectives Status: If file exists, AU Mandatory Path: If file exists, AU Mandatory	
Name Format	"Student_ID" Case insensitive.	
Value Format	See description of data type *CMIIdentifierINI*	
Data type	CMIIdentifierINI	
Examples	Student_ID=Ted_Roosevelt1	
	Student_id = JQH-1959	
	STUDENT_id =jack1991-3	
HACP Binding		
Name	Student_ID	

Data Element Name	Core.Student ID
HACP Message(s) & Obligations	GetParam(response) : CMI Mandatory PutComments: Optional PutInteractions: Optional PutObjectives Status: Optional PutPath: Optional
Name Format	Same as File binding
Value Format	Same as File binding
Data type	Same as File binding
Examples	Same as File binding
API Binding	
Name	cmi.core.student_id
API & Obligations	LMSGetValue(): CMI Mandatory
Name Format	"cmi.core.student_id" Case sensitive.
Value Format	Alphanumeric group of characters with no white space or unprintable characters in it. Maximum of 255 characters.
Data type	CMIIdentifierINI
Examples	Stu_id = LMSGetValue("cmi.core.student_id")

2.1.2 Core.Student Name

Data Element Name	Core.Student Name
Definition	Normally, the official name used for the student on the course roster. A complete name, not just a first name.
Usage	Used to represent the student's official name
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Student_Name
Files & Obligations	Startup : Mandatory
Name Format	"Student_Name" case insensitive
Value Format	See DataType *CMIStudentName* for detailed formatting rules.
Data type	CMIStudentName
Examples	STUDENT_NAME = Blough, Joseph
	studeNT_nAME = Brown ,
	student_name = Smith-Farley von Sant , Johann A
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam(response) : Mandatory
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.student_name
API & Obligations	LMSGetValue() : Mandatory
Name Format	"cmi.core.student_name" case sensitive

Value Format	Same as File Binding
Data type	Same as File Binding
Examples	var StudentName = LMSGetValue("cmi.core.student_name");

2.1.3 Core.Output File

Data Element Name	Core.Output File
Definition	A fully qualified file path for the Finish file, which the AU must construct if it is to pass information back to the CMI system.
Usage	AU writes output data (i.e. the Output_File) to location specified in this element (used for the File binding only).
CMI Behavior	CMI determines the location for Finish file
AU Behavior	AU writes the Finish file at this location prior to session termination.
File Binding	
Name	Output_File
Files & Obligations	Startup : CMI Mandatory
Name Format	"Output_file" case insensitive
Value Format	See description for the *CMIFileNameFull* data type
Data type	CMIFileNameFull
Examples	Output_file = C:\windows\outparam.cmi
	OUTPUT_FILE = BB:\ r
	OUTPUT_FILE = C:\ directory with spaces\file with spaces.txt
HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.1.4 Core.Lesson Location

Data Element Name	Core.Lesson Location
Definition	This corresponds to the AU exit point passed to the CMI system the last time the student experienced the AU. This element provides a mechanism to let the student return to an AU at the same place he/she left it earlier. This element can identify the student's exit point and that exit point can be used by the AU as an entry point the next time the student runs the AU.
Usage	The element could be used by the AU to store resume information for a session. If the AU is exited, and then is re-entered later, this element could be used by the AU to send the student back into the AU where they left off.

Data Element Name	Core.Lesson Location
	This element is only set by the AU. The CMI must always return the value provided from the previous AU session. The first time a student enters the AU, the value of *Core.Lesson Location* is an empty string ("").
CMI Behavior Notes	The CMI must set aside a space for this element for each AU in the course(s) for each student. It stores this data and returns it to the AU when it is run again. The CMI shall retain this data as long as the student is enrolled in (or has access to) the course CMI must always return the value previously stored by the AU in this element (in the last AU session). The CMI is not required to report on this data element.
AU Behavior Notes	The AU is not required to read/use this element
File Binding	
Name	Lesson_Location
Files & Obligations	Startup: CMI Mandatory Finish: CMI Mandatory
Name Format	"Lesson_Location" case insensitive
Value Format	Implementation dependent. Carriage returns, and line feeds are not allowed. See datatype *CMIString255INI*.
Data type	CMIString255INI
Examples	Lesson_Location = 1,,,,,2
	Lesson_Location = Page 1
	Lesson_Location = #$#&^%&^*$Q#)*%afgfg
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam(Response) : CMI Mandatory PutParam : CMI Mandatory
Name Format	Same as File Binding
Value Format	Implementation dependent. Carriage returns, and line feeds are not allowed. See datatype *CMIString255INI*
Data type	CMIString255INI
Examples	Same as File Binding
API Binding	
Name	cmi.core.lesson_location
API & Obligations	LMSGetValue() : CMI Mandatory LMSSetValue() : CMI Mandatory
Name Format	"cmi.core.lesson_location" case sensitive
Value Format	Implementation dependent. Carriage returns, and line feeds are not allowed. See datatype *CMIString255INI*
Data type	CMIString255INI
Examples	var AULocation = LMSGetValue("cmi.core.lesson_location");

2.1.5 Core.Credit

Data Element Name	Core.Credit

Data Element Name	Core.Credit
Definition	Indicates whether the student is being credited by the CMI system for his performance (pass/fail and score) in this AU.
Usage	Used by the CMI system to indicate to the AU whether the student is being given credit for his or her score and status in the usage of the content. There are two possible arguments for this keyword, Credit or No-credit. • **Credit**. The student is taking the AU "for credit". The CMI system is telling the AU that if the AU sends data to the CMI system, the CMI system will credit it to the student. (i.e. record status and score related changes normally) • **No-credit**. The student is <u>not</u> taking the AU "for credit". The current credit will not be changed by the student's performance in this AU session. With this value the CMI system is communicating to the AU that if the AU sends data to the CMI system, it will not change the student's accreditation. (i.e. will NOT record status and score related changes). When a CMI sets the value of this data element to "No-credit" at AU launch, certain elements are **<u>not</u>** updated with AU session data. These elements are as follows: *Core.Score* *Objectives.Score* *Objectives.Status* All other data elements (with the exception of *Core.Lesson Status*) are normally updated with AU's session data when *Core.Credit* is set to "No Credit". When *Core.Credit* is set to "No Credit, *Core.Lesson Status* can only be changed from a value of "Not Attempted" to "Browsed", otherwise *Core.Lesson Status* is not updated (as a result of "No Credit" AU session). When *Core.Lesson_Mode* is set to "Browsed", *Core.Credit* must be set to "No Credit". (See *Core.Lesson_Mode*.)
CMI Behavior	CMI determines whether an AU is to be taken "for Credit". Usually via a student's user interface selection.
AU Behavior	If an unrecognized or unanticipated CREDIT value is received, then Credit is assumed by the AU.
File Binding	
Name	Credit
Files & Obligations	Startup: CMI Mandatory
Name Format	"credit" Case insensitive.
Value Format	One of two words: "credit" or "no-credit". Case insensitive. Only the first character is significant.
Data type	CMIVocabularyINI:Credit
Examples	Credit=c Credit = credit credit = N
HACP Binding	
Name	Credit
HACP Message(s) & Obligations	GetParam (response): CMI Mandatory

Data Element Name	Core.Credit
Name Format	Same as file binding.
Value Format	Same as file binding.
Data type	Same as file binding.
Examples	Same as file binding.
API Binding	
Name	cmi.core.credit
API & Obligations	LMSGetValue():CMI Mandatory
Name Format	"cmi.core.credit" Case sensitive.
Value Format	"credit" or "no-credit" Case sensitive. All characters must be present.
Data type	CMIVocabulary:Credit
Examples	grading = LMSGetValue("cmi.core.credit")

	var creditFlag = LMSGetValue("cmi.core.credit")
	if (creditFlag == "credit"
	{
	// Student is taking course for credit. Handle appropriately.
	}
	else
	{
	// Student is taking course for no credit. Handle appropriately.
	}

2.1.6 Core.Lesson Status

Data Element Name	Core.Lesson Status
Definition	The current student status for a given AU. The CMI system determines this status based on data returned from the AU and other factors. Six status values are possible: • **passed**: A necessary number of objectives in the AU were mastered by the student, and/or the necessary score was achieved. Student is considered to have "completed" the AU and "passed". • **completed**: The AU may or may not be "passed", but all the elements in the AU were experienced by the student. The student is considered to have completed the AU. For instance, "passing" may depend on a certain minimum score known to the CMI system. • **failed**: The AU was not passed. The student experienced some kind of assessment within the AU but did not demonstrate mastery of the AU's instruction material. The student has viewed some (or all) of the AU's instructional material. • **incomplete**: The AU was started but not finished. The student did not view all the required elements in the AU. • **browsed**: The student launched the AU with a CMI mode of Browse. • **not attempted**: Incomplete implies that the student made an attempt to perform the AU, but for some reason was unable to finish it. Not attempted means that the student did not even begin the AU. Maybe he just read the table of contents, or AU abstract and decided he was not ready. Any algorithm within the AU may be used to determine when the AU moves from "not attempted" to "incomplete".
Usage	The CMI initializes *Core.Lesson Status* to "not attempted". Except for course structures with complex logic statements, a *Core.Lesson Status* value of "passed" or "completed" is treated the same for course prerequisites and completion requirements.

Data Element Name	Core.Lesson Status
	Normally, the AU determines *Core.Lesson Status* and passes it to the CMI. On re-entry into the AU, the CMI passes the previous status returned by the AU. However, the CMI can change the status based on the following rules: 1) If Core.Credit is set to "credit" and there is a value for *Student Data.Mastery Score* and the AU returns a value for *Core.Score.Raw*, the CMI can change the status to either passed or failed depending on whether the student's score meets/exceeds *Student Data.Mastery Score*. If there is no value returned by the AU session for *Core.Score.Raw,* the CMI does not change the status using this rule. 2) If the AU is part of a course that has completion requirements in its course structure, then the CMI can change the status depending on the completion requirements rules defined *(*see *Course Elements.Completions.Requirement).* 3) If there is no value for *Student Data.Mastery Score* passed to AU and there are no completion requirements rules defined (in the course structure), then the CMI cannot override an AU determined status. 4) If the CMI sets *Core.Credit* to "no-credit" for the AU session, the CMI is not allowed to change/update *Core.Lesson Status* unless the initial value of *Core.Lesson Status* was "not attempted". In this particular case, *Core.Lesson Status* is changed to "browsed". (See *Core.Credit)* 5) The CMI cannot change a previously (CMI) recorded *Core.Lesson Status* to "not attempted" in the course of normal operation.
CMI Behavior	The CMI is responsible for setting the initial value of *Core.Lesson* Status to "not attempted". The CMI may further "preset" the value of *Core.Lesson* Status (prior to the first student launch) based the completion requirements rules *(*see *Course Elements.Completions.Requirement).* Manual manipulation of *Core.Lesson.Status* by administrative users is outside the scope of this specification.
AU Behavior	In File & HACP binding's the AU is required to report status. With the API binding, the AU is not required to report status.
File Binding	
Name	Lesson_Status
Files & Obligations	Startup: CMI Mandatory Finish: AU Mandatory
Name Format	"Lesson_Status" case insensitive
Value Format	One of the following vocabulary values: "passed" , "failed", "complete", "incomplete", "not attempted", or "browsed". All values are case insensitive. Only the first character is significant.
Data type	CMIVocabularyINI:Status
Examples	lesson_status = Passed
	LESSON_STATUS = c
	LessoN_Status = F
HACP Binding	
Name	Same as File Binding

Data Element Name	Core.Lesson Status
HACP Message(s) & Obligations	GetParam(response) : CMI Mandatory PutParam : AU Mandatory
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.lesson_status
API & Obligations	LMSGetValue() : CMI Mandatory LMSSetValue() : CMI Mandatory
Name Format	"cmi.core.lesson_status" case sensitive
Value Format	A specific vocabulary limited to one of the following values: "passed", "completed", "failed" , "incomplete", "browsed", or "not attempted". All values are case sensitive
Data type	CMIVocabulary:Status
Examples	var x = LMSGetValue("cmi.core.lesson_status") LMSSetValue("cmi.core.lesson_status", "passed")

2.1.7 Core.Exit

Data Element Name	Core.Exit
Definition	An indication of how or why the student left the AU.
Usage	This element can only be set by the AU. There are four possible values: • **"time-out":** This indicates the AU ended because the AU has determined an excessive amount of time has elapsed with no student interaction, or the "max_time_allowed" has been exceeded. • **"suspend":** This indicates the student leaves the AU with the intent of returning to it later at the point where he/she left. • **"logout":** This indicates that the student logged out from within the AU instead of returning to the CMI system to log out. The AU passed control to the CMI system, and the CMI system automatically logged the student out of the course -- after updating the appropriate data model elements. CMI would then require the student to re-authenticate (login) prior to viewing any other material in the course. • **Empty string** – an empty string ("") or no value given indicates a normal exit state.
CMI Behavior Notes	The CMI does not initialize this element. **"logout" behavior**: If the CMI receives a logout value from an AU it must logout the student (after the AU session terminates). The student would then be required to re-authenticate (login) prior to viewing other material in the course. **"time-out" behavior**: The CMI may provide a visual cue to the student indicating that the reporting AU was terminated due to a time-out. The CMI may also exhibit logout behavior in addition to this visual cue. **"suspend" behavior**: The CMI may provide a visual cue indicating that the student exited with AU with the intent of returning to it later. The CMI must set Core.Entry to "resume" on the next launch of this AU.

Data Element Name	Core.Exit
AU Behavior Notes	**"logout" behavior**: The AU should provide a visual cue to the student as to which action will cause a logout value to be reported to the CMI.
File Binding	
Name	AU Lesson_Status Flag
Files & Obligation	Finish : CMI Mandatory
Name Format	Not Applicable. It is an appended to the Lesson_Status keyword/value pair.
Value Format	This element is appended to the keyword/value pair of *Lesson_Status* with "," (comma) preceding it. There may be spaces trailing and leading this comma. The element value is case-insensitive with only the first character being significant. If the element is not present, a normal exit shall be assumed.
Data type	CMIVocabularyINI:Exit
Examples	LESSON_Status = Passed, Logout
	Lesson_Status = Complete, t
	LESSON_Status = I, S
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligation	PutParam : CMI Mandatory
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.exit
API & Obligation	LMSSetvalue() : CMI Mandatory
Name Format	"cmi.core.exit" – case sensitive
Value Format	The value must be one of the following: "time-out" , "logout" , "suspend" or the empty string ("").
Data type	CMIVocabulary:Exit
Examples	LMSSetValue("cmi.core.exit","time-out")

2.1.8 Core.Entry

Data Element Name	Core.Entry
Definition	Indication of whether the student has entered the AU before.
Usage	This element is set by the CMI and is only readable by the AU. Three possible values for Core.Entry : • **"ab-initio":** This indicates it is the first time the student is entering the AU. Because the student may have passed all of the objectives in a AU by completing a pre-test, the lesson_status of not attempted is not a reliable indicator. That is, an AU may be passed without the student having ever seen it. • **"resume":** This indicates that the student was in the AU earlier. The student is resuming a suspended AU. Core.Entry is only set to this value if *Core.Exit* was set to "suspend" in the previous AU session. • **"":** The empty string should be used to represent an entry into the AU that is neither an initial (ab-initio) nor a continuation from a suspended

Data Element Name	Core.Entry
	state (resume). A scenario that might be used is if the AU was already completed and then later it was loaded for review purposes. In this case it was neither an initial launch (ab-initio) nor a continuation from a suspended state (resume). When a student enters the AU for the first time the Core.Entry element must be set to "ab-initio" by the CMI. If the student re-enters an AU that previously exited with a value of "suspend", then the entry flag must be set to "resume" by the CMI.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	CMI Lesson_Status Flag
Files & Obligations	Startup: CMI mandatory
Name Format	Not Applicable. It is an appended to another keyword/value pair
Value Format	This element is appended to the keyword/value pair of Lesson_Status with "," (comma) preceding it. There may be spaces trailing and leading this comma. The element value is case-insensitive with only the first character being significant. This element is not present if the value is empty string.
Data type	CMIVocabularyINI:Entry
Examples	LESSON_STATUS = NA , A
	lesson_status = p, a
	lesson_status = f, r
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response) : CMI Mandatory
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.entry
API & Obligations	LMSGetValue() : CMI mandatory
Name Format	"cmi.core.entry" case sensitive
Value Format	One of the following values: "ab-initio", "resume" , or empty string (""). All values are case sensitive.
Data type	CMIVocabulary:Entry
Examples	var entry_val = LMSGetValue("cmi.core.entry")

2.1.9 Core.File Path

Data Element Name	Core.File Path
Definition	This element indicates to the AU where additional (AU-specific) data files may be written by the AU. The directory path indicated by this element is unique to an individual student for a given AU in a given course.
Usage	A (logically or explicitly) unique directory location must be maintained by the CMI for an individual student data for a given AU in a given course.

Data Element Name	Core.File Path
	The path to this location must be provided to the AU at launch time.
CMI Behavior Notes	CMI Implementations of this element will vary widely depending distribution of writable local drive volumes, (network) shared drive volumes, and storage management features. CMI implementations may require the student to use a specific workstation or specific shared network volumes to support this element.
AU Behavior Notes	
File Binding	
Name	Path
Files & Obligations	Startup : CMI Mandatory
Name Format	"Path" – case insensitive
Value Format	Fully qualified Windows directory path specification with drive letter(s), directory path. <Drive Letter>**:**<directories>**** Embedded spaces in directory names are allowed. Non printable characters and **< > ? * " / \ :** are not allowed in directory names. Directory names are separated by \'s (back slashes). Leading and trailing spaces are not allowed around the back slashes.
Data type	CMIDirectoryNameFull
Examples	Path=X:\CMI student data\course 101\joe student\
	Path = D:\USERDATA\CRS123\USER123\
HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.1.10 Core.Score

Data Element Name	Core.Score

Data Element Name	Core.Score
Definition	This data element indicates the performance of the student during his last session in the assignable unit. It may have up to three sub-elements: Core.Score.Raw, Core.Score.Max, and Core.Score.Min. **Score.Raw** This may be an unprocessed or processed indicator of how the student performed with the interactions he experienced. **Score.Max** This is the largest score the student could have achieved with the interactions that he experienced. **Score.Min** This is the smallest score that the student could have achieved with the interactions he/she experienced.
Usage	If Score.Raw is not accompanied by Score.Max or Score.Min, it may be determined and calculated in any manner that makes sense to the program designer. For instance, it could reflect the percentage of objectives complete, it could be the raw score on a multiple choice test, or it could indicate the number of correct first responses to the embedded questions in the AU. If the value return by the AU session for *Score.Raw* is empty string (“”) , then the student is considered to have not visited the scored portion of the content. If Score.Raw is accompanied by Score.Max or Score.Min, it reflects the performance of the learner relative to the max and min values. If Score.Max accompanies Score.Raw with no Score.Min, Score.Min is assumed to be “0”. If Score.Min is included then Score.Max must be also be included. The value of each of the score sub-elements (in relation to one another) must be as follows: Score.Max >= Score.Raw >= Score.Min The AU is responsible for setting this element and the CMI is responsible for providing the previous AU session value for this element given the rules: • CMI must initialize this element to an empty string ("") upon initial launch of an AU. • On subsequent launches of an AU, The CMI must provide the value recorded by the AU in the last session in which *Core.Credit* had a value of “credit. • If the AU sets this value multiple times in a session, only the final value is recorded by the CMI (When *Core.Credit* has a value of “credit”)
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Score

Data Element Name	Core.Score
In Files & Obligations	Startup, Finish: Core.Score.Raw: CMI Mandatory. AU Mandatory Core.Score.Max: If *Core.Score.Min* exists, then CMI and AU Mandatory, otherwise optional. Core.Score.Min: CMI and AU Optional
Name Format	"Score" Case insensitive.
Value Format	Empty string or comma separated list of numeric scores. See description for data type *CMIScoreINI*
Data type	CMIScoreINI
Examples	SCORE= 79
	SCORE= 0.654
	Score = 8, 10 , 0 ; Raw score of 8 with a maximum possible of 10 and minimum of 0.
	score=1.3, 2 ; Raw score of 1.3 with a maximum of 2. Min is assumed to be 0.
	Score= ; Either the student's first entry or he did not attempt ; any scored interactions in his earlier use of the AU.
HACP Binding	
Name	Score
HTTP Messages & Obligations	GetParam, PutParam: Core.Score.Raw: CMI Mandatory, AU Mandatory Core.Score.Max: If *Core.Score.Min* exists, then CMI Mandatory and AU Mandatory, otherwise optional. Core.Score.Min: Optional (CMI and AU)
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.score
API & Obligations	LMSGetValue() and LMSSetValue() : Core.Score.Raw: CMI Mandatory Core.Score.Max: CMI Mandatory. If Core.Score.Min exists AU mandatory, otherwise AU optional. Core.Score.Min: CMI Mandatory, AU Optional
Name Format	"cmi.core.score.raw", "cmi.core.score.max", "cmi.core.score.min" Case sensitive.
Value Format	Single decimal number or empty string "" (for each sub element).
Data type	CMIDecimal (for each sub-element)
Examples	LessonScore = LMSGetValue("cmi.core.score.raw")
	LessonRaw = LMSGetValue("cmi.core.score.raw") LessonMax = LMSGetValue("cmi.core.score.max") LessonMin = LMSGetValue("cmi.core.score.min")
	Success_state = LMSSetValue("cmi.core.score.raw" , ".83")

2.1.11 Core.Session Time

Data Element Name	Core.Session Time
Definition	The amount of time in hours, minutes, and seconds that the student has spent in the AU at the time they leave it. This represents the time from beginning of the session to the end of a single use of the AU.
Usage	Used to keep track of the time spent in an AU for a session. Only the AU sets this element.
CMI Behavior Note	If the AU does not report a value for *Core.Session Time* (or reports an empty string), then the CMI may use its own internal time tracking mechanism to determine *Core.Session Time* (and add to *Core.Total Time)*. The CMI will use the values reported via this element to calculate the **Core.Total Time** (which is a total of all Core.Session Time values reported by a given AU for a given student)
AU Behavior Note	During an AU session, the AU may record **Core.Session Time** multiple times. Should this occur, only the final instance will be recorded for the AU session and added to **Core.Total Time**.
File Binding	
Name	AU Time
Files & Obligations	Finish: AU Mandatory
Name Format	"Time" – case insensitive
Value Format	See Datatype *CMITimespan*
Data type	CMITimespan
Examples	Time = 02:34:05
	TIME = 1002:34:05
	Time = 00:12:23.3
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutParam : AU Mandatory PutParam : CMI Mandatory
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.session_time
Supported API	LMSSetValue()
Obligation	LMSSetValue() : CMI Mandatory
Name Format	"cmi.core.session_time" – case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.core.session_time","0000:12:30")
	LMSSetValue("cmi.core.session_time","03:11:23.45")
	LMSSetValue("cmi.core.session_time","00:18:29")

2.1.12 Core.Total Time

Data Element Name	Core.Total Time
Definition	Accumulated time of all the student sessions of the AU in a given course.
Usage	Used to keep track of the total time spent in every session of a given AU for a given student (enrolled in a given course).
CMI Behavior Notes	CMI must initialize the ***Core.Total Time*** to a (valid time) value of zero the first time the AU is launched and then use the ***Core.Session Time*** values reported by the AU (for each session) to keep a running total.
AU Behavior Notes	
File Binding	
Name	Time
Files & Obligations	Startup : CMI Mandatory
Name Format	"Time" – case insensitive
Value Format	See Datatype
Data type	CMITimespan
Examples	Time = 1002:34:05
	TIME = 02:34:05
	Time = 019:12:23.3
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response) : CMI Mandatory
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.total_time
API & Obligations	LMSGetValue() : CMI Mandatory
Name Format	''cmi.core.total_time" – case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	var x = LMSGetValue("cmi.core.total_time")

2.1.13 Core.Lesson Mode

Data Element Name	Core.Lesson Mode
Definition	Identifies the AU behavior desired after launch. Many AU's have a single "behavior". Some AU's, however, can present different amounts of information, or present information in different sequences, or present information reflecting different training philosophies based on an instructor's or designer's decisions. Designers may enable AU's to behave in a virtually unlimited number of ways. This element supports the communication of three parameters that may result in different AU behaviors.
Usage	This element is set by the CMI. There are three possible values: • **"browse":** The student wants to preview the materials, but not necessarily challenge the AU for an assessment, grade, or evaluation of any kind. The CMI must set *Core.Credit* to "no-credit"

Data Element Name	Core.Lesson Mode
	if a mode of “browse” is used. Also if “browse” mode is used and the current status is “not attempted”, the *Core.Lesson Status* will set to “browsed” by the CMI regardless of what status the AU provides. • **"normal":** This indicates that the AU should behave as designed for a student wanting to get credit for his learning. • **"review":** The student has already seen the material at least once and been graded. The CMI must set *Core.Credit* to “no-credit” if a mode of “review” is used If an unrecognized or unanticipated *Core.Lesson Mode* is received, then the mode the AU designer considers normal is assumed by the AU. (“normal” mode is the default)
CMI Behavior Notes	If *Core.Lesson Mode* is supported in the CMI, the CMI should have a user interface that allows to the student user the ability to select the mode that the AU will be launched with.
AU Behavior Notes	If an AU supports *Core.Lesson Mode*, the AU must return a *Core.Lesson Status* of “browsed” if launched in “browse” mode.
File Binding	
Name	Lesson_Mode
Files & Obligations	Startup : CMI Optional Startup : AU optional
Name Format	“Lesson_Mode” – case insensitive
Value Format	One of the following values: “browse” , “normal”, “review”. All values are case insensitive. Only the first character is significant.
Data type	CMIVocabularyINI:Mode
Examples	Lesson_mode = Normal
	Lesson_MODE = r
	LESSON_MODE = browse
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response) : CMI optional GetParam (response) : AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.core.lesson_mode
API & Obligations	LMSGetValue() : CMI Optional
Name Format	“cmi.core.lesson_mode” – case sensitive
Value Format	One of the following values: “browse” , “normal”, “review”. All values are case sensitive.
Data type	CMIVocabulary:Mode
Examples	var x = LMSGetValue(“cmi.core.lesson_mode”)

2.2 Suspend Data

Data Element Name	Suspend Data

Data Element Name	Suspend Data
Definition	Unique information generated by the AU during previous sessions for a given student that is needed for the current AU session. This data is created by the AU and stored by the CMI to pass back to the AU the next time the AU is run. This element typically used by the AU to retrieve previous state information from the last session (i.e. "restart" or "book-marking" information).
Usage	An AU can set this value at any anytime prior to AU session exit. The AU then could use this information in the next session for that AU.
CMI Behavior Notes	The CMI must set aside a space for this element for each AU in the course(s) for each student. It stores this data and returns it to the AU when it is run again. The CMI shall retain this data as long as the student is enrolled in (or has access to) the course. The CMI is not required to report on this data element.
AU Behavior Notes	
File Binding	
Name	Core_Lesson
Files & Obligations	Finish: CMI Mandatory, AU optional Startup: CMI Mandatory, AU optional
Name Format	"[Core_Lesson]" case insensitive
Value Format	A string of up to 4096 characters in length located in the "[Core_Lesson]" group. The string format is free-form with the following restrictions: • Square brackets "[ ]" are not allowed. • Leading and trailing whitespace (carriage-returns, tabs, spaces) are not included. • Embedded whitespace is allowed and must be included (See Data Type *CMIString4096INI* for more detail)
Data type	CMIString4096INI

Examples	`; In this example the value for`
	`; Core_Lesson starts with "9" and ends with "z".`
	
	`[CORE_lesson]`
	
	`9 00  001010101100110`
	
	`000   001010101100110`
	`0000010101011001l0`
	
	`rtgagfhdfhjkhjkhjk`
	`gl';sdfgl';sdfhgl';sdfhgls';df`
	`z`
	
	`[Core_Vendor]`

	`; This example shows how keyword/value pairs could`
	`; be used in CORE_Lesson.`
	
	`[CORE_Lesson]`

Data Element Name	Suspend Data
	1BookMark = Some book mark data 2BookMark = Some more book mark data 1StateData = Some state data 2StateData = Some more state data.
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (Response): CMI Mandatory, AU optional PutParam: CMI Mandatory, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.suspend_data
API & Obligations	LMSGetvalue(): CMI Mandatory, AU optional LMSSetvalue(): CMI Mandatory, AU optional
Name Format	"cmi.suspend_data" – case sensitive
Value Format	A 4096 character string. The string format is free-form with the following restrictions: • Square brackets "[ ]" are not allowed. • Leading and trailing whitespace (carriage-returns, tabs, spaces) are not included. • Embedded whitespace is allowed and must be included (See Data Type *CMIString4096INI* for more detail)
Data type	CMIString4096INI
Examples	

2.3 Launch Data

Data Element Name	Launch Data
Definition	Unique information specific to an AU that is needed for every use. Without this information, an AU may not execute.
Usage	The data contained in this element is static and will always be the same for a given AU in a given course.
CMI Behavior Notes	A CMI system must allow for administrative users to add *Launch data* for AU's through course structure import. In addition, a CMI system should also allow a user interface for administrative users to directly enter *Launch data* information for a given AU.
AU Behavior Notes	
File Binding	
Name	Core_Vendor
Files & Obligations	Startup: CMI Mandatory, AU Optional
Name Format	"[Core_Vendor]" – case insensitive
Value Format	A string of up to 4096 characters in length located in the

Data Element Name	Launch Data
	"[Core_Vendor]" group. The string format is free-form with the following restrictions: • Square brackets "[ ]" are not allowed. • Leading and trailing whitespace (carriage-returns, tabs, spaces) are not included. • Embedded whitespace is allowed and must be included (See Data Type *CMIString4096INI* for more detail)
Data type	CMIString4096INI

Examples	`; In this example the value for`
	`; Core_Vendor starts with "L" and ends with "8".`
	`; The second "[core_vendor]" is ignored.`
	
	`[CORE_Vendor]`
	
	`Launch stuff …`
	`00110`
	
	`rtgagfhdfhjkhjkhjk`
	`gl';sdfgl';sdfhgl';sdfhgls';df`
	
	`8`
	
	`[Core_Lesson]`

	`; This example shows how keyword/value pairs could`
	`; be used in CORE_VENDOR.`
	
	`[CORE_Vendor]`
	`LaunchParam1 = Some launch stuff`
	`LaunchParam2 = Some more launch stuff`
	`LaunchParam3 = Some launch stuff`

HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (Response): CMI Mandatory, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.launch_data
API & Obligations	LMSGetvalue() : CMI Mandatory, AU Optional
Name Format	"cmi. launch_data" – case sensitive
Value Format	A 4096 character string. The string format is free-form with the following restrictions: • Square brackets "[ ]" are not allowed. • Leading and trailing whitespace (carriage-returns, tabs, spaces) are not included. • Embedded whitespace is allowed and must be included

Data Element Name	Launch Data
	(See Data Type *CMIString4096INI* for more detail)
Data type	CMIString4096INI
Examples	

2.4 Comments From Learner

Data Element Name	Comments From Learner
Definition	This data element contains freeform textual feedback (comments) from a student user during an AU session. The comment (or set of comments) may also have an indication of where or when in the AU it was created.
Usage	A comment (or set of comments) input by the student user of the AU while in an AU session. The AU collects the data for this element and reports it to the CMI system. In the API binding, sequential LMSSetValue() commands create additional comments adding to the string. Comments are not replaced.
CMI Behavior Notes	The CMI system should have a mechanism to report comments (collected using this element) to administrative users.
AU Behavior Notes	The user may have the option of leaving comments at any point in the AU.
File Binding	
Name	AU Comments
Files & Obligations	Finish: CMI Optional, AU Optional
Name Format	"[COMMENTS]" – case insensitive
Value Format	A string of type *CMICommentINI* located in the "[Comments]" group. Multiple comments can be included in this string. (See data type definition for *CMIComment4096INI*) Leading and training whitespace is not included in this string.
Data type	CMIComment4096INI

Examples	`; The string contents start at the "<1>" and`
	`; end at the "<e.4>" (inclusive)`
	`[COMMENTS]`
	
	`<1>The background color is too blue!<1.e><2>The CDU panel has the incorrect 'way points' displayed for this route. <2.e><3>The CDU panel has the incorrect 'way points' displayed for this route. <3.e><4>The CDU panel has the incorrect 'way points' displayed for this route. <e.4>`
	
	`[Evaluation]`

HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding

Data Element Name	Comments From Learner
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.comments
API & Obligations	LMSGetValue() : CMI Optional, AU Optional LMSSetValue() : CMI Optional, AU Optional
Name Format	“cmi.comments”
Value Format	4096 Character string. The format is “free form”. There is no formatting structure to separate multiple comments in an AU session. Square brackets “[]” are not allowed.
Data type	CMIString4096INI
Examples	LMSSetValue(“cmi.evaluation.comments”,”This color is ALL wrong !!”)

2.5 Itemized Comments From Learner

Data Element Name	Itemized Comments From Learner
Definition	An array of comments (freeform textual feedback) made by the student user during an AU session. Each record in this array is made up of the following sub-elements: *Itemized Comments From Learner.Content* *Itemized Comments From Learner.Date* *Itemized Comments From Learner.Location* *Itemized Comments From Learner.Time* Each array record sub-element is described individually in this section. This data element is an alternative to *Comments From Learner.*
Usage	A set of free-form textual comments input by the student user of the AU while in an AU session. The AU collects the data for this element and reports it to the CMI system. Each individual comment is itemized as separate array element with additional sub elements.
CMI Behavior Notes	If a CMI receives data from the AU in both *Itemized Comments From Learner* and *Comments From Learner,* the CMI must save the *Itemized Comments From Learner* and discard the *Comments From Learner* data.
AU Behavior Notes	An AU should only use one method for student comments collection, *Itemized Comments From Learner* or *Comments From Learner.*

2.5.1 Itemized Comments From Learner.Content

Data Element Name	Itemized Comments From Learner.Content
Definition	This data element contains freeform textual feedback (a comment) from the student user during an AU session.
Usage	A comment input by the student user of the AU while in an AU session. The AU collects the data for this element and reports it to the CMI system.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Comment
Files & Obligations	Comments File : CMI Optional, AU Optional
Name Format	Field Name: “Comment” case insensitive
Value Format	A free-form text string with no double quotes (“)s or carriage returns, or control characters allowed.
Data type	CMIString255CSV
Examples	This is ‘comment’ example.
	This is another ‘comment’ example.
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutComments : CMI Optional, AU Optional

Data Element Name	Itemized Comments From Learner.Content
Obligation	
Name Format	Not Applicable
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.evaluation.comments.n.content
API & Obligations	LMSGetValue() : CMI Optional, AU Optional LMSSetValue() : CMI Optional, AU Optional
Name Format	"cmi.evaluation.comments.***n***.content" - case sensitive where ***n*** is the index of the array record .
Value Format	A free-form text string with no double quotes (")s or carriage returns, or control characters allowed.
Data type	CMIString255CSV
Examples	LMSSetValue("cmi.evaluation.comments.2.content","This color is ALL wrong !!")
	var last_comment = LMSGetValue("cmi.evaluation.comments.1.content")

2.5.2 Itemized Comments From Learner.Date

Data Element Name	Itemized Comments From Learner.Date
Definition	The date (including year, month, and day) at which the student user made the comment.
Usage	
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Date
Files & Obligations	Comments File: CMI Optional, AU Optional
Name Format	Field Name: “Date” case insensitive
Value Format	See *CMIDate* data type definition
Data type	CMIDate
Examples	1992/05/20
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutComments: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.5.3 Itemized Comments From Learner.Location

Data Element Name	Itemized Comments From Learner.Location
Definition	Indication of where in the AU that the student user made the comment.
Usage	
CMI Behavior Notes	
AU Behavior Notes	When a developer builds an Assignable Unit, he may give individual sections or frames in the unit their own identifiers or names. These may be used to indicate to which part of the AU the student comment refers.
File Binding	
Name	Location
Files & Obligations	Comments File: CMI Optional, AU Optional
Name Format	Field Name: “Location” case insensitive
Value Format	255 character string without (”)s, carriage returns, or control characters.
Data type	CMIString255CSV
Examples	Frame 13
	Position 4-5
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutComments: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.evaluation.comments.n.location
API & Obligations	LMSGetValue() : CMI Optional, AU Optional LMSSetValue() : CMI Optional, AU Optional
Name Format	“cmi.evaluation.comments.***n***.location” - case sensitive where ***n*** is the index of the array record.
Value Format	255 character string without (”)s, carriage returns, or control characters. (See Data Type *CMIString255CSV* for more detail)
Data type	CMIString255CSV
Examples	

2.5.4 Itemized Comments From Learner.Time

Data Element Name	Itemized Comments From Learner.Time
Definition	A chronological point in a 24-hour clock (i.e. “the time”). Identified in hours, minutes and seconds. The time at which the student makes the comment.
Usage	At the moment that the student user completes a comment, the AU should get the time and record it in this element.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Time
Files & Obligations	Comments File: CMI Optional, AU Optional
Name Format	Field Name: “Time” case insensitive
Value Format	See data type *CMITime* for format description.
Data type	CMITime
Examples	12:05:33 13:06:14.8
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutComments: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.evaluation.comments.***n***.time
API & Obligations	LMSGetValue() : CMI Optional, AU Optional LMSSetValue() : CMI Optional, AU Optional
Name Format	“cmi.evaluation.comments.***n***.time” – case sensitive where ***n*** is the index of the array record.
Value Format	See *CMITime* data type Definition
Data type	CMITime
Examples	

2.6 Comments From LMS

Data Element Name	Comments From LMS
Definition	This element represents comments that would come from the CMI. An example of how this might be used is in the form of instructor comments directed to a particular student (or group of students). These types of comments are directed at the student from the CMI so that the AU may present them to the student when appropriate.
Usage	A comment or set of comments input by an instructor or administrative user using the CMI system. The AU reads this data and displays it to the student.
CMI Behavior Notes	The CMI system may have a mechanism to allow instructors to direct their comments to specific student(s).
AU Behavior Notes	An AU may display comments from the CMI at the beginning of each session.
File Binding	
Name	CMI Comments
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"[COMMENTS]" – case insensitive
Value Format	A string of type *CMICommentINI* located in the "[Comments]" group. Multiple comments can be included in this string. (See data type definition for *CMIComment4096INI*) Leading and training whitespace is not included in this string. Square brackets "[]" are not allowed.
Data type	CMIComment4096INI
Examples	`; The string contents start at the "<1>" and` `; ends at the "<e.4>" (inclusive)` `[COMMENTS]` `<1>Notice that the background color is too blue!<1.e><2>Notice that the CDU panel has the incorrect 'way points' displayed for this route in the Taxi-Out phase. <2.e><3> Notice the CDU panel has the incorrect 'way points' displayed for this route in the Climb Phase. <3.e><4> Notice the CDU panel has the incorrect 'way points' displayed for this route in Cruise. <e.4>`  `[Evaluation]`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.comments_from_lms

Data Element Name	Comments From LMS
API & Obligations	LMSGetValue() : CMI Optional , AU Optional
Name Format	“cmi.comments_from_lms”
Value Format	4096 Character string. The format is “free form”. There is no formatting structure to separate multiple comments in an AU session. Square brackets “[]” are not allowed.
Data type	CMIString4096INI
Examples	var instructor_comments = LMSGetValue (“cmi.comments_from_lms”)

2.7 Evaluation

Data Element Name	Evaluation
Definition	A grouping for a variety of data elements that are provided to the AU by the CMI.
Usage	All data elements in this category are optional.
Membership	Evaluation.Comments_File Evaluation.Course_ID Evaluation.Interactions_File Evaluation.Objective_Status_File Evaluation.Path_File Evaluation.Performance_File

2.7.1 Evaluation.Comments_File

Data Element Name	Evaluation.Comments_File
Definition	A fully qualified file path for the Comments file, which the AU should construct if it is to pass itemized comments back to the CMI system. See (the "File Binding" of) *Itemized Comments from Learner* for the data format of this file. (This data element is only used in the File-Binding)
Usage	CMI determines the location for the Comments File AU writes the Comments file at this location prior to session termination. If this element is not present or set to an empty string, then a comments file will not be written.
CMI Behavior	
AU Behavior	AU writes the Comments file at this location prior to session termination. The AU may append records to this file during different points in an AU's session.
File Binding	
Name	Comments_File
Files & Obligations	Comments_File : CMI Optional, AU Optional
Name Format	"Comments_File" - case insensitive
Value Format	See description for the *CMIFileNameFull* data type
Data type	CMIFileNameFull

Examples	`Comments_File = C:\windows\itemized_comments.txt`
	`COMMENTS_FILE = BB:\somment.cmi`
	`COMMENTS_FILE = C:\ directory with spaces\file with spaces.txt`

HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable

Data Element Name	Evaluation.Comments_File
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.7.2 Evaluation.Course_ID

Data Element Name	Evaluation.Course_ID
Definition	The unique identifier for the course of which the current AU is a part. See COURSE_ID in the course structure.
Usage	The CMI provides the Course ID from the course structure to the AU. The AU used this value of this element to provide Course ID when reporting data out to the following elements: *Itemized Comments From Learner* *Interactions* *Objectives* *Paths*
CMI Behavior Notes	
AU Behavior Notes	AU uses the value of this element to provide Course ID for reporting other data elements in files or messages that require Course ID.
File Binding	
Name	Course_ID
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Course_ID" – case insensitive
Value Format	See data type *CMIIdentifierDevID* for format description. While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*
Data type	CMIIdentifierDevID
Examples	737-300-UAL-RND1
	SCORM-101
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	Getparam(response) : CMI Optional, AU Optional
Obligation	Getparam(response) : CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.7.3 Evaluation.Interactions_File

Data Element Name	Evaluation. Interactions_File
Definition	A fully qualified file path for the Interactions file, which the AU should construct if it is to pass *Interactions* data back to the CMI system. See (the "File Binding" of) *Interactions* for the data format of this file. This data element is only used in the File-Binding.
Usage	The CMI determines the location for the Interactions File The AU writes the Interactions file at this location prior to session termination. If this element is not present or set to an empty string, then the Interactions File will not be written.
CMI Behavior Notes	After the AU session has terminated, the CMI should read and store the contents of this file (and provide a reporting mechanism for administrative users).
AU Behavior Notes	The AU may append records to this file during different points in an AU's session.
File Binding	
Name	Interactions_File
Files & Obligations	Interactions_File: CMI Optional, AU Optional
Name Format	"Interactions_File" - case insensitive
Value Format	See description for the *CMIFileNameFull* data type
Data type	CMIFileNameFull
Examples	`Interactions_File = C:\windows\interact.cmi`
	`Interactions_File = BB:\inter.txt`
	`Interactions_File = C:\ directory with spaces\file with spaces.txt`
HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.7.4 Evaluation.Objective_Status_File

Data Element Name	Evaluation.Objective_Status_File
Definition	A fully qualified file path for the Objective_Status file, which the AU should construct if it is to pass itemized objectives back to the CMI system.

Data Element Name	Evaluation.Objective_Status_File
	See (the "File Binding" of) *Objectives* for the data format of this file. This data element is only used in the File-Binding.
Usage	The CMI determines the location for the Objective_Status File The AU writes the Objective_Status file at this location prior to session termination. If this element is not present or set to an empty string, then the Objective_Status File will not be written.
CMI Behavior Notes	After the AU session has terminated, the CMI should read and store the contents of this file (and provide a reporting mechanism for administrative users).
AU Behavior Notes	The AU may append records to this file during different points in an AU sessions.
File Binding	
Name	Objective_Status_File
Files & Obligations	Objective_Status_File: CMI Optional, AU Optional
Name Format	"Objective_Status_File" - case insensitive
Value Format	See description for the *CMIFileNameFull* data type.
Data type	CMIFileNameFull
Examples	`Objective_Status_File = C:\windows\Objectives status.cmi`
	`Objective_Status_File = X:\objstat.txt`
	`Objective_Status_File = C:\dir1\file with spaces.txt`
HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.7.5 Evaluation.Path_File

Data Element Name	Evaluation.Path_File
Definition	A fully qualified file path for the Path file, which the AU should construct if it is to pass *Path* data back to the CMI system. See (the "File Binding" of) *Path* for the data format of this file. This data element is only used in the File-Binding.
Usage	CMI determines the location for the Path File

Data Element Name	Evaluation.Path_File
	AU writes the Path file at this location prior to session termination. If this element is not present or set to an empty string, then the Path File will not be written
CMI Behavior	After the AU session has terminated, the CMI should read and store the contents of this file (and provide a reporting mechanism for administrative users).
AU Behavior	AU writes the Path file at this location prior to session termination. The AU may append records to this file during different points in an AU sessions.
File Binding	
Name	Path_File
Files & Obligations	Path_File: CMI Optional, AU Optional
Name Format	"Path_File" - case insensitive
Value Format	See description for the *CMIFileNameFull* data type.
Data type	CMIFileNameFull
Examples	`Path_File = C:\windows\path data.txt`
	`PATH_FILE = BB:\path.cmi`
	`PATH_FILE = C:\directory with spaces\file with spaces.txt`
HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.7.6 Evaluation.Performance_File

Data Element Name	Evaluation.Performance_File
Definition	A fully qualified file path for the Performance file, which the AU should construct if it is to pass *Performance* data back to the CMI system. See (the "File Binding" of) *Performance* for the data format of this file. This data element is only used in the File-Binding.
Usage	The CMI determines the location for the Performance File. The AU writes the Path file at this location prior to session termination. If this element is not present or set to an empty string, then the Performance File will not be written
CMI Behavior	After the AU session has terminated, the CMI should read and store the contents of this file (and provide a reporting mechanism for administrative users).

Data Element Name	Evaluation.Performance_File
AU Behavior	AU writes the Performance file at this location prior to session termination. The AU may append records to this file during different points in an AU's session.
File Binding	
Name	Performance_File
Files & Obligations	Performance_File: CMI Optional, AU Optional
Name Format	"Performance_File" - case insensitive
Value Format	See description for the *CMIFileNameFull* data type.
Data type	CMIFileNameFull
Examples	`Performance_File = C:\windows\perf data.txt`
	`Performance_File = BB:\perf.cmi`
	`Performance_File = C:\directory with spaces\file with spaces.txt`
HACP Binding	
Name	Not Applicable
HACP Message(s) & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

2.8 Objectives

Data Element Name	Objectives
Definition	This element contains Information on how the student has performed on objectives related to the AU. The performance may be related to previous sessions in the AU, or to the student user's performance in other AUs (in the same course) related to the same objectives. These objectives are only those associated with the current launching AU, not all the objectives in the course or curriculum. This element is an array. Each record in this array is made up of the following sub-elements: *Objectives.ID* *Objectives.Score* *Objectives.Status* *Objectives.Date* *Objectives.Time* *Objectives.Mastery Time* Each array record sub-element is described individually in this section.
Usage	Information for each individual objective is itemized as separate array record with additional sub elements. The CMI may provide the values for each sub element at AU session start. These values may be determined by completion requirements in the course structure (see *Course.Elements.Completion Requirements*) or prior AU session results. The AU may set the values of each of the provided sub element prior to session end. An objective may be associated with more than one AU in the same course but only those objectives associated with an AU in the course structure will have their data passed to that AU at run time. An AU may set *Objectives.Score* and *Objectives.Score* data for an objective that another AU may read and change. Only following *Objectives* data elements can be transmitted from the CMI to the AU. These elements are as follows: *Objectives.ID* *Objectives.Score* *Objectives.Status* The *Objectives* array is the only array in the communication data model that has elements that both the CMI and the AU can modify. **File & HACP Bindings Usage Specifics** *Objectives.ID, Objectives.Score, Objectives.Status* elements are transmitted to the AU using the Startup File (File binding) or the GetParam Message (HACP binding). In addition (with the File and HACP bindings), these same 3

Data Element Name	Objectives
	data elements have 2 methods that the AU can use to transmit this data to the CMI. They are as follows: **Reporting Method #1** – This method only allows for the reporting of *Objectives.ID, Objectives.Score,* and *Objectives.Status* ▪ HACP Binding: PutParam ▪ File Binding: Finish File **Reporting Method #2 -** – This method allows for the reporting of all sub-elements in *Objectives.* ▪ HACP Binding: PutObjectives ▪ File Binding: Objectives Status File If an AU (with the File or HACP binding) reports this data using *both* methods (and the CMI used supports both methods), then the following rules of precedence apply: File-Based Binding: Method #1 data takes precedence over Method #2 data. HACP Binding: The last HACP message posted (PutObjectives or PutParam) in the AU session takes precedence. **API Binding Usage Specifics** The API binding only has one method for the AU to report all of the sub elements in *Objectives* to the CMI, LMSSetValue(). The CMI system is responsible for initializing all Objectives array data elements during or prior to the AU calling LMSIntialize().
CMI Behavior Notes	
AU Behavior Notes	

2.8.1 Objectives.ID

Data Element Name	Objectives.ID
Definition	A developer defined, course-unique identifier for an objective.
Usage	When an AU sets this data element, it must pass the value of *Course Elements.Developer ID* associated with the objective (that is associated with that AU) in the course structure. When the CMI sets this data element, it must pass the value of *Course Elements.Developer ID* associated with the objective (that is associated with that AU being launched) in the course structure.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Objective_ID

Data Element Name	Objectives.ID	
Files & Obligations	Startup: Finish: Objective Status File:	CMI Optional, AU Optional CMI Mandatory, AU Optional CMI Optional, AU Optional
Name Format	Depends on method used Method #1: "J_ID.***n*"** (case insensitive) where ***n*** number from "1" to "9999" with no leading zeros. Method #2: Not applicable	
Value Format	See description of data type *CMIIdentifierDevID* While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*	
Data type	CMIIdentifierDevID	
Examples	"OBJ-Eng-Start-1"	
	J_ID.1 **=** OBJ-Eng-Start-1	
HACP Binding		
Name	Same as File Binding	
HACP Message(s) & Obligations	GetParam(response): PutParam: PutObjectives:	CMI Optional, AU Optional CMI Optional, AU Optional CMI Optional, AU Optional
Name Format	Same as File Binding	
Value Format	Same as File Binding	
Data type	Same as File Binding	
Examples	Same as File Binding	
	Same as File Binding	
API Binding		
Name	"cmi.objectives.n.id"	
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional	
Name Format	"cmi.objectives.***n***.id" – case sensitive where ***n*** is the (zero-based) array index	
Value Format	See description of data type *CMIIdentifierINI*	
Data type	CMIIdentifierINI	
Examples	LMSSetValue("cmi.objectives.2.id", "OBJ-Eng-Start-1")	
	var objective_var = LMSGetValue("cmi.objectives.2.id")	

2.8.2 Objectives.Score

Data Element Name	Objectives.Score
Definition	Indication of the score obtained by the student after each attempt to master an objective. A maximum and minimum may accompany score. It may have up to three sub-elements: **Raw** This may be an unprocessed or processed indicator of how the student performed with the AU's interactions (related to the objective) experienced. **Max** This is the largest score the student could have with the AU's interactions (related to the objective) experienced.

Data Element Name	Objectives.Score
	Min This is the smallest score that the student could have achieved with the AU's interactions (related to the objective) experienced.
Usage	• If Raw is not accompanied by Max or Min, it may be determined and calculated in any manner that makes sense to the program designer. • If Raw is accompanied by Max or Min, it reflects the performance of the learner relative to the max and min values. • If Max accompanies Raw with no Min, Min is assumed to be "0". • If Min is included then Max must be included. The value of each of the score sub-elements (in relation to one another) must be as follows: Objectives.Score.Max >= Objectives.Score.Raw >= Objectives.Score.Min The AU is responsible for setting this element and the CMI is responsible for providing the value(s) for this element to the AU (in subsequent AU sessions) given the following rules: • CMI must initialize all score *Objectives.Score* elements to an empty string ("") • On subsequent launches of a given AU, The CMI must provide the current value *Objectives.Score* if another AU updated it. • The CMI must update the value of *Objectives.Score* returned by the AU unless *Core.Credit* has a value of "no-credit" for that AU session. If the AU sets *Objectives.Score* multiple times in a session, only the final value is recorded by the CMI (When *Core.Credit* has a value of "credit").
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Score
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI CMI Optional, AU Optional Objective Status File: CMI Optional, AU Optional
Name Format	Depends on method used Method #1: "J_Score.***n***" (case insensitive) where ***n*** number from "1" to "9999" with no leading zeros. Method #2: Not applicable
Value Format	See description of data type *CMIScoreINI*
Data type	CMIScoreINI

Examples	`"75,100,0"`
	`J_score.1 = 75,100,0`
	`J_score.34 = 75`

HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam(response): CMI Optional, AU Optional PutParam: CMI Mandatory, AU Optional PutObjectives: CMI Optional, AU Optional
Name Format	Same as File Binding

Data Element Name	Objectives.Score
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.objectives.n.score.raw" "cmi.objectives.n.score.max" "cmi.objectives.n.score.min"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.objectives.***n***.score.raw" – case sensitive where ***n*** is the (zero-based) array index "cmi.objectives.***n***.score.max" – case sensitive where ***n*** is the (zero-based) array index "cmi.objectives.***n***.score.min" – case sensitive where ***n*** is the (zero-based) array index
Value Format	
Data type	CMIDecimal (for each sub element)
Examples	LMSSetValue("cmi.objectives.2.score.raw", "75") LMSSetValue("cmi.objectives.2.score.max", "75") LMSSetValue("cmi.objectives.2.score.min", "75")
	var objscoreraw = LMSGetValue("cmi.objectives.2.score.raw") var objscoremax = LMSGetValue("cmi.objectives.2.score.max")

2.8.3 Objectives.Status

Data Element Name	Objectives.Status
Definition	Indication of the status of an objective. Six statuses are possible. The CMI system determines this status based on data returned from the AU and other factors. Six status values are possible: • **passed**: A necessary number of objectives in the AU were mastered by the student, and/or the necessary score was achieved. Student is considered to have "completed" the objective and "passed". • **completed**: The student has visited all segments of the AU related to the objective. The student may or may not have passed. The CMI system may make the judgment of whether he passed based upon the score (if one is provided). • **failed**: The objective was not passed. The student experienced some kind of assessment within the AU (specifically related to the objective) but did not demonstrate mastery of the objective. • **incomplete**: The AU was started but not finished. The student did not view all the required elements in the AU related to this objective. • **browsed**: The student launched the AU with a *Core.Lesson Mode* value of "browse" on the initial attempt. In "browse" mode, the student experienced one or more segments of the AU related to the objective. • **not attempted**: The student has not visited any of the segments of the AU related to this objective.".
Usage	Normally, the AU determines *Objectives.Status* and passes it to the CMI. On re-entry into the AU, the CMI passes the previous status returned by the AU. However, the CMI can change the status based on the following

Data Element Name	Objectives.Status
	rules: *1)* If the AU is part of a course that has completion requirements or objectives relationships in its course structure, then the CMI can change the status depending on the rules defined. ***(See Course Structure)*** 2) If there are no completion requirements/objectives relationships rules defined in the course structure, then the CMI cannot change an AU determined objective status. 3) If the CMI sets *Core.Credit* to "no-credit" for the AU session, the CMI is not allowed to change/update *Objectives.Status* based on data set by that AU session. 4) The CMI cannot change a previously (CMI) recorded *Objectives.Status* to "not attempted"
CMI Behavior	The CMI is responsible for setting the initial value to *Objectives.Status* "not attempted".
AU Behavior	
File Binding	
Name	Status.x
In Files	Startup, Finish
Obligation	Startup: CMI Optional, Finish: AU Optional
Name Format	Depends on method used Method #1: "J_Status.***n*"** (case insensitive) where ***n*** number from "1" to "9999" with no leading zeros. Method #2: Not applicable
Value Format	One of the following vocabulary values: "passed" , "failed", "complete", "incomplete", "not attempted", or "browsed". All values are case insensitive. Only the first character is significant.
Data type	CMIVocabularyINI:Status
Examples	J_Status.3 = Passed J_STATUS.1 = c "F"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam(response) : CMI Optional, AU optional PutParam : CMI Optional, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.objectives.n.status
API & Obligations	LMSGetValue() : CMI Optional, AU optional LMSSetValue() : CMI Optional, AU optional
Name Format	"cmi.objectives.***n***.status" - case sensitive where ***n*** is the (zero-based) array index.
Value Format	A specific vocabulary limited to one of the following values: "passed", "completed", "failed" , "incomplete", "browsed", or "not attempted". All values are case sensitive
Data type	CMIVocabulary:Status
Examples	var stat5 = LMSGetValue("cmi.objectives.5.status")

Data Element Name	Objectives.Status
	LMSSetValue("cmi.objectives.8.status", "passed")

2.8.4 Objectives.Date

Data Element Name	Objectives.Date
Definition	The calendar day on which the objective status last updated by the AU.
Usage	This element is set by the AU and read by the CMI
CMI Behavior	
AU Behavior	
File Binding	
Name	Date
In Files	Objective Status File
Obligation	Objective Status File: CMI Optional, AU optional
Name Format	Not Applicable
Value Format	See description of data type *CMIDate*.
Data type	CMIDate
Examples	1997/05/20
HACP Binding	
Name	Same as File Binding
In HACP Message(s)	PutObjectives
Obligation	PutObjectives: CMI Optional, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	Not Applicable
Supported API	Not Applicable
Obligation	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
	Not Applicable

2.8.5 Objectives.Time

Data Element Name	Objectives.Time
Definition	The time of day at which the objective status was last updated by the AU.
Usage	This element is set by the AU and read by the CMI
CMI Behavior	
AU Behavior	
File Binding	
Name	Time
In Files	Objective Status File
Obligation	Objective Status File: CMI Optional, AU optional
Name Format	Not Applicable
Value Format	See description of data type *CMITime*
Data type	CMITime
Examples	12:01:55

Data Element Name	Objectives.Time
	12:01:55.23
HACP Binding	
Name	Same as File Binding
In HACP Message(s)	PutObjectives
Obligation	PutObjectives: CMI Optional, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	Not Applicable
Supported API	Not Applicable
Obligation	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable
	Not Applicable

2.8.6 Objectives.Mastery Time

Data Element Name	Objectives.Mastery Time
Definition	The total time spent by the student on the objective material during the AU session.
Usage	This element is set by the AU and read by the CMI
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Mastery_Time
In Files	Objective Status File
Obligation	Objective Status File: CMI Optional, AU optional
Name Format	Not Applicable
Value Format	See description of data type *CMITimespan*.
Data type	CMITimespan
Examples	12:01:55
	0012:01:55.23
HACP Binding	
Name	Same as File Binding
In HACP Message(s)	PutObjectives
Obligation	PutObjectives: CMI Optional, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	Not Applicable
Supported API	Not Applicable
Obligation	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable

Data Element Name	Objectives.Mastery Time
Data type	Not Applicable
Examples	Not Applicable
	Not Applicable

2.9 Student Data

Data Element Name	Student Data
Definition	A grouping for a variety of data elements.
Usage	All data elements in this category are optional. (See individual member data elements for obligations)
Membership	Student Data.Attempt Number Student Data.Tries Student Data.Tries.Try_Score Student Data.Tries.Try_Status Student Data.Tries.Try_Time Student Data.Mastery Score Student Data.Max Time Allowed Student Data.Time Limit Action Student Data.Tries During Lesson Student Data.Score.n Student Data.Lesson_Status.n

2.9.1 Student Data.Attempt Number

Data Element Name	Student Data.Attempt Number
Definition	The number of previous AU sessions that student has had with the current AU.
Usage	This element is set by the CMI. The CMI must initialize this element to "0". For the student's initial session with the AU, the *Student Data.Attempt Number* will always be "0".
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Attempt_Number
Files & Obligations	Startup : CMI Optional, AU Optional
Name Format	"Attempt_Number" – case insensitive
Value Format	A integer number from 0 to 100 (unsigned)
Data type	CMIInteger
Examples	Attempt_Number = 0
	ATTEMPT_NUMBER = 3
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response) : CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.student_data.attempt_number
API & Obligations	LMSGetValue() : CMI Mandatory

Data Element Name	Student Data.Attempt Number
Name Format	'cmi.student_data.attempt_number" – case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	var x = LMSGetValue("cmi.student_data.attempt_number")

2.9.2 Student Data.Tries

Data Element Name	Student Data.Tries
Definition	This element contains a list of attempts made by the student user to complete the AU's required tasks during an AU session. These attempts may correspond to embedded test(s) or exercise(s) in the AU. This element is an array. Each record in this array is made up of the following sub-elements: *Student Data.Tries.Try_Score* *Student Data.Tries.Try_Status* *Student Data.Tries.Try_Time* Each array record sub-element is described individually in this section.
Usage	The element is set by the AU and stored by the CMI. Data stored from previous AU sessions (in these sub-elements) are not made available to the AU.
CMI Behavior Notes	The CMI should provide a means for administrative users to report data collected from this element.
AU Behavior Notes	

2.9.2.1 Student Data.Tries.Try_Score

Data Element Name	Student Data.Tries.Try_Score
Definition	Indication of the score obtained by the student after each attempt to complete the AU within the current AU session. A maximum and minimum may accompany score. It may have up to three sub-elements: **Raw** This may be an unprocessed or processed indicator of how the student performed with the AU's interactions experienced. **Max** This is the largest score the student could have with the AU's interactions experienced. **Min** This is the smallest score that the student could have achieved with the AU's interactions experienced.
Usage	• If Raw is not accompanied by Max or Min, it may be determined and calculated in any manner that makes sense to the program designer. • If Raw is accompanied by Max or Min, it reflects the performance of the learner relative to the max and min values. • If Max accompanies Raw with no Min, Min is assumed to be "0". • If Min is included then Max must be included. The AU is responsible for setting this element and the CMI is responsible for storing it.
CMI Behavior Notes	
AU Behavior Notes	

Data Element Name	Student Data.Tries.Try_Score
File Binding	
Name	Try_Score
Files & Obligations	Finish: CMI Optional, AU Optional
Name Format	"Try_Score.***n***" (case insensitive CMI Optional, AU Optional) where ***n*** is the array index (a number from "1" to "100" with no leading zeros).
Value Format	See description of data type *CMIScoreINI*
Data type	CMIScoreINI
Examples	`Try_Score.1 = 75,100,0`
	`Try_Score.34 = 75`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_data.tries.n.score.raw" "cmi.student_data.tries.n.score.max" "cmi.student_data.tries.n.score.min"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	case sensitive where ***n*** is the (zero-based) array index "cmi.student_data.tries.***n***.score.raw" "cmi.student_data.tries.***n***.score.max" "cmi.student_data.tries.***n***.score.min
Value Format	
Data type	CMIDecimal (for each sub element)
Examples	LMSSetValue("cmi.student_data.tries.2.score.raw", "75") LMSSetValue("cmi.student_data.tries.2.score.max", "75") LMSSetValue("cmi.student_data.tries.2.score.min", "75")

2.9.2.2 Student Data.Tries.Try_Status

Data Element Name	Student Data.Tries.Try_Status
Definition	The status of the attempt within the AU session.
Usage	Six status values are possible: • **passed**: Mastery of the AU's material was achieved during the attempt. • **completed**: The student has visited all relevant segments of the AU during the attempt. The student may or may not have passed the AU. • **failed**: The student experienced some kind of assessment within the AU but did not demonstrate mastery of the material presented in the attempt. • **incomplete**: The attempt in the AU material was started but not finished. The student did not view all the required elements in the AU to complete the attempt.

Data Element Name	Student Data.Tries.Try_Status
	• **browsed**: The student launched the AU with a *Core.Lesson Mode* value of "browse" on the initial attempt. In "browse" mode, the student experienced one or more segments of the AU related to the attempt. (Note: this status is only possible on the initial attempt in the first AU session) • **not attempted**: The student has not visited any of the segments of the AU related to the attempt
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Try_Status
Files & Obligations	Finish: CMI Optional, AU Optional
Name Format	"Try_Status.***n***" (case insensitive CMI Mandatory, AU Optional) where ***n*** is the array index (a number from "1" to "100" with no leading zeros).
Value Format	See description of data type CMIVocabularyINI:Status
Data type	CMIVocabularyINI:Status
Examples	`Try_Status.1 = passed`
	`Try_Status.23 = C`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_data.tries.n.status
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	case sensitive where ***n*** is the (zero-based) array index: "cmi.student_data.tries.***n***.status"
Value Format	
Data type	CMIVocabulary:Status
Examples	LMSSetValue("cmi.student_data.status", "passed") LMSSetValue("cmi.student_data.tries.2.status", "failed") LMSSetValue("cmi.student_data.tries.2.status", "incomplete")

2.9.2.3 Student Data.Tries.Try_Time

Data Element Name	Student Data.Tries.Try_Time
Definition	The time elapsed during the student user's attempt to complete the AU's required tasks during the AU session.
Usage	The value of this element is only the time spent for a specific "attempt" in the AU session (not the entire AU session). An AU may have multiple "attempts" within a given AU session.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	

Data Element Name	Student Data.Tries.Try_Time
Name	Try_Time
Files & Obligations	Finish: CMI Optional, AU Optional
Name Format	"Try_Time.***n***" (case insensitive CMI Optional, AU Optional) where ***n*** is the array index (a number from "1" to "100" with no leading zeros).
Value Format	See description of data type CMITimespan
Data type	CMITimespan
Examples	`Try_Time.1 = 0000:10:15.01`
	`Try_Time.23 = 00:11:12`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_data.tries.***n***.time
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	case sensitive where ***n*** is the (zero-based) array index cmi.student_data.tries.***n***.time
Value Format	
Data type	CMITimespan
Examples	LMSSetValue("cmi.student_data.tries.2.time","00:00:30") LMSSetValue("cmi.student_data.tries.2.time","00:01:30.45") LMSSetValue("cmi.student_data.tries.2.time","00:02:30.01")

2.9.3 Student Data.Mastery Score

Data Element Name	Student Data.Mastery Score
Definition	This element defines a score level at which an AU is considered mastered.
Usage	This element is set by the CMI. When the *Core.Score.Raw* returned by an AU session is greater than or equal to the *Student Data.Mastery Score*, then the student is considered to have passed, or mastered the content. If the value of *Core.Score.Raw* returned is less than *Student Data.Mastery Score* then the student is considered to have failed the content. If a value is present for both *Student Data.Mastery Score* and *Core.Score.Raw*, the CMI must change the *Core.Lesson Status* to "passed" or "failed" accordingly for that AU. (unless *Core.Credit* is set to "no-credit" or completion requirements rules in the course structure have additional mastery requirements) If the AU does not return a value for *Core.Score.Raw,* then the student is

Data Element Name	Student Data.Mastery Score
	considered to have not performed the portion of the AU's content that was the scored activity and the CMI does not modify *Core.Lesson Status* based on *Student Data.Mastery Score.* The value for *Student Data.Mastery Score* is provided by the CMI.
CMI Behavior Notes	
AU Behavior Notes	Since this element is optional, it is recommended that an AU have a default mastery score internally defined in the event that the CMI does not provide .
File Binding	
Name	Mastery_Score
Files & Obligations	Startup : CMI Optional, AU Optional
Name Format	"Mastery_Score" - case insensitive
Value Format	Empty ("") string or decimal number. See description of data type *CMIDecimal*
Data type	CMIDecimal
Examples	`Mastery_Score = 75`
	`Mastery_Score = 75.6`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_data.mastery_score
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_data.mastery_score" - case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	var mc = LMSGetValue("cmi.student_data.mastery_score")

2.9.4 Student Data.Max Time Allowed

Data Element Name	Student Data.Max Time Allowed
Definition	The amount of time the student is allowed to have in the current AU session. See *Student Data.Max Time Limit Action* for the AU's expected response to exceeding this time limit.
Usage	This element is set by the CMI.
CMI Behavior Notes	
AU Behavior Notes	See *Student Data.Max Time Limit Action* .
File Binding	
Name	Max_Time_Allowed
Files & Obligations	Startup : CMI Optional, AU Optional

Data Element Name	Student Data.Max Time Allowed
Name Format	"Max_Time_Allowed" - case insensitive
Value Format	See description of data type *CMITimespan*
Data type	CMITimespan
Examples	`Max_Time_Allowed = 0000:10:00`
	`Max_Time_Allowed = 00:20:00.34`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_data.max_time_allowed
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_data.max_time_allowed" -case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	var mc = LMSGetValue("cmi.student_data.max_time_allowed")

2.9.5 Student Data.Time Limit Action

Data Element Name	Student Data.Max Time Limit Action
Definition	Indicates to the AU what actions to perform when the *Student Data.Max Time Allowed* time limit is exceeded for the AU session.
Usage	There are four possible values for this data element: • **Exit**, **Message –** The AU displays a message to the student (indicating that the time limit was exceeded) and then exits the AU session. • **Exit**, **No Message -** The AU session exits without displaying a message to the student • **Continue**, **Message -** The AU session continues but AU displays a message to the student (indicating that the time limit) was exceeded. • **Continue**, **No Message -** The AU session continues without displaying a message to the student (i.e. the AU ignores the time limit being exceeded)
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Time_Limit_Action
Files & Obligations	Startup : CMI Optional, AU Optional
Name Format	"Max_Time_Allowed" - case insensitive
Value Format	See description of data type *CMIVocabularyINI:Time Limit Action*
Data type	CMIVocabularyINI:Time Limit Action

Data Element Name	Student Data.Max Time Limit Action
Examples	`Time_Limit_Action = Continue, Message`
	`Time_Limit_Action = E, n`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_data.time_limit_action
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_data.time_limit_action" - case sensitive
Value Format	See description of data type *CMIVocabulary:Time Limit Action*
Data type	CMIVocabulary:Time Limit Action
Examples	var mc = LMSGetValue("cmi.student_data.time_limit_action")

2.9.6 Student Data.Tries During Lesson

Data Element Name	Student Data.Tries During Lesson
Definition	The number of attempts made by the student user to complete the AU's required tasks during an AU session. These attempts may correspond to embedded test(s) or exercise(s) in the AU. The value of this element directly corresponds to the number of array records in the *Student Data.Tries*.
Usage	This element is set by the AU.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Tries_During_Lesson
Files & Obligations	Finish: CMI Optional, AU Optional
Name Format	"Tries_During_Lesson" - case insensitive
Value Format	
Data type	CMIInteger
Examples	`Tries_During_Lesson = 1`
	`TRIES_DURING_LESSON = 5`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding

Data Element Name	Student Data.Tries During Lesson
API Binding	
Name	cmi.student_data.tries_during_lesson
API & Obligations	LMSSetValue(): CMI Optional, AU Optional LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_data.tries_during_lesson -case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.student_data.tries_during_lesson")

2.9.7 Student Data.Sessions Journal

Data Element Name	Student Data.Session Journal
Definition	This element contains score and status data from previous AU sessions. It is intended to provide a session history so that the AU designer may vary the current AU session presentation based on student user performance in past sessions. This element is an array. Each record in this array is made up of the following sub-elements: Student Data.Session Journal.Lesson Score Student Data.Session Journal.Lesson Status

2.9.7.1 Student Data.Sessions Journal.Lesson Score

Data Element Name	Student Data.Session Journal.Lesson Score
Definition	This data element contains the value of *Core.Score* returned from a previous AU session indicated by the array index. See *2.1.10 Core.Score* for a detailed description.
Usage	The CMI sets the value of this element based on data returned from prior AU sessions. It is read only to the AU. See *2.1.10 Core.Score* for more information on score usage.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	score.***n***
Files & Obligations	Startup: Core.Score.Raw: CMI and AU Optional Core.Score.Max: If *Core.Score.Min* exists, then CMI and AU Mandatory, otherwise optional. Core.Score.Min: CMI and AU Optional
Name Format	"score.***n*"** (case insensitive) where ***n*** is a number from "1" to "9999" with no leading zeros. The index value of ***"n "*** corresponds directly to the ordinal number of previous sessions (i.e. "1" is the value for the first AU session, "2" is the second AU session, etc.)

Data Element Name	Student Data.Session Journal.Lesson Score
Value Format	See *2.1.10 Core.Score*
Data type	See *2.1.10 Core.Score*
Examples	`score.1 = 75`
	`score.2 = 75.6`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_data.attempt_records.***n.***score.raw cmi.student_data.attempt_records.***n.***score.min cmi.student_data.attempt_records.***n.***score.max
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_data.lesson_status.***n***" - case sensitive where ***n*** is the (zero-based) array index. The index value of ***"n "*** corresponds to the ordinal number of previous sessions minus 1. (i.e. "0" is the value for the first AU session, "1" is the second AU session, etc.)
Value Format	See *2.1.10 Core.Score*
Data type	See *2.1.10 Core.Score*
Examples	session2_score_raw = LMSGetValue("attempt_records.1.score.raw") session2_score_min = LMSGetValue("attempt_records.1.score.min") session2_score_max = LMSGetValue("attempt_records.1.score.max")

2.9.7.2 Student Data.Sessions Journal.Lesson Status

Data Element Name	Student Data.Session Journal.Lesson Status
Definition	This data element contains the value of *Core.Lesson Status* returned from an previous AU session indicated by the array index. See *2.1.6 Core.Lesson Status* for more information.
Usage	The CMI sets the value of this element based on data returned from prior AU sessions. It is read only to the AU. See 2.1.6 Core.Lesson Status for more information on usage.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	lesson_status.***n***
Files & Obligations	Startup : CMI Optional, AU Optional

Data Element Name	Student Data.Session Journal.Lesson Status
Name Format	"lesson_status. ***n*"** (case insensitive) where ***n*** is a number from "1" to "9999" with no leading zeros. The index value of ***"n "*** corresponds directly to the ordinal number of previous sessions (i.e. "1" is the value for the first AU session, "2" is the second AU session, etc.)
Value Format	
Data type	
Examples	`lesson_status.1 = Incomplete`
	`LESSON_STATUS.2 = Passed,L`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_data.lesson_status.***n***
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_data.lesson_status.***n***" - case sensitive where ***n*** is the (zero-based) array index. The index value of ***"n "*** corresponds to the ordinal number of previous sessions minus 1. (i.e. "0" is the value for the first AU session, "1" is the second AU session, etc.)
Value Format	See *2.1.6 Core.Lesson Status*
Data type	See *2.1.6 Core.Lesson Status*
Examples	var session2status = LMSGetValue("cmi.student_data.lesson_status.1")

2.10 Student Preference

Data Element Name	Student Preference
Definition	A grouping for a variety of data elements relating to preferences that a given student user has set for a given course.
Usage	All data elements in this group are set by the AU (by some user interface in the AU presented the student user to pick the course preferences). For a given student, all data elements in this group are shared with all of other AU's in a given course. So a student may set a preference data element (such as *Student Preference.Audio*) in one AU and that preference value will persist (throughout the course) until changed by the student in subsequent AU's/AU sessions. To provide this persistence, the CMI must store/update the data elements in this group at the end of each AU session and pass them to any other AU in a given course (for a given student). This data is retained for the duration of the student's enrollment in a course. Some data elements in this group do not have controlled vocabularies, so some preferences set by one AU may not "translate" among AU's from different designers. Regardless, the values for preferences still persist until changed (even if some AU's do not understand them). All data elements in this category are optional. (See individual member data elements for obligations).
Membership	Student Preference.Audio Student Preference.Language Student Preference.Lesson Type Student Preference.Speed Student Preference.Text Student Preference.Text Color Student Preference.Text Location Student Preference.Text Size Student Preference.Video Student Preference.Windows

2.10.1 Student Preference.Audio

Data Element Name	Student Preference.Audio
Definition	This element determines the student preference for playing audio and audio volume during AU presentations.
Usage	The possible states for this element are as follows: • **On** – Play audio at the indicated volume (an integer value of 1 to 100. 1 being the lowest volume, 100 being the highest)

Data Element Name	Student Preference.Audio
	• **Off –** No audio is played (an integer value of -1) • **Default –** Play audio based on AU's internal defaults (an integer value of 0). If no value is available (or this element is not supported) and AU should assume "0".
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Audio
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Audio**"** - case insensitive
Value Format	An integer value from –1 to 100. Values are as follows: -1 : Off – No audio is played 0 : Default – Play audio based on AU's internal defaults 1 to 100 : On - Play audio at the indicated volume. (unsigned)
Data type	CMISinteger
Examples	`; Audio is off` `Audio = -1`
	`; Audio is set to maximum possible volume` `Audio = 100`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.audio
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.audio" - case sensitive
Value Format	Same as File Binding
Data type	CMISinteger
Examples	var mc = LMSGetValue("cmi.student_preference.audio")
	/* set audio off */ LMSSetValue("cmi.student_preference.audio","-1")
	/* set audio on and at half volume*/ LMSSetValue("cmi.student_preference.audio","50")

2.10.2 Student Preference.Language

Data Element Name	Student Preference.Language
Definition	For AU's with multi-lingual capability, this element identifies which

Data Element Name	Student Preference.Language
	language should be used to deliver instruction based on the student's selected preference.
Usage	This element can be set (by the AU) to any string that represents a language. There is no preset vocabulary for language values. If a *Student Preference.Language* value is not recognized by the AU, it should then use its own internal default for language delivery. Because of there is no preset vocabulary for *Student Preference.Language* values, this element is AU implementation specific. AU's from different designers in the same course may not be able to interpret language values.
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Language
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Language" - case insensitive
Value Format	A 255 character string representing a language. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	Language = French
	Language = English
	Language = Chinese
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.language
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.language" - case sensitive
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var lang = LMSGetValue("cmi.student_preference.language")
	LMSSetValue("cmi.student_preference.language")","French")

2.10.3 Student Preference.Lesson Type

Data Element Name	Student Preference.Lesson Type

Data Element Name	Student Preference.Lesson Type
Definition	This data element specifies the (AU designer specific) "type" of AU that made the last updates to other *Student Preference* data elements. The purpose for this element is to communicate to other AU's which "type" of AU updated the *Student Preference* data elements last, since some *Student Preference* data elements set in one type of AU may be meaningless when applied to another type of AU. The reason for this limitation is that some *Student Preference* data elements in this specification are not defined with controlled vocabularies. These (implementation specific) *Student Preference* data elements are as follows: Student Preference.Language Student Preference.Text Color Student Preference.Text Location Student Preference.Text Size Student Preference.Video Student Preference.Windows
Usage	This element is set by the AU when changing the values of any of the following *Student Preference* data elements: Student Preference.Language Student Preference.Text Color Student Preference.Text Location Student Preference.Text Size Student Preference.Video Student Preference.Windows The CMI passes this element to all AU's in a course. After the value for this element is updated by a given AU, the CMI passes the new value to all subsequent AU's and AU sessions for a given student in a given course. This value for this data element is AU designer specific.
CMI Behavior Notes	
AU Behavior Notes	A designer creating large numbers of AU's should make them as homogenous as possible with regards to use of *Student Preference* data elements (i.e. use the same *Student Preference.Lesson Type* whenever possible)
File Binding	
Name	Lesson_Type
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Lesson_Type" - case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	Lesson_Type = Airbus-A320-Adopt-PPT Lesson_Type = Boeing-777-Authorware-5 Lesson_Type = NWA-Flash-Flight
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional

Data Element Name	Student Preference.Lesson Type
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.lesson_type
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.lesson_type" - case sensitive
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var Lessontype = LMSGetValue("cmi.student_preference.lesson_type")
	LMSSetValue("cmi.student_preference.lesson_type")","Airbus-A320-Adopt-PPT")

2.10.4 Student Preference.Speed

Data Element Name	Student Preference.Speed
Definition	The student's preferred playback speed for AU materials.
Usage	The allowed values for this element is an integer number from -100 to 100 where: • The value of "–100" is slowest playback speed. The AU plays back at the slowest speed possible, • The value of "0" is a "no-change status". The AU defaults to its normal playback speed. • The value of "100" is the fastest playback speed. The AU plays back at the fastest speed possible,
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Speed
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Speed" - case insensitive
Value Format	An integer value from –100 to 100. Values are as follows: -1 to -100 : Slower speeds 0 : Default – Speed based on AU's internal defaults 1 to 100 : Faster speeds
Data type	CMISinteger
Examples	`; Speed is set to slowest possible pace` `Speed = -100`
	`; Speed is set to fasted possible pace` `Speed = 100`
HACP Binding	
Name	Same as File Binding

Data Element Name	Student Preference.Speed
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.speed
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.speed" - case sensitive
Value Format	Same as File Binding
Data type	CMISinteger
Examples	var mc = LMSGetValue("cmi.student_preference.speed")
	LMSSetValue("cmi.student_preference.speed","-1")
	LMSSetValue("cmi.student_preference.speed","50")

2.10.5 Student Preference.Text

Data Element Name	Student Preference.Text
Definition	This element identifies whether the audio narration text appears in the AU's presentation.
Usage	This element is an integer with 3 possible values (–1, 0, and 1) where these values have the following meaning: -1 Is text off. Narration text is not displayed by the AU 0 Is no change to text setting, the AU uses its default value. 1 Is text on. The AU displays narration text to the student
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Text
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Text" - case insensitive
Value Format	An integer with 3 possible values (–1, 0, and 1) see ***usage***
Data type	CMISinteger
Examples	`Text = -1`
	`Text = 1`
HACP Binding	
Name	Same as File Binding

Data Element Name	Student Preference.Text	
HACP Message(s) & Obligations	GetParam (response): PutParam:	CMI Optional, AU Optional CMI Optional, AU Optional
Name Format	Same as File Binding	
Value Format	Same as File Binding	
Data type	Same as File Binding	
Examples	Same as File Binding	
	Same as File Binding	
API Binding		
Name	cmi.student_preference.text	
API & Obligations	LMSGetValue(): LMSSetValue():	CMI Optional, AU Optional CMI Optional, AU Optional
Name Format	"cmi.student_preference.text" - case sensitive	
Value Format	Same as File Binding	
Data type	CMISinteger	
Examples	var textpref = LMSGetValue("cmi.student_preference.text")	
	LMSSetValue("cmi.student_preference.text","-1")	
	LMSSetValue("cmi.student_preference.text","0")	

2.10.6 Student Preference.Text Color

Data Element Name	Student Preference.Text Color
Definition	This element stores student preferences for text color and text background in the AU presentation.
Usage	Format of data in this element is AU implementation specific.
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Text_Color
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Text_Color" - case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Text_Color = R23,B34,G465`
	`Text_Color =`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): PutParam:	CMI Optional, AU Optional CMI Optional, AU Optional
Name Format	Same as File Binding	
Value Format	Same as File Binding	
Data type	Same as File Binding	
Examples	Same as File Binding	
	Same as File Binding	
API Binding		

Data Element Name	Student Preference.Text Color
Name	cmi.student_preference.text_color
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.text_color" - case sensitive
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var textcolorpref = LMSGetValue("cmi.student_preference.text_color")
	LMSSetValue("cmi.student_preference.text_color","green")
	LMSSetValue("cmi.student_preference.text_color","blue")

2.10.7 Student Preference.Text Location

Data Element Name	Student Preference.Text Location
Definition	This element stores student preferences for location of narration text in the AU presentation.
Usage	Format of data in this element is AU implementation specific.
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Text Location
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Text_Location" – case insensitive
Value Format	255 Character String
Data type	CMIString255INI
Examples	`Text_Location = Lower-right`
	`Text_Location = 123, 240`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.text_location
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.text_location" - case sensitive
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var textcolorpref = LMSGetValue("cmi.student_preference.text_location")
	LMSSetValue("cmi.student_preference.text_location","lower-right")

Data Element Name	Student Preference.Text Location
	LMSSetValue("cmi.student_preference.text_location","234,56")

2.10.8 Student Preference.Text Size

Data Element Name	Student Preference.Text Size
Definition	This element stores student preferences for the size of displayed text in the AU presentation.
Usage	Format of data in this element is AU implementation specific.
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Text Size
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Text_Size" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Text Size = 124%`
	`Text_Size = Large`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.text_size
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.text_size" - case sensitive
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var textcolorpref = LMSGetValue("cmi.student_preference.text_size")
	LMSSetValue("cmi.student_preference.text_size","124%")
	LMSSetValue("cmi.student_preference.text_size","Large")

2.10.9 Student Preference.Video

Data Element Name	Student Preference.Text Video
Definition	This element stores student preferences for display/control properties for video presented in the AU.
Usage	Format of data in this element is AU implementation specific.
CMI Behavior Notes	

Data Element Name	Student Preference.Text Video
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Video
Files & Obligation	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Video" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Video = 124, 56 – controls on`
	`Video = normal size`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.video
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.video" - case sensitive
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var textcolorpref = LMSGetValue("cmi.student_preference.video")
	LMSSetValue("cmi.student_preference.video","124, 56 – controls on")
	LMSSetValue("cmi.student_preference.video","normal size")

2.10.10 Student Preference.Windows

Data Element Name	Student Preference.Windows
Definition	This element stores student preferences for display properties of presentation window(s) used by the AU. This element is an array. Each array record represents properties for a single display window. There is only a single value per record.
Usage	An AU may use multiple display windows. Format of data in this element is AU implementation specific.
CMI Behavior Notes	
AU Behavior Notes	This data element is set by the AU, usually by some user interface in the AU that presents the student with user selectable preference options. It is recommended that the AU does not change this element without student prompting.
File Binding	
Name	Window.1

Data Element Name	Student Preference.Windows
Files & Obligations	Startup: CMI Optional, AU Optional Finish: CMI Optional, AU Optional
Name Format	"Window.***n***" – case insensitive where ***n*** is the array index.
Value Format	255 Character String. Format of data is AU implementation specific. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Window.2 = 124, 56 – controls on`
	`window.1 = normal size`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional PutParam: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	cmi.student_preference.windows.n
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_preference.windows.***n***" - case sensitive where ***n*** is the (zero-based) array index.
Value Format	Same as File Binding
Data type	CMIString255INI
Examples	var textcolorpref = LMSGetValue("cmi.student_preference.windows.0")
	LMSSetValue("cmi.student_preference.widnows.2","124, 56 – controls on")
	LMSSetValue("cmi.student_preference.windows.3","normal size")

2.11 Interactions

Data Element Name	Interactions
Definition	In this context, an "interaction" is a recognized and recordable input from the student to the computer. All of the items in this group are related to a recognized and recordable input from the student. The purpose of the element is to collect detailed information on each interaction measured in an AU session. This element is an array. Each record in this array corresponds to a single interaction in the current AU session. Each record is made up of the following sub-elements: *Interactions.ID* *Interactions.Objectives* *Interactions.Date* *Interactions.Time* *Interactions.Type* *Interactions.Correct Responses* *Interactions.Weighting* *Interactions.Student Response* *Interactions.Result* *Interactions.Latency* Each array record sub-element is described individually in this section
Usage	The AU sets all data elements in this group. The CMI stores and retains this data for reporting purposes.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	

2.11.1 Interactions.ID

Data Element Name	Interactions.ID
Definition	A developer defined, unique identifier for a specific "interaction" within an AU.
Usage	This element is internally determined and is set by the AU.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Interaction_ID
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Not applicable
Value Format	See data type *CMIIdentifierDevID* for format description. While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*
Data type	CMIIdentifier*DevID*
Examples	"Int-Eng-Start-1"
	"XYZ-1230-122"

Data Element Name	Interactions.ID
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.id"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Obligation	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.interactions.***n***.id" – case sensitive where ***n*** is the (zero-based) array index
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.objectives.2.id", "int-Eng-Start-1")
	var inter_var = LMSGetValue("cmi.interactions.2.id")

2.11.2 Interactions.Objectives

Data Element Name	Interactions.Objectives
Definition	The identifier(s) of the objectives associated with the *Interactions* record.
Usage	This element is internally determined and set by the AU. The objective ID's used must match those associated with objectives in the course structure. In the API binding, this element is an array and can contain multiple objective ID's associated with the *Interactions* record. For HACP and File bindings there can only be a single objective ID in this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Objective_ID
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Objective_ID" case insensitve
Value Format	See data type *CMIIndentifierDevID* for description While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*
Data type	*CMIIdentifierDevID.*
Examples	"Int-Eng-Start-1"
	"XYZ-1230-122"

Data Element Name	Interactions.Objectives
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.id.objectives.n.id"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.interactions.***n***.id.objectives.***n1***.id" case sensitive where ***n*** is the (zero-based) array index for the interaction record and ***n1*** is the sub (zero-based) array index for the objectives associated with the interaction record,
Value Format	Same as File Binding
Data type	*CMIIdentifierDevID* (for each element)
Examples	LMSSetValue("cmi.interactions.2.id.objectives.1.id", "int-Eng-Start-1")
	var iObj_var = LMSGetValue("cmi.interactions.3.id.objectives.2.id")

2.11.3 Interactions.Date

Data Element Name	Interactions.Date
Definition	The calendar day on which the *Interactions* array record was recorded by the AU.
Usage	This element is set by the AU.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Date
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Date" case insensitve
Value Format	See description of data type *CMIDate*
Data type	CMIDate
Examples	"1999/03/22"
	"2001/09/11"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding

Data Element Name	Interactions.Date
API Binding	
Name	"cmi.interactions.n.date"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.interactions.***n***.date" case sensitive where ***n*** is the (zero-based) array index for the interaction record
Value Format	See description of data type *CMIDate*
Data type	CMIDate
Examples	LMSSetValue("cmi.interactions.2.date", "2002/05/23")
	var iObj_var = LMSGetValue("cmi.interactions.3.date")

2.11.4 Interactions.Time

Data Element Name	Interactions.Time
Definition	The time of day on which the *Interactions* array record was recorded by the AU.
Usage	This element is set by the AU.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Interactions Time
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Time" case insensitve
Value Format	See description of data type *CMITime*
Data type	CMITime
Examples	"12:01:02"
	"13:05:56.23"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.time"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.interactions.***n***.time" case sensitive where ***n*** is the (zero-based) array index for the interaction record.
Value Format	Same as File Binding
Data type	Same as File Binding

Data Element Name	Interactions.Time
Examples	LMSSetValue("cmi.interactions.2.time", "12:01:03")
	var iTime_var = LMSGetValue("cmi.interactions.3.time")

2.11.5 Interactions.Type

Data Element Name	Interactions.Type
Definition	The "type" of interaction that was recorded. The type of interaction determines how the *Interactions.Student Response* and *Interactions.Correct* Response will be interpreted.
Usage	The AU sets this element. The seven possible values are defined. **True/False** A question with only two possible responses (true or false). There is only one possible correct response for this type of interaction. **Multiple Choice** A question with a limited number of predefined responses from which the student may select. Each response is numbered or lettered. One or more responses may be correct for this type of interaction. **Fill in the Blank** A question with a simple one or few-word answer. The answer/response is not predefined, but must be created by the student (as opposed to selected). There is only one possible correct response for this type of interaction. **Matching** A question with one or two sets (or lists) of items. Two or more of the members of these sets are related. Answering the question requires finding and matching related members in different sets (or lists). One or more responses may be correct for this type of interaction. **Simple Performance** A performance question is in some ways similar to multiple choice and sequencing questions. However, instead of selecting a written answer, the student must perform a task or action. This step in the task or action when input to the computer may have two parts. They are translated and stored as an alpha-numeric codes or tokens. One or more responses may be correct for this type of interaction. **Sequencing** In a sequencing question, the student is required to identify a logical order for the members of a set or list. For instance, he or she may be asked to place a series of events in chronological order. Or the student may be asked to rank a group of items by the order of their importance. One or more responses may be correct for this type of interaction. **Likert** A Likert question offers the student a group of alternatives on a continuum. The response is generally based on the student's opinion or attitude. Typical scales are as follows: • FROM Strongly agree TO Strongly disagree

Data Element Name	Interactions.Type
	• FROM Way too much TO Way too little • FROM Understand completely TO Do not understand at all There is no "correct answer" for likert type interactions. There is only one response. **Numeric** A numeric value with or without a decimal point is required in answering the question. The correct answer may be a single number within a range of numbers.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Type_Interaction
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Type_Interaction" case insensitve
Value Format	See description of data type *CMIVocabularyINI:Interaction*
Data type	CMIVocabularyINI:Interaction
Examples	"Likert"
	"M"
	"Fill-in"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.type"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	Case sensitive where ***n*** is the (zero-based) array index for the *Interactions* record: "cmi.interactions.***n***.type"
Value Format	See description of data type *CMIVocabularyINI:Interaction*
Data type	CMIVocabularyINI:Interaction
Examples	LMSSetValue("cmi.interactions.2.type", "likert")
	var iType_var = LMSGetValue("cmi.interactions.3.type")

2.11.6 Interactions.Correct Responses

Data Element Name	Interactions.Correct Responses
Definition	All possible correct responses to the interaction. There may be more than one correct response depending upon the interaction "type".
Usage	The AU sets this element. The format of this element is determined by type indicated in *Interactions.Type.* (See *Interactions Type* for Type definitions)

Data Element Name	Interactions.Correct Responses
	• Type **Likert** has no "correct response". The element is left blank for interactions of type "Likert". • The following types can have multiple possible correct responses: **Multiple Choice** **Matching** • The following types can have only one possible correct response: **Fill in the Blank** **Simple Performance** **Sequencing** **Numeric** In the API binding, this element is an array with one record for each possible correct response. For HACP and File bindings this element is a single value with delimiters for multiple correct responses.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Correct_Response
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Correct_Response" case insensitive
Value Format	See data type *CMIFeedbackCSV* sub types for description of correct formats based on interaction type.
Data type	CMIFeedbackCSV
Examples	"Likert"
	"M"
	"Fill-in"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.correct_reponses.n.pattern"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	Case sensitive where ***n*** is the (zero-based) array index for the *Interactions* record and ***n1*** is the index for the correct response(s): "cmi.interactions.***n***.correct_reponses.***n1***.pattern"
Value Format	See data type *CMIFeedbackCSV* sub types for description of correct formats based on interaction type.
Data type	CMIFeedbackCSV
Examples	LMSSetValue("cmi.interactions.2.correct_reponses.1.pattern", "{1.b,2.c}")
	Cor1 = LMSGetValue("cmi.interactions.3.correct_reponses.1.pattern")

2.11.7 Interactions.Weighting

Data Element Name	Interactions.Weighting
Definition	The weighted value of the interaction. The weighting is a factor, which is used to identify the relative importance of one interaction compared to another.
Usage	The AU sets this element. If all interactions are equal in importance, then each interaction has the same weight.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	An AU's weighting of interactions may reflect their impact on the score for an AU session. A weight of 0 indicates that the AU may not count the interaction in the weighted final score.
File Binding	
Name	Weighting
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Weighting" case insensitive
Value Format	See data type CMIDecimal.
Data type	CMIDecimal
Examples	1
	2.5
	3
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.weighting"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	Case sensitive where ***n*** is the (zero-based) array index for the *Interactions* record: "cmi.interactions.***n***.weighting"
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.interactions.2.weighting", "2")
	Weight1 = LMSGetValue("cmi.interactions.3. weighting")

2.11.8 Interactions.Student Response

Data Element Name	Interactions.Student Response
Definition	The student user response to the interaction.
Usage	The AU sets this element. The format of this element is determined by type indicated in *Interactions.Type.* (See *Interactions Type* for Type definitions)
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	

Data Element Name	Interactions.Student Response
File Binding	
Name	Student_Response
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Student_Response" case insensitive
Value Format	See data type *CMIFeedbackCSV* sub types for description of correct formats based on interaction type.
Data type	CMIFeedbackCSV
Examples	"{1.a,2.b,3.c}"
	"2.a" --
	"a"
	"This is a response to a fill-in-the-blank question"
	34
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.student_reponse"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	Case sensitive where ***n*** is the (zero-based) array index for the *Interactions* record: "cmi.interactions.***n***.student_reponse"
Value Format	See data type *CMIFeedbackCSV* sub types for description of correct formats based on interaction type.
Data type	CMIFeedbackCSV
Examples	LMSSetValue("cmi.interactions.2.student_reponse.1", "{1.b,2.c}")
	StudResp1 = LMSGetValue("cmi.interactions.3. student_reponse")

2.11.9 Interactions.Result

Data Element Name	Interactions.Result
Definition	Judgment of the acceptability of the student response in the interaction.
Usage	The AU sets this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Result
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Result" case insensitive
Value Format	See data type *CMIVocabularyINI:Result* for description of data formating.
Data type	CMIVocabularyINI:Result

Data Element Name	Interactions.Result
Examples	"C"
	"wrong"
	"Unanticipated"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.result"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	Case sensitive where ***n*** is the (zero-based) array index for the *Interactions* record: "cmi.interactions.***n***.result"
Value Format	See data type *CMIVocabularyINI:Result* for description of data formating.
Data type	CMIVocabulary:Result
Examples	LMSSetValue("cmi.interactions.2.result", "correct")
	res1 = LMSGetValue("cmi.interactions.3. result")

2.11.10 Interactions.Latency

Data Element Name	Interactions.Latency
Definition	The time from the presentation of the Interaction stimulus to the completion of the measurable response in the AU.
Usage	The AU sets this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Latency
Files & Obligations	Interactions File: CMI Optional, AU Optional
Name Format	Field Name: "Latency" case insensitive
Value Format	See data type *CMITimespan* for description.
Data type	CMITimespan
Examples	"00:00:03"
	"00:01:03.50"
	"0000:03:03.1"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutInteractions: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding

Data Element Name	Interactions.Latency
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.interactions.n.latency"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	Case sensitive where ***n*** is the (zero-based) array index for the *Interactions* record: "cmi.interactions.***n***.latency"
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.interactions.2.latency", "0000:45:02.22")
	res1 = LMSGetValue("cmi.interactions.3. latency")

2.12 Paths

Data Element Name	Paths
Definition	A record of the path that the student took through an AU's material during an AU session. This data element allows the AU to record the AU segments entered by the student, the order in which the student experienced the segments, and the time spent in each segment (during an AU session). The number of segments in an AU is implementation dependent. This element is an array. Each record in this array corresponds to a single path taken in the current AU session. Each record is made up of the following sub-elements: *Paths.Location ID* *Paths.Date* *Paths.Time* *Paths.Status* *Paths.Why Left* *Paths.Time in Element* Each array record sub-element is described individually in this section
Usage	The AU sets all data elements in this group.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	

2.12.1 Paths.Location ID

Data Element Name	Paths.Location ID
Definition	A developer defined, unique identifier for a specific location within the AU visited by the student during an AU session.
Usage	The AU sets this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Element_Location

Data Element Name	Paths.Location_ID
Files & Obligations	Path File: CMI Optional, AU Optional
Name Format	Field Name: "Element_Location" case insensitive
Value Format	See data type *CMIIdentifierINI* for description.
Data type	CMIIdentifierINI
Examples	"Int-Eng-Start-1"
	"XYZ-1230-122"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutPath: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.paths.n.location_id"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.paths.***n***.location_id" – case sensitive where ***n*** is the (zero-based) array index
Value Format	See data type *CMIIdentifierINI* for description.
Data type	CMIIdentifierINI
Examples	LMSSetValue("cmi.paths.2.location_id", "int-Eng-Start-1")
	var log_path = LMSGetValue("cmi.paths.2.location_id")

2.12.2 Paths.Date

Data Element Name	Paths.Date
Definition	The calendar day on which the AU segment was entered.
Usage	The AU sets this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Path Date
Files & Obligations	Path File: CMI Optional, AU Optional
Name Format	Field Name: "Date" case insensitive
Value Format	See description of data type *CMIDate*
Data type	CMIDate
Examples	"1999/03/22"
	"2001/09/11"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutPath: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding

Data Element Name	Paths.Date
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.paths.n.date"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.paths.***n***.date" case sensitive where ***n*** is the (zero-based) array index for the Path record.
Value Format	See description of data type *CMIIDate*
Data type	CMIDate
Examples	LMSSetValue("cmi.paths.2.date", "2002/05/23")
	var pdate = LMSGetValue("cmi.paths.3.date")

2.12.3 Paths.Time

Data Element Name	Paths.Time
Definition	The time of day at which the student entered the AU segment.
Usage	The AU sets this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Path Time
Files & Obligations	Path File: CMI Optional, AU Optional
Name Format	Field Name: "Time" case insensitive
Value Format	See description of data type *CMIDate*
Data type	CMITime
Examples	"12:01:23.33"
	"14:05:43"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutPath: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.paths.n.time"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.paths.***n***.time" - case sensitive where ***n*** is the (zero-based) array index for the *Paths* record.
Value Format	Same as File Binding
Data type	Same as File Binding

Data Element Name	Paths.Time
Examples	LMSSetValue("cmi.paths.2.time", "13:03:45.45")
	var ptime = LMSGetValue("cmi.paths.3.time")

2.12.4 Paths.Status

Data Element Name	Paths.Status
Definition	A record of the student's performance in an AU segment each time he/she leaves that segment during an AU session.
Usage	Only the AU sets the value of *Paths.Status*. There are four possible values: • **passed**: The student mastered the AU segment. • **completed**: The student has visited all parts of the segment • **failed**: The student experienced some kind of assessment within the AU segment but did not demonstrate mastery. • **incomplete**: The AU segment was started but not finished. .
CMI Behavior	
AU Behavior	
File Binding	
Name	Status
Files & Obligations	Path File: CMI Optional, Finish: AU Optional
Name Format	Field Name: "Status" case insensitive
Value Format	One of the following vocabulary values: "passed" , "failed", "complete", "incomplete", "not attempted". All values are case insensitive. Only the first character is significant.
Data type	CMIVocabularyINI:Status
Examples	"Passed"
	"c"
	"F"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutPath : CMI Optional, AU optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	cmi.paths.n.status
API & Obligations	LMSGetValue() : CMI Optional, AU optional LMSSetValue() : CMI Optional, AU optional
Name Format	"cmi.paths.***n***.status" - case sensitive where ***n*** is the (zero-based) array index.
Value Format	A specific vocabulary limited to on of the following values: "passed", "completed", "failed" , "incomplete", "browsed", or "not attempted". All values are case sensitive
Data type	CMIVocabulary:Status
Examples	var stat5 = LMSGetValue("cmi.paths.5.status")

Data Element Name	Paths.Status
	LMSSetValue("cmi.paths.8.status", "passed")

2.12.5 Paths.Why Left

Data Element Name	Paths.Why Left
Definition	An indication why the student departed a segment in an AU.
Usage	The AU sets this element. There are four possible values that may be recorded: **Student selected:** The student selected some AU option, which resulted in his leaving the current AU segment. (Typically a menu, icon or some other kind of navigation control) **Lesson directed:** The logic of the AU moved a student out of the current AU segment to some other segment in the AU. **Exit by student**: A complete departure from the AU. For instance the student may have selected to log out or exit the AU. **Directed departure:** The AU forced the student out of the current session. An example might occur when the time limit is exceeded.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Why_Left
Files & Obligations	Path File: CMI Optional, AU Optional
Name Format	Field Name: "Why_Left" case insensitive
Value Format	See description of data type *CMIVocabularyINI:Why Left*
Data type	CMIVocabularyINI:Why Left
Examples	"S"
	"exit"
	"directed departure"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutPath: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.paths.n.why_left
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.paths.***n***.why_left" - case sensitive where ***n*** is the (zero-based) array index for the *Paths* record.
Value Format	See description of data type *CMIVocabulary:Why Left*
Data type	CMIVocabulary:Why Left
Examples	LMSSetValue("cmi.paths.2.why_left", "directed departure")

Data Element Name	Paths.Why Left
	var whyLeft = LMSGetValue("cmi.paths.3.why_left")

2.12.6 Paths.Time in Element

Data Element Name	Paths.Time in Element
Definition	The amount of time spent by the student in the AU segment.
Usage	The AU sets this element.
CMI Behavior Notes	The CMI stores and retains this data for reporting purposes.
AU Behavior Notes	
File Binding	
Name	Time_In_Element
Files & Obligations	Path File: CMI Optional, AU Optional
Name Format	Field Name: "Time_In_Element" case insensitive
Value Format	See description of data type *CMIDate*
Data type	CMITimespan
Examples	"12:01:23.33"
	"0014:05:43"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutPath: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.paths.n.time_in_element"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.paths.***n***.time_in_element" - case sensitive where ***n*** is the (zero-based) array index for the *Paths* record.
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.paths.2.time_in_element", "13:03:45.45")
	var ptime = LMSGetValue("cmi.paths.3.time_in_element")

2.13 Student Demographics

Data Element Name	Student Demographics
Definition	A grouping for a variety of data elements relating to demographic information about the student user.
Usage	All data elements in this category are optional. (See individual member data elements for obligations)
Membership	Student Demographics.City Student Demographics.Class Student Demographics.Company Student Demographics.Country Student Demographics.Experience Student Demographics.Familiar Name Student Demographics.Instructor Name Student Demographics.Native Language Student Demographics.State Student Demographics.Street Address Student Demographics.Telephone Student Demographics.Title Student Demographics.Years Experience

2.13.1 Student Demographics.City

Data Element Name	Student Demographics.City	
Definition	A Portion of student's current address that denotes the city.	
Usage	The CMI sets this element.	
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.	
AU Behavior Notes		
File Binding		
Name	City	
Files & Obligations	Startup: CMI Optional, AU Optional	
Name Format	"City" – case insensitive	
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)	
Data type	CMIString255INI	
Examples	`City = Toulouse`	
	`City = Seattle`	
	`City = Montreal`	
	`City = St. Louis`	
HACP Binding		
Name	Same as File Binding	
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional	
Name Format	Same as File Binding	
Value Format	Same as File Binding	
Data type	Same as File Binding	
Examples	Same as File Binding	
	Same as File Binding	
API Binding		

Data Element Name	Student Demographics.City
Name	"cmi.student_demographics.city"
API & OBLIGATIONS	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.city" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.city", "Toulouse")
	var city = LMSGetValue("cmi.student_demographics.city")

2.13.2 Student Demographics.Class

Data Element Name	Student Demographics.Class
Definition	An identifier for a predefined group of students, which are all, enrolled in the same course (of which the current AU is a member).
Usage	This grouping (class) is determined by the CMI and is implementation dependent. The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI may have a "class" of students that is enrolled in multiple courses.
AU Behavior Notes	
File Binding	
Name	Class
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Class" – case insensitive
Value Format	See description for data type *CMIIdentifierINI*
Data type	CMIIdentifierINI
Examples	`Class = FSL-737-200-Rdn1`
	`Class = NWA-A330-1204`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.class"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.class" - case sensitive
Value Format	See description for data type *CMIIdentifierINI*
Data type	CMIIdentifierINI
Examples	LMSSetValue("cmi.student_demographics.class", "FSL-737-200-Rdn1")
	var class = LMSGetValue("cmi.student_demographics.class")

2.13.3 Student Demographics.Company

Data Element Name	Student Demographics.Company	
Definition	The company or organization that the student is an employee and/or member of.	
Usage	The CMI sets this element. Format is implementation dependent.	
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.	
AU Behavior Notes		
File Binding		
Name	Company	
Files & Obligations	Startup: CMI Optional, AU Optional	
Name Format	"Company" – case insensitive	
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)	
Data type	CMIString255INI	
Examples	`Company = Airbus`	
	`Company = Northwest Airlines`	
	`Company = Alteon`	
HACP Binding		
Name	Same as File Binding	
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional	
Name Format	Same as File Binding	
Value Format	Same as File Binding	
Data type	Same as File Binding	
Examples	Same as File Binding	
	Same as File Binding	
API Binding		
Name	"cmi.student_demographics.company"	
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional	
Name Format	"cmi.student_demographics.company" - case sensitive	
Value Format	255 Character String	
Data type	CMIString255INI	
Examples	LMSSetValue("cmi.student_demographics.company", "Northwest Airlines")	
	var company = LMSGetValue("cmi.student_demographics.company")	

2.13.4 Student Demographics.Country

Data Element Name	Student Demographics.Country
Definition	A Portion of student's current address that denotes the country.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Country
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Country" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)

Data Element Name	Student Demographics.Country
Data type	CMIString255INI
Examples	`Country = Canada`
	`Country = France`
	`Country = United Kingdom`
	`Country = United States`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.country"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.country" - case sensitive
Value Format	255 Character String
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.country", "France")
	var country = LMSGetValue("cmi.student_demographic.country")

2.13.5 Student Demographics.Experience

Data Element Name	Student Demographics.Experience
Definition	Information on the student's past experience that may be used by an AU to determine what to present, or what presentation strategies to use.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Experience
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Experience" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Experience = 737-700 Type Rating`
	`Experience = 5 Years Avionics 737,767`
	`Experience = Type Rating - A330/A340`
	`Experience = A/P only`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding

Data Element Name	Student Demographics.Experience
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.experience"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.experience" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.experience", "France")
	var country = LMSGetValue("cmi.student_demographic.experience")

2.13.6 Student Demographics.Familiar Name

Data Element Name	Student Demographics.Familiar Name
Definition	In some cases, an AU may attempt to be more personal by using a student's name in its feedback. This provides a mechanism for the CMI system to inform the AU how it should refer to the student.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Familiar_Name
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Familiar_Name" – case insensitive
Value Format	255 Character String
Data type	CMIString255INI
Examples	Familiar_Name = `Skip Winger`
	Familiar_Name = `Chip`
	Familiar_Name = `Jacques`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.familiar_name"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.familiar_name"" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)

Data Element Name	Student Demographics.Familiar Name
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.familiar_name", "Jacques")
	var country = LMSGetValue("cmi.student_demographic.familiar_name"")

2.13.7 Student Demographics.Instructor Name

Data Element Name	Student Demographics.Instructor Name
Definition	Name of the instructor responsible for the student's understanding of the material in the AU.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Instructor_Name
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Instructor_Name" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	Instructor_Name = Jack Hyde
	Instructor_Name = Jean-François Schmidt
	Instructor_Name = Xavier Zeigler
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.instructor_name"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.instructor_name" - case sensitive
Value Format	255 Character String
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.instructor_name", "Xavier Zeigler")
	var instrName = LMSGetValue("cmi.student_demographic.instructor_name"")

2.13.8 Student Demographics.Native Language

Data Element Name	Student Demographics.Native Language
Definition	The language with which the student is most familiar. This may not be the preferred language for the instructional delivery.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile

Data Element Name	Student Demographics.Native Language
	information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Native_Language
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Native_Language" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Native_Language = French`
	`Native_Language = Chinese`
	`Native_Language = English`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.native_language"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.native_language" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.native_language", "`French`")
	var natlang = LMSGetValue("cmi.student_demographics.native_language")

2.13.9 Student Demographics.State

Data Element Name	Student Demographics.State
Definition	A Portion of student's current address that denotes the state, province, or local region within the country.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	State
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"State" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`State = Quebec`
	`State = Missouri`
	`State = Manitoba`

Data Element Name	Student Demographics.State
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.state"
API & OBLIGATIONS	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.state" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.state", "Missouri")
	var state = LMSGetValue("cmi.student_demographic.state")

2.13.10 Student Demographics.Street Address

Data Element Name	Student Demographics.Street Address
Definition	A Portion of student's current address that denotes the street address.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Street_Address
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Street_Address" – case insensitive
Value Format	255 Character String
Data type	CMIString255INI
Examples	`Street_Address = 1601 Pennsylvania Avenue`
	`Street_Address = 1301 SW 16th Street`
	`Street_Address = Manitoba`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.street_address"

Data Element Name	Student Demographics.Street Address
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.street_address" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.street_address", "Missouri")
	var addr = LMSGetValue("cmi.student_demographics.street_address")

2.13.11 Student Demographics.Telephone

Data Element Name	Student Demographics.Telephone
Definition	The telephone number of a student. May include country codes or extensions.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Telephone
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Telephone" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	`Telephone = 1-800-555-5555 ext 123`
	`Telephone = +44 482 663622`
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.telephone"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional LMSSetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.telephone" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.telephone", "+1-555-555-5555")
	var tel = LMSGetValue("cmi.student_demographics.telephone")

2.13.12 Student Demographics.Title

Data Element Name	Student Demographics.Title

Data Element Name	Student Demographics.Title
Definition	The job title of the student.
Usage	The CMI sets this element. Format is implementation dependent.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Job_Title
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Job_Title" – case insensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	Job_Title = Pilot
	JOB_TITLE = First Officer
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.title"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.title" - case sensitive
Value Format	A 255 character string. (See Datatype *CMIString255INI* for details)
Data type	CMIString255INI
Examples	LMSSetValue("cmi.student_demographics.title", "First Officer")
	var title = LMSGetValue("cmi.student_demographics.title")

2.13.13 Student Demographics.Years Experience

Data Element Name	Student Demographics.Years Experience
Definition	Number of years the student has performed in current or similar position.
Usage	The CMI sets this element.
CMI Behavior Notes	The CMI passes this data to the AU based on student user profile information stored in the CMI.
AU Behavior Notes	
File Binding	
Name	Years_Experience
Files & Obligations	Startup: CMI Optional, AU Optional
Name Format	"Years_Experience" – case insensitive
Value Format	Integer value 0 or higher
Data type	CMIInteger
Examples	Years_Experience = 5
	Years_Experience = 6

Data Element Name	Student Demographics.Years Experience
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	GetParam (response): CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
	Same as File Binding
API Binding	
Name	"cmi.student_demographics.years_experience"
API & Obligations	LMSGetValue(): CMI Optional, AU Optional
Name Format	"cmi.student_demographics.years_experience"" - case sensitive
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	LMSSetValue("cmi.student_demographics.years_experience", "5")
	var yearsexp = LMSGetValue("cmi.student_demographics.years_experience",")

2.14 Lesson_ID

Data Element Name	.Lesson_ID
Definition	The unique identifier for the Assignable Unit that the student user was in when the comment was written. This is unique to, and inherent in each AU. See *Course Elements.Developer ID* (section 3.4.2).
Usage	The value for this element must set to the same value as the AU's Developer_ID (in the course structure). See *Course Elements.Developer ID* (section 3.4.2).
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Lesson_ID
Files & Obligations	Comments File : CMI Optional, AU Optional Interactions File :CMI Optional, AU Optional Objectives Status File : CMI Optional, AU Optional Path File : CMI Optional, AU Optional
Name Format	Not Applicable
Value Format	See data type *CMIIdentifierDevID* for format description. While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*
Data type	CMIIdentifierDevID
Examples	"{E8128C30-6BF8-11cf-96FC-0020AFED9A65}"
HACP Binding	
Name	Same as File Binding
HACP Message(s) & Obligations	PutComments : CMI Optional, AU Optional PutInteractions : CMI Optional, AU Optional PutObjectives : CMI Optional, AU Optional PutPath: CMI Optional, AU Optional
Name Format	Same as File Binding
Value Format	Same as File Binding
Data type	Same as File Binding
Examples	Same as File Binding
API Binding	
Name	Not Applicable
API & Obligations	Not Applicable
Name Format	Not Applicable
Value Format	Not Applicable
Data type	Not Applicable
Examples	Not Applicable

3.0 Course Structure Data Model

This data model contains all of the information covered by this specification to describe a course (that may be passed from one CMI system to another thru a course import/export process).

Data in this data model is also stored internally by the CMI system and is used by the CMI in determining some values of the communication data model elements sent to AU's in the course at runtime. The communication data model is described in chapter 2.0. A CMI system may have the ability to internally store and use some of the data elements in the course structure data model without necessarily supporting them for import/export.

There is only one binding for this data model (described in chapter 8.0).

The sequencing (of Assignable Units) within a course (using this data model) is described in chapter 4.0. The table below lists all of the data elements in this data model.

Table Legend:

Name Indicates the name of the data element.
Definition Indicates where in this document a definition of the data element is found.
Mult Indicates whether the element has only a single value – SV - or may have multiple values - MV.
Obligation This indicates whether the data element is required or optional

Name	Definition	Mult	Obligation
Course	Section 3.1	SV	Mandatory
Course.Creator	Section 3.1.1	SV	Mandatory
Course.ID	Section 3.1.2	SV	Mandatory
Course.System	Section 3.1.3	SV	Mandatory
Course.Title	Section 3.1.4	SV	Mandatory
Course.Level	Section 3.1.5	SV	Mandatory
Course.Max Fields CST	Section 3.1.6	SV	Mandatory
Course.Max Fields ORT	Section 3.1.7	SV	Optional
Course.Total AUs	Section 3.1.8	SV	Mandatory
Course.Total Blocks	Section 3.1.9	SV	Mandatory
Course.Total Objectives	Section 3.1.10	SV	Optional
Course.Total Complex Objectives	Section 3.1.11	SV	Optional
Course.Version	Section 3.1.12	SV	Mandatory
Course Behavior	Section 3.2	SV	Mandatory
Course Behavior. Max Normal	Section 3.2.1	SV	Mandatory
Course Description	Section 3.3	SV	Mandatory
Course Elements	Section 3.4	MV	Mandatory
Course Elements.System ID	Section 3.4.1	SV	Mandatory
Course Elements.Developer ID	Section 3.4.2	SV	Mandatory
Course Elements.Title	Section 3.4.3	SV	Mandatory
Course Elements. Description	Section 3.4.4	SV	Mandatory
Course Elements.Type	Section 3.4.5	SV	Mandatory
Course Elements.Command Line	Section 3.4.6	SV	Mandatory
Course Elements.File Name	Section 3.4.7	SV	Mandatory
Course Elements.Mastery Score	Section 3.4.8	SV	Optional
Course Elements.Max Score	Section 3.4.9	MV	Optional
Course Elements.Max Time Allowed	Section 3.4.10	SV	Optional
Course Elements.Time Limit Action	Section 3.4.11	SV	Optional
Course Elements.Development System	Section 3.4.12	SV	Mandatory
Course Elements.Launch Data	Section 3.4.13	SV	Mandatory
Course Elements.Web Launch Parameters	Section 3.4.14	MV	Mandatory
Course Elements.AU Password	Section 3.4.15	SV	Optional
Course Elements.Members	Section 3.4.16	MV	Mandatory
Course Elements.Members.System ID	Section 3.4.16.1	SV	Mandatory
Course Elements.Prerequisite	Section 3.4.17	SV	Optional

Name	Definition	Mult	Obligation
Course Elements.Completions	Section 3.4.18	MV	Optional
Course Elements.Completions.Requirement	Section 3.4.18.1	SV	Optional
Course Elements.Completions.Status if True	Section 3.4.18.2	SV	Optional
Course Elements.Completions.Next AU if True	Section 3.4.18.3	SV	Optional
Course Elements.Completions.Goto after Next	Section 3.4.18.4	SV	Optional

Each element in this data model is described in tables in the following sections. The fields for each of these tables are as follows:

Data Element Name
The data elements in this model are arranged hierarchically (in a "parent/child" relationship). Hierarchy levels are delimited by period (".")s in the data element name. Any item to the right of the period delimiter is the "child" of proceeding item (e.g. in "Course.ID", "Course.ID" is a child of "Course" and "Course" is the parent of "Course.ID").

Definition
A description of the data element and what it is used for.

Usage
Usage rules for data element.

CMI Behavior Notes
A description of the expected or recommended CMI behavior when using the data element. (This field augments "Usage)

AU Behavior Notes
A description of the expected or recommended CMI behavior when using the data element. (This field augments "Usage)

File Binding: Name
Data element name when used when referring to this element when used in the file binding.

File Binding: In File(s)
Files in which the data element is contained.

File Binding: Obligation
Whether or not the data element is required for a valid course structure (in the file binding).

File Binding: Name Format
Formatting for the Name of the data element written in the files.

File Binding: Value Format
This field adds additional explanation for valid values that a field may have (in addition to the definition that *data type* provides).

File Binding: Data Type
Each data element binding is assigned a "data type". The data type defines the size of data element and the valid ranges of values. See *section 10. Data Types*

File Binding: Examples
Examples of how data element is represented in files.

3.1 Course

Data Element Name	Course
Definition	This category of data elements contains information that applies to the course as a whole. Some of this data is designed to help in processing the more detailed information on other data elements in the course and how they are ordered.
Usage	See individual member data elements for obligations
Membership	Course.Creator Course.ID Course.System Course.Title Course.Level Course.Max Fields CST Course.Max Fields ORT Course.Total Aus Course.Total Blocks Course.Total Objectives Course.Total Complex Objectives Course.Version

3.1.1 Course.Creator

Data Element Name	Creator
Definition	The name of the organization or individual that authored of the course
Usage	
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Course_Creator
In Files	Course
Obligation	Mandatory
Name Format	"Course_Creator" Case insensitive.
Value Format	
Data type	CMIString255INI
Examples	Course_Creator = "Boeing Commercial Airplane Group, Customer Services"
	Course_Creator = Airbus
	Course_Creator = John, Bill, Bob, Anne, Sally

3.1.2 Course.ID

Data Element Name	ID
Definition	A unique identifier for the course.
Usage	The value of this element is provided by CMI to AU's at runtime via the *Evaluation.Course_ID* communication data element.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Course_ID
In Files	Course
Obligation	Mandatory

Data Element Name	ID
Name Format	"Course_ID" Case insensitive.
Value Format	See data type *CMIIndentifierDevID* for description While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*
Data type	CMIIndentifierDevID
Examples	A320-Trans-NWA-2
	737-700-EZY-2002-Rec

3.1.3 Course.System

Data Element Name	System
Definition	The name the predominant authoring system used to create the course.
Usage	Values are not intended for runtime (machine) interpretation. Provided for informational purposes only.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Course_System
In Files	Course
Obligation	Mandatory
Name Format	"Course_System" Case insensitive.
Value Format	255 character string
Data type	CMIString255INI
Examples	Course_System=Authorware
	Course_system = PCD3 authoring
	Course_System=WISE
	Course_System=VACBI

3.1.4 Course.Title

Data Element Name	Title
Definition	A descriptive name (or title) given to the course.
Usage	Used by the CMI to display (or report) course title to students and administrative users
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Course_Title
In Files	Course
Obligation	Mandatory
Name Format	"Course_Title" Case insensitive.
Value Format	255 character string
Data type	CMIString255INI
Examples	747 Flight Crew Training
	Maintaining 747 Avionics
	Maintaining A310 Hydraulic Systems

3.1.5 Course.Level

Data Element Name	Level
Definition	Complexity level of the file's description of the course. There are three levels of complexity numbered 1 through 3. One is the simplest to 3, the most complex. Level 3 is divided into two parts, referred to as 3a and 3b. (See section 3.5 for a detailed description of each level of course complexity)
Usage	The CMI system may or may not support all levels. Support for level 1 is a minimum requirement. Possible values for this element are as follows: **1** Support Level 1 course interchange. May support some features from higher levels as well. **2** Supports all features of level 1 and level 2. May support some features from level 3. **3** Supports all level 1, 2, 3a, and 3b features of course interchange. **3a** Supports level 1, 2, and 3a interchange. **3b** Supports level 1, 2, and 3b interchange
CMI Behavior Notes	If the complexity level of a specific course is not supported, the CMI system may provide a warning to the user.
AU Behavior Notes	
File Binding	
Name	Level
In Files	Course
Obligation	Mandatory
Name Format	"Level" Case insensitive.
Value Format	Alphanumeric characters. Allowed vocabulary is "1", "2", "3", "3a", or "3b"
Data type	CMILevel
Examples	`Level = 3`
	`level=2`
	`level = 3a`

3.1.6 Course.Max Fields CST

Data Element Name	Max Fields CST
Definition	Identifies the maximum number of fields that are in the course structure table/file (xxxxxxxx.CST file).
Usage	Some CMI systems may use this information to help process the information in the Course Structure Table.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Max_Fields_CST
In Files	Course
Obligation	Mandatory
Name Format	"Max_Fields_CST" Case insensitive.
Value Format	Numeric characters.
Data type	CMInteger
Examples	Max_fields_CST=12 ; There is at least one block (or the course itself) that ; has 11 members.
	Max_Fields_CST = 9

3.1.7 Course.Max Fields ORT

Data Element Name	Max Fields ORT
Definition	Identifies the maximum number of fields that are in the objectives relationships table (any.ORT file).
Usage	Some CMI systems may use this information to help process the information in the Objectives Relationship Table.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Max_Fields_ORT
In Files	Course
Obligation	Optional
Name Format	"Max_Fields_ORT" Case insensitive.
Value Format	Numeric characters.
Data type	CMInteger
Examples	Max_fields_ORT=12 ; There is at least one element in the left-most column that ; has 11 members.
	Max_Fields_ORT = 9

3.1.8 Course.Total AUs

Data Element Name	Total AUs
Definition	The total number of unique assignable units in the course.
Usage	This information may aid in the processing of information in the course structure.
CMI Behavior Notes	This number does not necessarily represent the largest digit used to identify an AU. AU identifiers do not have to be consecutive. If there are 5 AUs in a course (Total_AUs=5), they could be identified as A.001, A.0021, A2, A3, A.505.
AU Behavior Notes	
File Binding	
Name	Total_AUs
In Files	Course
Obligation	Mandatory
Name Format	"Total_AUs" Case insensitive.
Value Format	Numeric characters.
Data type	CMIInteger
Examples	Total_AUs = 3 ; There are three assignable units in the course.
	Total_AUs= 84

3.1.9 Course.Total Blocks

Data Element Name	Total Blocks
Definition	The total number of unique blocks in the course.
Usage	This information may aid in the processing of information in the course structure.
CMI Behavior Notes	As with *Course.Total AUs* this number does not have to be equal to the largest number used in Block System Identifiers.

Data Element Name	Total Blocks
AU Behavior Notes	
File Binding	
Name	Total_Blocks
In Files	Course
Obligation	Mandatory
Name Format	"Total_Blocks" Case insensitive.
Value Format	Numeric characters.
Data type	CMIInteger
Examples	Total_Blocks = 3 ; There are three blocks in the course.
	Total_blocks= 84

3.1.10 Course.Total Objectives

Data Element Name	Total Objectives
Definition	The total number of unique objectives in the course. This number includes both complex and simple objectives.
Usage	This information may aid in the processing of information in the course structure.
CMI Behavior Notes	As with *Course.Total AUs*, this number does not have to be equal to the largest number used in Objectives System Identifiers.
AU Behavior Notes	
File Binding	
Name	Total_Objectives
In Files	Course
Obligation	Optional
Name Format	"Total_Objectives" Case insensitive.
Value Format	Numeric characters.
Data type	CMIInteger
Examples	Total_Objectives = 3 ; There are three objectives in the course.
	Total_objectives= 84

3.1.11 Course.Total Complex Objectives

Data Element Name	Total Complex Objectives
Definition	The total number of unique complex objectives in the course. A complex objective is an objective that has one or more *Course Elements.Members.*
Usage	This information may aid in the processing of information in the course structure.
CMI Behavior Notes	As with *Course.Total AUs* this number does not have to be equal to the largest number used in Objectives System Identifiers.
AU Behavior Notes	
File Binding	
Name	Total_Complex_Obj
In Files	Course
Obligation	Optional
Name Format	"Total_Complex_Obj" Case insensitive.
Value Format	Numeric characters.
Data type	CMIinteger
Examples	Total_Complex_Obj = 3

Data Element Name	Total Complex Objectives
	; There are three complex objectives in the course.
	Total_complex_obj= 84

3.1.12 Course.Version

Data Element Name	Version
Definition	Identifies the ***CMI001 - CMI Guidelines for Interoperability*** document (i.e. THIS specification's) revision number on which the Course Structure format is based.
Usage	This element may aid in the processing of information in the accompanying files. Version number vocabulary is restricted to published versions of this document.
CMI Behavior Notes	CMI systems may use different course structure import/export logic based on the value of this element.
AU Behavior Notes	
File Binding	
Name	Version
In Files	Course
Obligation	Mandatory
Name Format	"Version" Case insensitive.
Value Format	See data type *CMIVersionNumber* for the vocabulary of allowed values.
Data type	CMIVersionNumber
Examples	Version = 2.0
	version=3.5

3.2 Course Behavior

Data Element Name	Course Behavior
Definition	This category of data elements is used to define keywords that can be used to affect the behavior of the CMI system for the course.
Usage	
Membership	Course Behavior.Max Normal

3.2.1 Course Behavior.Max Normal

Data Element Name	Max Normal
Definition	The maximum number of assignable units that may be taken for credit simultaneously. This value indicates how many AU's launched with credit = credit are allowed to be incomplete.
Usage	When this number is exceeded, subsequent launches of AU's in the course must be with a Core.Credit value of "no-credit". Further, the default CMI behavior is to launch all subsequent AU's with a Core.Lesson Mode value of "Browse". Valid values are 1 to 99 inclusive. If no number is indicated, 1 is assumed. If a number greater than 99 is indicated, then 99 is assumed.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Max_Normal
In Files	Course
Obligation	Mandatory

Data Element Name	Max Normal
Name Format	"Max_Normal" Case insensitive
Value Format	A single integer number. Valid values are 1 to 99 inclusive.
Data type	CMIinteger
Examples	Max_Normal=1 ; only 1 AU being taken for credit can be incomplete.
	Max_Normal = 5

3.3 Course Description

Data Element Name	Course Description
Definition	This is a textual description of the contents of the course. It may contain the purpose, or the scope, or a summary of the course objectives.
Usage	May be used to display/report to a student or an administrative user the instructional description and purpose of the course.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Course_Description
In Files	Course
Obligation	Mandatory
Name Format	"[Course_Description]" Case insensitive
Value Format	Freeform text. Carriage returns are implied (explicitly) at the end of each line.
Data type	CMIString4096INI

Examples	`; The value of Course description starts with "This"`
	`; and ends with "change."`
	`;`
	
	`[COURSE_DESCRIPTION]`
	
	`This course explains the new JAA rules for RVSM and the procedures affected by this change.`
	
	`[Vendor Specific Group]`

3.4 Course Elements

Data Element Name	Course Elements
Definition	A Course Element is an Assignable Unit, a Block, or an Objective. This category has information about individual Course Elements and indicates how those Course Elements are organized and how they relate to one another. Additionally, this category includes information that allows the sequencing of the Course Elements using prerequisites and completion requirements for each when necessary.
Usage	Describe features of individual Course Elements, and how they are organized and sequenced in a course.
CMI Behavior Notes	The order of the data elements implies (but does not force) an order for presentation to the student.

Data Element Name	Course Elements
	Should a developer wish to specify a course sequence, *Course Elements.Prerequisite* and *Course Elements.Completion Requirement* are used to specify the order. The first element in this category is always a Block and always has the System ID of "root".
AU Behavior Notes	
File Binding	
Name	N/A
In Files	Information in this category can be found in the following files: Assignable Unit, Descriptor, Course Structure, Objectives Relationships, Prerequisites, Completion Requirements
Obligation	
Name Format	N/A
Value Format	
Data type	
Examples	

3.4.1 Course Elements.System ID

Data Element Name	System ID
Definition	A system assigned, unique, Course Element identifier. The exporting system for the course structure generates this unique identifier for every Course Element.
Usage	The value for *Course Elements.System ID* must be unique for each individual Course Element within a given course structure. One Course Element in a course structure has a *Course Elements.System ID* value of "root" (this is a special ID for the root membership of Course Elements in the course structure's hierarchy). All other values for *Course Elements.System ID* have the following naming convention: A letter and a number. The letter identifies the category of Course Element. Possible Course Element categories are as follows: **A** -- Assignable Unit **B** – Block **J** -- Objective or complex objective The number is a simple integer to distinguish each unique item in a category. Lead/trailing zeros are significant ("B011" and "B11" are different identifiers)
CMI Behavior Notes	The numbers assigned by the CMI system do not have to be sequential.
AU Behavior Notes	
File Binding	
Name	System_ID
In Files	Assignable Unit, Descriptor, Course Structure, Objectives Relationships, Prerequisites, and Completion Requirements.
Obligation	Mandatory
Name Format	System ID's appear in all of the course structure files and have the following field (header) names:

Data Element Name	System ID
	• "System_ID" in Assignable Unit and Descriptor Files. • "Block" in Course Structure File. • "Course_Element" in Objectives Relationships File. • "Member" in Course Structure and Objectives Relationships Files. • "Structure_Element" in Prerequisites File. • "Structure_Element" in Completion Requirements File. • Case insensitive in all files.
Value Format	A valid system identifier (as defined in data type *CMISIdentifier*) or the value "root". Case insensitive in all files.
Data type	CMISIdentifier
Examples	A15
	B1005
	J015

3.4.2 Course Elements.Developer ID

Data Element Name	Developer ID
Definition	A developer assigned (unique) identifier for a Course Element.
Usage	For each Course Element, the value of *Course Elements.Developer ID* must be unique within a course structure. At AU launch time, the CMI system passes the value of this item to the AU via *Evaluation.Lesson ID* or *Objectives.ID* communication data (See sections *2.7.2 Evaluation.Lesson ID* and *2.8.1 Objectives.ID*).
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Developer_ID
In Files	Descriptor
Obligation	Mandatory
Name Format	"Developer_ID" Case insensitive.
Value Format	See description for data type *CMIIndentifierDevID* While the *CMIIdentifierDevID* data format is valid, it is recommended that data type *CMIIdentifierGUID's* formatting rules be used instead to reduce the problems associated with developer ID collisions. Note that *CMIIdentifierGUID* is a subset of *CMIIdentifierDevID.*
Data type	CMIIndentifierDevID
Examples	APU-747-003
	A320_415_ELEC_001
	A320_415_ELEC_002

3.4.3 Course Elements.Title

Data Element Name	Title
Definition	Commonly used name for an assignable unit, block, objective, or complex objective.
Usage	May be used by CMI system in menu screens where students can see or select an assignable unit or block, or see the status of an objective.
CMI Behavior Notes	

Data Element Name	Title
AU Behavior Notes	
File Binding	
Name	Title
In Files	Descriptor
Obligation	Mandatory
Name Format	“Title” Case insensitive.
Value Format	Alphanumeric plus hyphens and underscores spaces and commas.
Data type	CMIString255CSV
Examples	"Auxiliary Power Unit, Part 1"
	"Auxiliary Power Unit Start"
	"Electrical Power, Part 3"

3.4.4 Course Elements.Description

Data Element Name	Description
Definition	This is a textual description of the assignable unit, objective, etc. It may contain the purpose, or the scope, or a summary of the element.
Usage	Designed for human reading and understanding (display/reporting) only, not intended for other purposes.
CMI Behavior Notes	The CMI system may provide a visual interface to display *Course Elements.Description* to a student or administrative user on request.
AU Behavior Notes	
File Binding	
Name	Description
In Files	Descriptor
Obligation	Mandatory
Name Format	“Title” Case insensitive.
Value Format	Free form textual description. Carriage returns are specially encoded and are translated prior to display/reporting. The string “<CR>” delimits embedded carriage returns.
Data type	CMIString4096CSV
Examples	“This course teaches the following: <CR> 1. How to Locate the exits <CR>2. How to locate the emergency equipment<CR> 3. How use the cabin intercom system”

3.4.5 Course Elements.Type

Data Element Name	Type
Definition	Assignable units (AU’s) may be categorized. *Course Elements.Type* identifies a developer-defined category of assignable unit. These are determined by the designer/developer of the assignable unit.
Usage	*Course Elements.Type* may be related to the ability of an assignable unit to respond to student preferences. Assignable units with the same value of *Course Elements.Type* may be able to process all student preferences created and passed from other AU’s of the same “type”.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Type
In Files	Assignable Unit

Data Element Name	Type
Obligation	Optional
Name Format	"Type" Case insensitive.
Value Format	Alphanumeric. Not case sensitive. May contain spaces and commas.
Data type	CMIString255CSV
Examples	BTI Lesson
	A320 Unit
	M1684_ZX

3.4.6 Course Elements.Command Line

Data Element Name	Command Line
Definition	The string of characters needed to successfully launch an executable program in the Microsoft Windows operating environment. Environment variables may be embedded in the command line
Usage	This information is only used by Assignable Units. It is not appropriate for Blocks and Objectives. Specific file and directory locations that may be contained within this data element are installation specific. It is the course structure creator's responsibility to provide either an automated installation process or a written manual procedure for modifying this data element in the AU file to reflect the actual installed location of the AU's in a course. This field is left blank for web-based AU's.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Command Line
In Files	Assignable Unit
Obligation	Mandatory
Name Format	"Command_Line" Case insensitive.
Value Format	Alphanumeric. Not case sensitive. May contain spaces.
Data type	CMIString255CSV
Examples	"APU /UAL/MN"
	"ELEC3 –nuv3"
	"%lesloc%ELEC3 –nuv3"

3.4.7 Course Elements.File Name

Data Element Name	File Name
Definition	The fully qualified name of the file containing the most critical content of the assignable unit (an assignable unit may require several files). The purpose of this field is to enable the CMI to locate the primary file needed to launch an AU.
Usage	The filename indicates either a fully qualified windows file path or a fully qualified URL (depending upon whether the course is file-based or web based) For web-based courses, this URL indicates the "point of entry" for web-based AU's. The AU filename location is installation specific. It is the course structure creator's responsibility to provide either an automated

Data Element Name	File Name
	installation process or a written manual procedure for modifying the filename values in the AU file to reflect the actual installed location of the AU's in a course. This field is not used for non-AU Course Elements (i.e. Blocks and Objectives).
CMI Behavior Notes	This element may be used to reference a non-conforming AU that does not communicate with the CMI. In this case, the method of determining communication data elements (like *Core.Lesson Status*) for the AU sessions by the CMI is undefined and implementation dependent.
AU Behavior Notes	
File Binding	
Name	File_Name
In Files	Assignable Unit
Obligation	Mandatory
Name Format	"File_Name" Case insensitive.
Value Format	A URL for web-based (in web-based courses) or a Windows File name (in file-base courses).
Data type	CMIurl or CMIFileNameFull
Examples	"C:\somedir\somefile.exe"
	"E:\afile.A4P"
	"http://somehost.com/dir1/dir2/index.html"

3.4.8 Course Elements.Mastery Score

Data Element Name	Mastery Score
Definition	See section *2.9.3 Student Data.Mastery Score.*
Usage	The value of *Course Elements.Mastery Score* is passed to the AU via *Student Data.Mastery Score* by the CMI at AU launch time.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Mastery_Score
In Files	Assignable Unit
Obligation	Optional
Name Format	"Mastery_Score" Case insensitive.
Value Format	Decimal number.
Data type	CMIDecimal
Examples	.85
	85
	16

3.4.9 Course Elements.Max Score

Data Element Name	Max Score
Definition	The maximum possible value for *Core.Score.Raw* that the assignable unit will return. The AU designer determines this value.
Usage	If an AU does not support a *Core.Score.Max* to the CMI, *Course Elements.Max Score* allows the CMI system to compute a percentage from the *Core.Score.Raw* value provided by the AU.
CMI Behavior Notes	

Data Element Name	Max Score
AU Behavior Notes	
File Binding	
Name	Max_Score
In Files	Assignable Unit
Obligation	Optional
Name Format	"Max_Score" Case insensitive.
Value Format	Decimal number.
Data type	CMIDecimal
Examples	1
	1.0
	23

3.4.10 Course Elements.Max Time Allowed

Data Element Name	Max Time Allowed
Definition	See section *2.9.3 Student Data.Max Time Allowed.*
Usage	The value of *Course Elements.Max Time Allowed* is passed to the AU by CMI via *Student Data.Max Time Allowed* at AU launch time.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Max_Time_Allowed
In Files	Assignable Unit
Obligation	Optional
Name Format	"Max_Time_Allowed" Case insensitive.
Value Format	See description of data type *CMITimeSpan*
Data type	CMITimeSpan
Examples	00:25:00
	01:12:00
	00:00:24.3

3.4.11 Course Elements.Time Limit Action

Data Element Name	Time Limit Action
Definition	See section *2.9.4 Student Data.Time Limit Action*
Usage	The value of *Course Elements.Time Limit Action* is passed to the AU by CMI via *Student Data.Time Limit Action* at AU launch time.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Time_Limit_Action
In Files	Assignable Unit
Obligation	Optional
Name Format	"Time_Limit_Action" Case insensitive.
Value Format	See *CMIVocabularyINI:Time Limit Action* for description
Data type	CMIVocabularyINI:Time Limit Action
Examples	"E,N"
	"exit,no_message"
	"c,m"

3.4.12 Course Elements.Development System

Data Element Name	System
Definition	Authoring system (or development tools) used to create the assignable unit. This information is provided by the course developer
Usage	For display/reporting (informational) purposes only. Not intended for machine interpretation.
CMI Behavior Notes	The CMI administrative user interface may display this information in a course editing or reporting functions.
AU Behavior Notes	
File Binding	
Name	System Vendor
In Files	Assignable Unit
Obligation	Optional
Name Format	"System_Vendor" Case insensitive.
Value Format	Authoring system and version number.
Data type	CMIString255CSV
Examples	Authorware 3.2
	Tool Book 4.0
	VACBI 2.0

3.4.13 Assignable Unit.Launch Data

Data Element Name	Launch Data
Definition	See section *2.3 Launch Data.*
Usage	The value of *Assignable Unit.Launch Data* is passed by the CMI to the AU via *Launch Data* at AU launch time. Prior to passing this value to *Launch Data,* carriage return tokens (in the form of the string "<CR>" - case insensitive) in *Assignable Unit.Launch Data* are translated to carriage return/line feeds.
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Core Vendor
In Files	Assignable Unit
Obligation	Mandatory
Name Format	"Core_Vendor" Case insensitive.
Value Format	Carriage returns are encoded with "<CR>" (case insensitive) tokens.
Data type	CMIString4096CSV
Examples	"Testmode=on <CR>configuration=PW168<CR>audience=FO"
	"Testmode/on, configuration/PW168, audience/FO"

3.4.14 Course Elements.Web Launch Parameters

Data Element Name	Web Launch Parameters
Definition	AU-specific launch parameters for web-based AU's. Additional name/value parameters that must be appended to the "URL Command line" (See sections 6.3) at AU launch time.
Usage	This data is appended to the "query" portion (after the "?" separator) of the "URL command line". (See sections 6.3)
CMI Behavior Notes	
AU Behavior Notes	

Data Element Name	Web Launch Parameters
File Binding	
Name	Web Launch Parameters
In Files	Assignable Unit
Obligation	Mandatory
Name Format	"Web_Launch" Case insensitive.
Value Format	URL-encoded name/value pairs. Formatted in the following manner: 1. Values of the parameters are communicated in (name/value pair) form "<parameter Name> = <Parameter value>". 2. The "name/value pairs" are separated by ampersands ("&"). 3. The name/value pairs can be in any order. 4. Parameter names are not case sensitive. 5. Parameter values may be case sensitive. All parameters must be URL-encoded (see section 6.4.1.1)
Data type	CMIurlEncNVPairList
Examples	Vparam1=1234&Vparam2=Question%3F&vparam3=more+stuff

3.4.15 Course Elements.AU Password

Data Element Name	AU Password
Definition	A string of characters sent to the CMI system that enables the CMI system to authenticate an assignable unit. This authentication is independent of any user authentication that the CMI system uses.
Usage	The password value is AU developer-defined and is sent with HACP request messages (see section 6.4.2), so that the CMI system can authenticate the AU making the request. The CMI compares the value of this element with the value passed by the AU in HACP request messages. If an AU has an *AU Password* defined in the course and the corresponding AU does not issue the proper password value in HACP request message, then the CMI must issue a HACP response message (see section 6.4.3), with the appropriate error number (see section 6.4.8).
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Web Launch Parameters
In Files	Assignable Unit
Obligation	Optional
Name Format	"AU_Password" Case insensitive.
Value Format	See datatype in section 9.0.
Data type	CMIString255CSV
Examples	Trust!one
	TheSecretWord

3.4.16 Course Elements.Members

Data Element Name	Members

Data Element Name	Members
Definition	A list (array) of "members" of a Course Element in the course structure data model hierarchy. The course structure data model follows a hierarchy where Course Elements may contain other Course Elements in the following manner: • Blocks may contain Assignable Units and Objectives. • Objectives may contain Blocks and Assignable Units. • Assignable Units may contain Objectives. Each record in this array is composed of the following sub-elements: • *Course Elements.Members.System ID*
Usage	The values for each item in this list are the *Course.System ID* values of other Course Elements. (i.e. the children referenced as being contained in this Course Element)
CMI Behavior Notes	When there are no explicit completion requirements for a Course Element, the status of the Course Element is determined by the status of its members and default rules (see section 4.2.1).
AU Behavior Notes	

3.4.16.1 Course Elements.Members.System ID

Data Element Name	System ID	
Definition	The System ID (See *Course Elements.System ID)* identifying the Course Element that is contained in (is a member of) the current Course Element.	
Usage	The value of this field is set to the value of *Course Elements.System ID* for the Course Element that is a member of the current Course Element	
CMI Behavior Notes		
AU Behavior Notes		
File Binding		
Name	Member	
In Files	Course Structure and Objectives Relationships	
Obligation	Mandatory	
Name Format	"Member" Case insensitive.	
Value Format	A valid system identifier. Case insensitive in all files. See description of data type *CMISIdentifier*	
Data type	CMISIdentifier	
Examples	B15	
	A023	
	J53	

3.4.17 Course Elements.Prerequisite

Data Element Name	Course Elements.Prerequisite
Definition	A logical (Boolean) expression indicates what other Course Elements must be complete before a student will be allowed to enter the given (Block or Assignable Unit) Course Element. If the expression evaluates true, the "prerequisites" are met, and the student user may enter the (Block or Assignable Unit) Course Element
Usage	*Course Elements.Prerequisite* does not apply to Objectives Course Elements. (Although the logical expression can reference Objectives). There shall be no more than one *Course Elements.Prerequisite* for

Data Element Name	Course Elements.Prerequisite
	each Block or Assignable Unit Course Element. The prerequisites for a Block Course Element apply to all the members of that Block. Prerequisites are additive. Individual members of a Course Element may have prerequisites in addition to the parent's prerequisites that must be met before a student may enter them. All logical expressions are Boolean (i.e. are evaluated to either true or false). Rules for interpreting logical expressions are described in *4.3.4 Logical Expressions*
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Prerequisite
In Files	Prerequisites File
Obligation	Optional
Name Format	"Prerequisite" case insensitive.
Value Format	Logical expression as described in Chapter 9.
Data type	CMILogic
Examples	A5&A6
	(A004 | A003 | A002) & B3
	3*{A1,A2,A3,A4,(B3|B4)}

3.4.18 Course Elements.Completions

Data Element Name	Course Elements.Completions
Definition	An array of data elements that define how to achieve a specific status for a Course Element, and what to do after that status is achieved. Used to force a student to follow a course sequence depending on performance in other Course Elements Each record in this array is made up of the following sub-elements: *Course Elements.Completions.Requirement* *Course Elements.Completions.Status if True* *Course Elements.Completions.Next AU if True* *Course Elements.Completions.Goto after Next*
Usage	There may be more than one *Course Elements.Completions* record for each Course Element. There may be a record for each possible status that may be achieved in a Course Element. Completions are evaluated in the order in which they appear. The first *Course Elements.Completions* record to evaluate true determines status of the Course Element and actions of the CMI system.
CMI Behavior Notes	
AU Behavior Notes	

3.4.18.1 Course Elements.Completions.Requirement

Data Element Name	Course Elements.Completions.Requirement
Definition	A logical (Boolean) expression indicating what conditions must be met before the status of a course element is modified to match the associated value of *Course Elements.Completions.Status_If_True* and any *Course Elements.Completions.Next AU if True* is launched by the CMI.
Usage	CMI system verifies that the logical expression in this field is true before setting the course element's status to the value in the associated *Course Elements.Completions.Status if True*. Also, if the expression is true then *Course Elements.Completions.Next AU if True* and *Course Elements.Completions.Goto after Next* are used to direct the student to the specified AUs. If the logical statement in Course Elements.Completions.Requirement does not evaluate to true then the course element's *Core.Lesson Status* is not changed by the current completion rule and *Course Elements.Completions.Next AU if True* is ignored along with Course *Elements.Completions.Goto after Next*. All logical expressions are Boolean (i.e. are evaluated to either true or false). Rules for interpreting logical expressions are described in *4.3.4 Logical Expressions*
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Requirement
In Files	Completion Requirements File
Obligation	Optional
Name Format	"Requirement" case insensitive.
Value Format	Logical expression as described in Chapter 9.
Data type	CMILogic
Examples	A5&A6
	(A004 | A003 | A002) & B3
	3*{A1,A2,A3,A4,(B3|B4)}

3.4.18.2 Course Elements.Completions.Status if True

Data Element Name	Course Elements.Completions.Status if True
Definition	The new status value that the Course Element is set to if the logical expression in *Course Elements.Completions.Requirement* evaluates as true.
Usage	When the logical expression in *Course Elements.Completions.Requirement* evaluates as true, the following things are done by the CMI: 1. The status of the Course Element is set to the value of *Course Elements.Completions.Status if True*. 2. If the Course Element is an AU, the value of *Core.Lesson Status* will be set to the value of *Course Elements.Completions.Status if True* for that AU at launch time.

Data Element Name	Course Elements.Completions.Status if True
	If this data element is left empty, then the status of the Course Element is computed by the "default status setting behaviors". The default status setting behaviors are as follows: 1. **AU Course Element -** If the course element is an AU, either the value returned by the AU in *Core.Lesson Status* will be used or the CMI will determine a status based on *Core.Lesson Score* and *Student Data.Mastery Score* rules.(See *Student Data.Mastery Score*) 2. **Block or Objective Course Element** – If the course element is a Block or an Objective, then status is determined by the status of all the course elements listed in *Course Elements.Completions-.Requirement*. If all of these evaluate to "complete", then the course element's status evaluates to "complete" otherwise the course element in question is "incomplete".
CMI Behavior Notes	
AU Behavior Notes	
File Binding	
Name	Result
In Files	Completion Requirements File
Obligation	Optional
Name Format	"Result" case insensitive.
Value Format	See description for data type *CMIVocabularyINI:Status*
Data type	CMIVocabularyINI:Status
Examples	Passed
	N
	F

3.4.18.3 Course Elements.Completions.Next AU if True

Data Element Name	Course Elements.Completions.Next AU if True
Definition	Identifier of the student's next assignable unit if the logical expression in Course Elements.Completions.Requirement evaluates true.
Usage	Force a student to follow a sequence (of AU's) without seeing any options. Link two or more assignable units together seamlessly.
CMI Behavior Notes	When this data element exists, the next AU shall be launched automatically without allowing the student to see any CMI menu screens. The AU launch shall take place regardless of prerequisites for the Next AU.
AU Behavior Notes	
File Binding	
Name	Next
In Files	Completion Requirements File
Obligation	Optional
Name Format	"Next" case insensitive.
Value Format	See description of data type CMISIdentifier
Data type	CMISIdentifier
Examples	A15
	A023
	A002

3.4.18.4 Course Elements.Completions.Goto after Next

Data Element Name	Course Elements.Completions.Goto after Next
Definition	Identifier of the student's assignable unit after finishing "Course elements.Next AU if True".
Usage	Force a student to return to the original assignable unit (Course Elements.ID) after a remedial unit. Force a sequence of 3 assignable units.
CMI Behavior Notes	When this data element exists, the Goto after Next AU shall be launched automatically without allowing the student to see any CMI menu screens. The launch shall take place regardless of the prerequisites for the "Goto after Next" AU.
AU Behavior Notes	
File Binding	
Name	Return
In Files	Completion Requirements File
Obligation	Optional
Name Format	"Return" case insensitive.
Value Format	System ID for an Assignable Unit.
Data type	CMISIdentifier
Examples	A15
	A023
	A002

3.5 Levels of Complexity

This specification defines 5 levels of complexity in describing a course structure. This section describes each course level. Each level is a grouping of course structure functionality (support of course data model elements). The course levels defined in this specification are as follows:

Level 1
This is the simplest level. It describes the contents of the course, assignable units. It also defines the course structure in terms of assignable units and blocks. It allows the construction of a course hierarchy. The order in which the student may go through the course is only implied with the structure. This description cannot force any order on the student user.

Includes all data elements defined as Mandatory. May include additional data elements defined as level 2, 3a, or 3b. (See section 3.5.1 for data elements included in this level)

Level 2
This level of complexity adds a possible single prerequisite for each structure element -- an assignable unit or a block. The evaluation of each prerequisite – true or false – is done by default. The order in which the student moves through the course can be affected by these prerequisites (see *Course Elements.Prequisite*).

This level also introduces the ability to identify simple completion requirements. This means a structural element's completion status can affect another element. This concept enables (among other things) the use of separate assignable units as pre-tests. Thus the completion of one assignable unit (such as a pre-test) can result in the "Pass" status of another unit (such as an instructional lesson).

Includes all information (data elements) defined as Level 1 and 2. May include additional data elements defined as level 3a or 3b. (See section 3.5.1 for data elements included in this level)

Level 3a
Level 3a adds to level 2 the ability to define complex prerequisites and complex completion requirements. Logical expressions (see section 4.2.3) may be used to describe these requirements. Completion requirements may be used to force assignable unit sequences without breaks between each.

Includes all information (data elements) defined as level 1, 2, and 3a. May include additional data elements defined as level 3b. (See section 3.5.1 for data elements included in this level)

Level 3b
Level 3b adds the description and use of objectives to the course description and sequencing information. It includes the description of the relationship of objectives to the course structural elements.

Includes all information (data elements) defined as level 1, 2, and 3b. May include additional data elements and features defined as level 3a. (See section 3.5.1 for data elements included in this level)

Level 3
Includes all information and features defined as level 1, 2, 3a, and 3b. Supporting 3a and 3b allows the use of complex prerequisites and completions with objectives. (See section 3.5.1 for data elements included in this level)

3.5.1 Course Level Mapping

The table below depicts the mapping of course levels to course data elements. The "Course level" column depicts at which level a data element is added (See notes below for exceptions). Most levels are additive - see section 3.5 for a description of each level.

Course Structure Data Element	Section	Course Level	Notes
Course	3.1	1	
Course.Creator	3.1.1	1	
Course.ID	3.1.2	1	
Course.System	3.1.3	1	
Course.Title	3.1.4	1	
Course.Level	3.1.5	1	
Course.Max Fields CST	3.1.6	1	
Course.Max Fields ORT	3.1.7	3b	
Course.Total Aus	3.1.8	1	
Course.Total Blocks	3.1.9	1	
Course.Total Objectives	3.1.10	3b	
Course.Total Complex Objectives	3.1.11	3b	
Course.Version	3.1.12	1	
Course Behavior	3.2	1	
Course Behavior. Max Normal	3.2.1	1	
Course Description	3.3	1	
Course Elements	3.4	1	
Course Elements.System ID	3.4.1	1	
Course Elements.Developer ID	3.4.2	1	
Course Elements.Title	3.4.3	1	
Course Elements. Description	3.4.4	2	
Course Elements.Type	3.4.5	2	
Course Elements.Command Line	3.4.6	1	
Course Elements.File Name	3.4.7	1	
Course Elements.Mastery Score	3.4.8	2	
Course Elements.Max Score	3.4.9	2	
Course Elements.Max Time Allowed	3.4.10	2	
Course Elements.Time Limit Action	3.4.11	2	
Course Elements.Development System	3.4.12	2	
Course Elements.Launch Data	3.4.13	1	
Course Elements.Web Launch Parameters	3.4.14	1	
Course Elements.AU Password	3.4.15	2	
Course Elements.Members	3.4.16	1	
Course Elements.Members.System ID	3.4.16.1	1	
Course Elements.Prerequisite	3.4.17	2, 3b	#1
Course Elements.Completions	3.4.18	2	
Course Elements.Completions.Requirement	3.4.18.1	2, 3a, 3b	#2
Course Elements.Completions.Status if True	3.4.18.2	2	
Course Elements.Completions.Next AU if True	3.4.18.3	2	
Course Elements.Completions.Goto after Next	3.4.18.4	2	

Notes

Course Elements.Prerequisite (Note #1)

Level 3b - Complex logic statements with objective references shall be supported.

Course Elements.Completions.Requirement (Note #2)

Level 2 - Only support for simple completion requirements is required.

Level 3a - Logic statements to define completion requirements shall be supported (see chapter 4.0).

Level 3b - Complex logic statements with objective references shall be supported. (see chapter 4.0).

4.0 Assignable Unit Sequencing within a Course

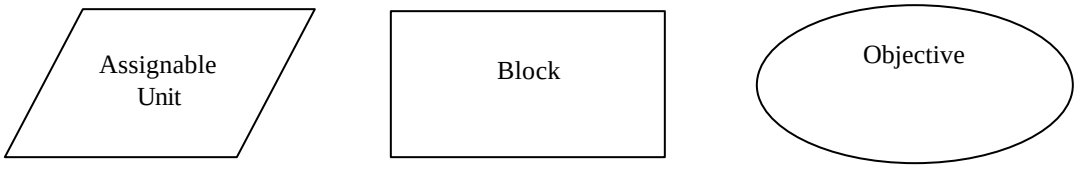
The course structure data model (see chapter 3.0) provides information on the sequencing of the assignable units in the course. This is not intended to limit the sequencing options of any CMI system. It does provide a description of sequencing (in a course structure definition) that can exported for use in other conforming CMI systems (that support the data elements used). This chapter provides further information on the usage of course data elements (in Chapter 3.0) for assignable unit sequencing in a course. There is only one binding for the course data model (the file binding - see chapter 8.0) all examples in this chapter use this binding.

4.1 Structure

Behavior of a course is based on how it is structured. This specification assumes there is a known world of course components, called Course Elements. There are three kinds of Course Elements:

1. Assignable Units: represented graphically as a parallelogram (shown below)
2. Blocks: represented graphically as rectangles (shown below)
3. Objectives: represented graphically as ovals (shown below)

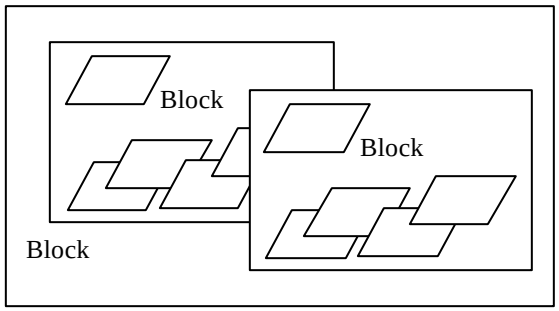
If a component can be selected and launched by a CMI, it is called an Assignable Unit (AU).



Assignable Unit, Block and Objective

Every Course Element has a unique identifier assigned by the CMI system. This identifier is called the "system identifier" and is only unique for a given course. See *Course Elements.System ID* for a description of system identifiers.

AU's can be grouped into blocks. Blocks in turn, can be grouped into other blocks, and so forth This ability to group AU's and Blocks offers the ability to organize a course into logical sections or units.



Blocks and AU's

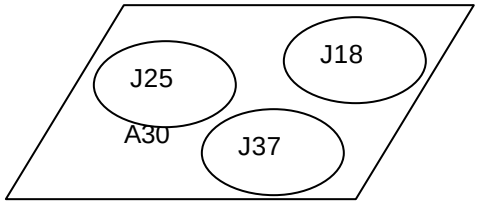
Objectives can be associated with Assignable Units and Blocks. Objectives can be associated with a single AU or block, or with many.

In the illustration below, one objective is associated with two assignable units. This means that the status of the objective depends upon the status of the two assignable units with which it is associated.

The way this relationship would appear in an Objectives Relationships File is shown here.

```
Course_Element, Member, Member
…
J20,              A21,      A22
```

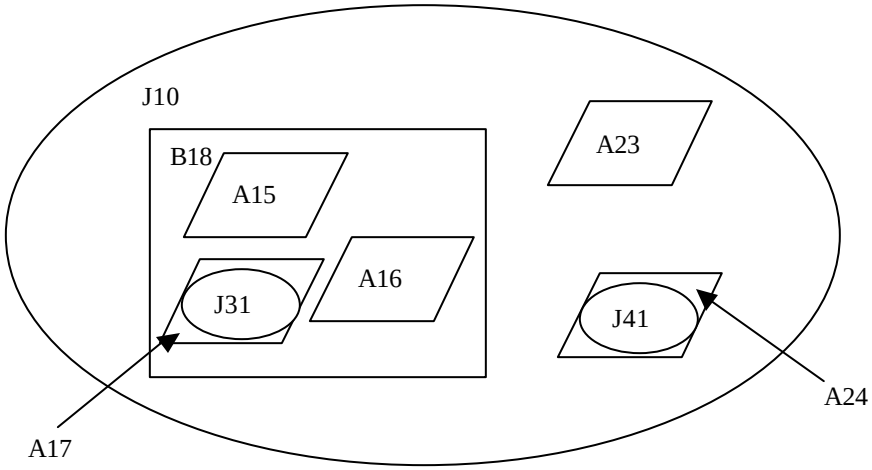
In the illustration below, there are three objectives covered in a single assignable unit.



The way this relationship would appear in an Objectives Relationships File is shown here.

```
Course_Element, Member, Member, Member
…
A30,              J18,      J25,      J37
```

In the illustration below, one objective (J10) is related to a block, several assignable units, and other objectives. Exactly how the objective is related is unclear from the illustration. However, the Course Description data model allows explicit relations to be identified.



The way this relationship would appear in an Objectives Relationships File is shown here.

```
Course_Element, Member, Member, Member
J10,            B18,    A23,    J41
B18,            A15,    A16,    A17
A17,            J31
A24,            J41
```

A course is therefore made up of blocks, assignable units, and objectives.

While course structure is a tool for organizing learning content, it does not provide a lot of sequencing information. For instance, the course in the illustration below could begin with any of the AU's.

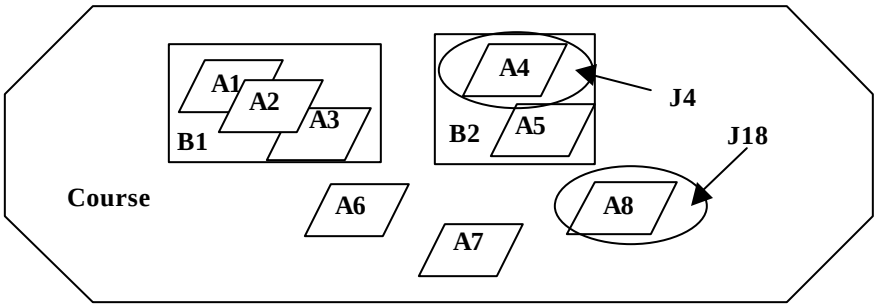


Figure 4.1-1 Course with Members Identified

The AICC course structure is described in the data model. It can also be described in a table. By default, the implied sequence of the elements in the course is from top to bottom in the data model, and from left to right and top to bottom of the table. If a more complex sequence is desired, sequencing rules must be used.

Course Structure Example:

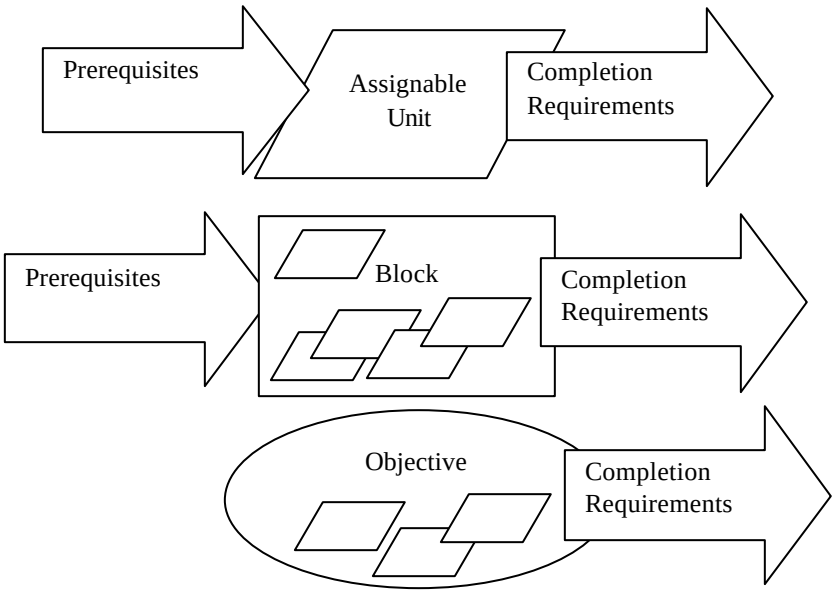
```
"Block","Member","Member","Member"
ROOT,B1,B2,A6,A7,A8
B1,A1,A2,A3
B2,A4,A5
```

4.2 Sequencing

The sequencing with in course structure is primarily defined by the following items:

- **Prerequisites**. Prerequisites are requirements that must be satisfied by a student before entering a new AU or Block. Prerequisites are defined in terms of completion statuses of AU's, Blocks, and Objectives. This is the main tool for sequencing decisions of the CMI.

- **Completion Requirements**. Completion Requirements refers to what is required in order to consider a Block or AU finished. More specifically, what is required to obtain a given status of a Course Element (AU, Block, or Objective).



Requirements for Sequencing

4.2.1 Course Element Status

The status of course elements is used in determining prerequisites and completion requirements. The status of a course element can be one of the following:

- passed
- failed
- completed
- incomplete
- browsed
- not attempted

The status can be defined explicitly, or be calculated by default. The status of an assignable unit is normally determined by the AU and reported to the CMI system via *Core.Lesson_Status.* The default status of a block is complete when all of its members are complete. It is passed when all of its members are passed. It is complete when some of its members are passed and the rest are complete. It is incomplete as long as a single member is not passed or complete. The default status of an objective is determined the same way. The objective is incomplete as long as a single member is not passed or complete.

Course Element Status	Member Status
Passed	All passed
Completed	All completed
Completed	One or more passed, the rest completed
Failed	One or more failed
Incomplete	One or more Incomplete
Incomplete	One but not all Not Attempted
Browsed	All members Browsed
Not Attempted	All members Not Attempted

Default Status for Blocks

4.2.2 Data Model Sequencing Elements

The following table lists all of the data model elements that are related to defining sequencing in a course structure. .

Data Elements	Section
Course Elements.System_ID	3.4.1
Course Elements.Members	3.4.15
Course Elements.Prerequisite	3.4.16
Course Elements.Completions	3.4.17
Course Elements.Completions.Requirement	3.4.17.1
Course Elements.Completions.Status if True	3.4.17.2
Course Elements.Completions.Next AU if True	3.4.17.3
Course Elements.Completions.Goto after Next	3.4.17.4

Data Elements Related to Sequencing

4.2.3 Logical Expressions

Some course sequencing depends upon "logical expressions". This section describes the logical expressions that may be used in the AICC data model for Course Interchange.

A logical expression is a list of one or more *Course Elements.System_ID's* combined with logical operators (see section 4.3.3.1). Logical expressions are Boolean (evaluated to either true or false) statements. The values of *Course Elements.Completions.Requirement* (see section 3.4.16) and *Course Elements.Prerequisite* (see section 3.4.17.1) are logical expressions. A logical expression containing only a single *Course Elements.System_ID* is a "simple logical expression". A logical expression containing one or more logical operators is a "complex logical expression".

Each *Course Elements.System_ID* listed in a logical expression is evaluated to either true or false depending upon the status (see section 4.3.1) of its associated course element. The table below shows how statuses are mapped to true or false by "default" (i.e. the absence of a *Course Elements.Completions.Requirement* for the given course element listed in the logical expression):

Course Element's Status	Evaluates to
passed	True
completed	True
failed	False
incomplete	False
browsed	False
not attempted	False

Logical operators are used to form **complex logical expressions**. The table below defines the allowed logical operators for complex logical expressions.

Operator	Symbol	Definition
and	&	All elements separated by an "&" (ampersand) must be complete (i.e. true) for the expression to be evaluated as complete. **A34 & A36 & A38** Assignable units number 34, 36, and 38 must all be completed or passed (i.e. "true") for the group to be considered complete.
or	|	If any of the elements separated by an | are "true" the expression is considered true. **A34=P | A36=P | A38=P** If any one of the Assignable Units, 34, 36, or 38, are passed then the expression is considered True.
not	~	An operator that returns false if the following element or expression evaluates true. It returns true if the following element or expression evaluates as false. **~A35** This expression is false if Assignable Unit 35 is Passed or Completed. This expression is true if AU 35 is Incomplete, Not Attempted, Failed, or Browsed.
equals	=	Used in a logical statement in the following manner: <Course Element>**=<**status value> Evaluates to true when a course element (on the left side of the sign) has the same status value (see section 4.3.3) as the one indicated on the right side of the equals sign. For example: **A35=P** If assignable unit A35's status is passed, then the statement evaluates to true otherwise it is false.
group or 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 describing prerequisites or completion requirements. A set has no effect on the structure of the course. For example: **{A34, A36, A37, A39}** Assignable units 34, 36, 37, and 39 are part of a set.
separator for set members	,	The comma is used to separate the members of a set. Each member of the set can be evaluated as a Boolean element – true or false. For example: **{A34, A36, A37, A39}** Assignable units 34, 36, 37, and 39 are each separated by a comma in this set.
complete X number out of a set	X*{ }	X is an integer number. This operator means that X or more members of the set that follows must be evaluated as true for the entire set to be evaluated true. **"3*{A34, A36, A37, A39}"** Any three or more of the following units – 34, 36, 37, 39 – must be Passed or Completed before the expression can be evaluated as true.
evaluate first	( )	The expression inside the parenthesis ( ) must be evaluated before combining its results with other parts of the logical statement. Parentheses may be nested. **"A34 & A35 | A36"** In this expression, completing A36 all by itself enables an evaluation of true. **"A34 & (A35 | A36)"** Adding parentheses makes it necessary to complete at least two units (A36 all by itself is no longer enough) to evaluate the expression as true.

Operator Precedence

Logical operators within are logical expression are evaluated in a specific order. The order of precedence is defined in the table below

Operator	Order of Precedence
=	1
( )	2
*{ }	3
~	4
&	5
|	6

Examples:

Example 1

```
A18
```

If this AU System Id appears in a logic statement, it evaluates as true if the course element A18 status is passed or completed.

Example 2

```
A18=P
```

If this expression appears in a logic statement, it evaluates as true only if the AU status is passed.

Example 3

```
A18=browsed
```

This expression evaluates as true only if the AU has a status of browsed.

Example 4

```
A23 & A28
```

Evaluates true if Both AU 23 and AU 28 have a status of passed or completed.

Example 5

```
(A23=p | A23=c) & (A28=p | A28=c)
```

Evaluates exactly the same as example 4.

Example 6

```
3*{A23, A25, A26, A28, A29}
```

Evaluates as true if three or more of the five members of the set of assignable units has a status of passed or completed.

Example 7

```
3*(A23=p, A25=p, A26=p, A28=p, A29=p}
```

Evaluates as true if three or more of the five members of the set of assignable units has a status of passed. A completed AU now evaluates as false.

Example 8

```
~A15
```

Evaluates as false with a status of passed or completed. Evaluates as true with a status of incomplete, not attempted, browsed, or failed.

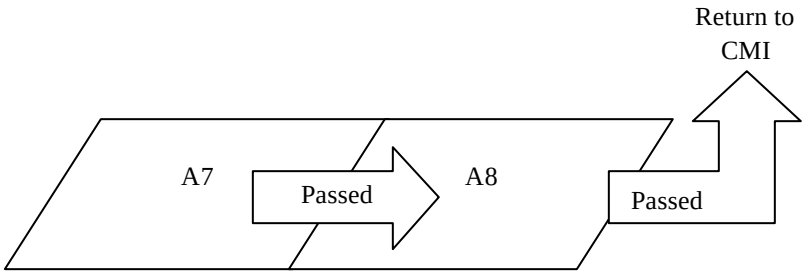
Example 9

```
~(A31=F)
```

Evaluates as true if A31 has a status of passed, browsed, not attempted, completed, or incomplete. Evaluates false if A31 is failed.

4.3 Completion Requirements

Completion requirements fall into two categories, simple and complex. Simple requirements contain only a single course element system identifier or simple logical expression (e.g. "A002") as the value for *Course Elements.Completions.Requirement*. Complex requirements contain a complex logical expression (e.g. "A003&A004") in the *Course Elements.Completions.Requirement*.



Seamless Linking of Assignable Units

In the figure above, the linking of AU number 7 and 8 is shown. The *Course Elements.Completions.Requirement* is stated as a single assignable unit. When the status of the AU is not made explicit with an equals sign (=), the AU evaluates as true whenever its status is Passed or Completed. In this case, the **Status if True** of the AU reporting back to the CMI with a status of Passed or Completed, is that the CMI assigns a status of Passed. The **Next AU if True** data element indicates that as soon as A7 achieves a status of passed, the CMI will automatically launch A8. When A8 is passed, the student will return to the course menu.

A Completion Requirements File would include the following lines.

```
Structure_Element, Requirement,              Result, Next, Return
A7,                A7=Passed | A7=Completed, Passed, A8
```

In the Completion Requirements File, the record for A8 is totally superfluous, because the default behavior is to return to the CMI anytime that a student leaves a AU for any reason.

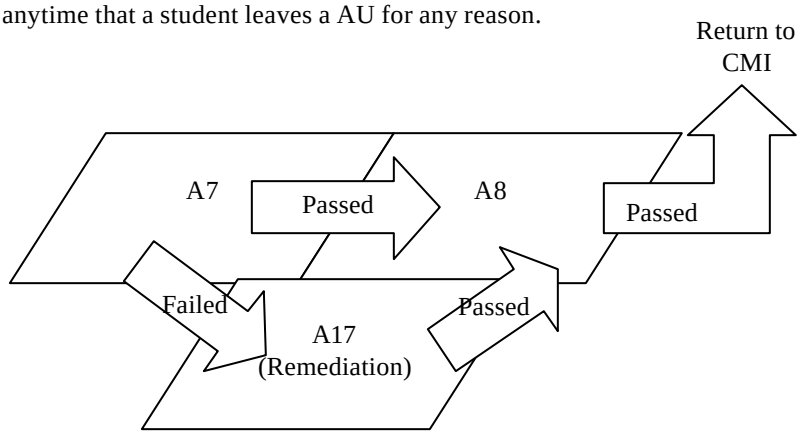


Figure 4.3-1 More Seamless Linking

Now assume that there is a remedial AU called "A17". If the student fails A7, he should immediately begin the remedial AU. After the student passes A17, he should then move seamlessly into A8.

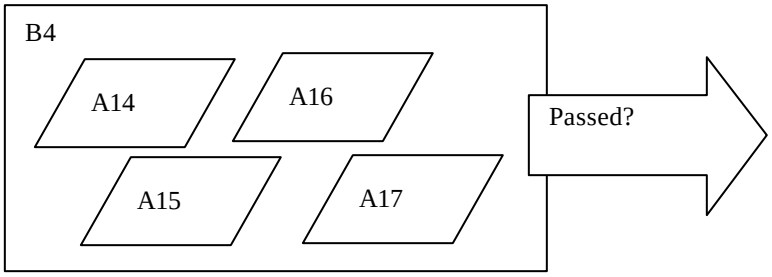
Lines in the Completion Requirements File would include the following

```
Structure_Element, Requirement,              Result, Next, Return
A7,                A7=Failed,                Failed, A17,
A7,                A7=Passed,                Passed, A8,
A17,               A17=Passed,               Passed, A8,
```

What happens if the student fails A17? There is no defined "next" for this status, so the student would return to the CMI, which is the default behavior. What the CMI does after the student fails the remedial AU is not defined here. In fact, what the CMI does upon student failure of the remedial AU may not be defined in any of the sequencing rules accompanying the course..

4.3.1 Complex Completion Requirements

Complex requirements are those with a complex logical expression (see section 4.3.3). Complex logical expressions are useful in defining when a block is complete. Assume there is a block, B4, with four assignable units, A14, A15, A16, and A17.



A Typical Block

By default, B4 is considered passed when all of its members are passed. This is defined explicitly in the file fragments below.

AICC Completion Requirements File fragment.

```
Structure_Element, Requirement,                  Result, Next, Return
B4,                A14=P & A15=P & A16=P & A17=P,Passed,,
```

By adding an additional line to the Completion Requirements file, the default for completed can also be explicitly expressed.

```
Structure_Element, Requirement, Result, Next, Return
B4,A14=P & A15=P & A16=P & A17=P,Passed,,
B4,(A14=P | A14=C)&(A15=P | A15=C)&(A16=P | A16=C)&(A17=P | A17=C), C,,
```

The computer shall evaluate Completion Requirements in the order in which they appear in the file. Consequently, the two completion requirements together express the default behaviors for Passed and Completed.

When a Course Element appears without an equal sign in the requirement field, it is evaluated as true when its status is Passed or Completed.

4.3.2 Completion Requirements - Rules of Execution

The following section lists the CMI requirements for the execution of complete requirements rules associated with a course structure. The order of completion requirements rule execution and when completion requirements rule execution occurs are described.

The CMI requirements for Completion Requirements (rules) execution are as follows:

1. Each record in the Completion Requirements (CMP) table defines a completion requirement rule for a course element.

2. A course element may have multiple rules (records) associated with it.

3. CMP rules for a course element override the default rules for status setting behavior.

4. CMP rules are order dependent. The order in which each rule appears in the CMP table must be preserved by the CMI. The CMP table rules must be evaluated in that order by the CMI.

5. A CMP rule "fires" (updates a course element's status and/or automatically launches an AU) when its associated REQUIREMENTS field expression evaluates to "true".

6. The CMP rules are evaluated in a "single-pass" from the beginning of the table. There is no recursion of rules (evaluation of the current rule does not trigger evaluation of any rules associated with the course elements in the REQUIREMENTS expression for the current rule). The rules evaluation "pass" will continue to the end of the CMP table (unless an AU is automatically launched).

7. CMP rules are evaluated when a student launched AU exits or when a NEXT AU or a NEXT/RETURN AU sequence terminates. If a rule "fires" and results in the automatic launch of a NEXT AU, followed by a RETURN AU. Then the CMP rules are evaluated after the RETURN AU exits (NOT when the NEXT AU exits)

8. If no NEXT/RETURN sequence is invoked during a "pass", the CMP rules will continue to be evaluated until the end of CMP records are reached. The CMP rules will not be evaluated again until another AU is launched (by the student) and terminated.

9. If a CMP rule automatically launches an AU or a sequence of two AU's, rule evaluation is halted until the associated AU(s) have been sequentially launched and terminated. When the automatically launched AU(s) have terminated, rule evaluation will restart from the beginning of the CMP table.

10. If a course element does have multiple rules (records) associated with it, only the first one to evaluate to "true" is allowed to "fire" during a "pass". All subsequent rules for the same course element are ignored during the remainder of the rules evaluation "pass".

11. When evaluating the "Requirement" field to determine if a rule fires, the current status of all referenced course elements are to be used. Status changes due to rules that fired earlier in the same "pass" through the CMP rules are included in rule evaluation of subsequent rules. (i.e. status changes caused by rule 1 "firing" will affect rule 3 if rule 3 referenced course elements changed by rule 1)

12. A NEXT/RETURN launch sequence overrides any prerequisites defined in the PRE (Prerequisites) file. The indicated AUs must be launched by the CMI even if the student would not otherwise be allowed to launch the lessons due to unfulfilled prerequisites.

13. A rule for a given course element may reference itself in the REQUIREMENTS field. The status value used (for the self-referring course element) in rule evaluation would either be determined by the last AU launched or the previous rules evaluation "pass".

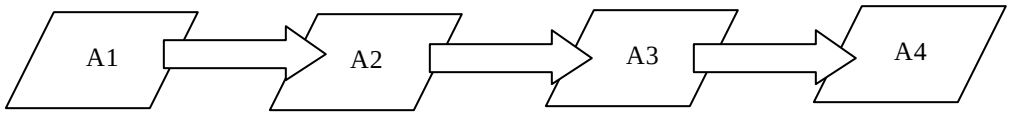
14. If the firing of a CMP rule results in a change to the status of the associated course element then the parent course element (if any) containing the changed element must have its own status re-evaluated. The parent element must be re-evaluated using the default status rules described in 4.2.1. If the re-evaluation results in a status change in the parent element then its parent must also be re-evaluated. This upward ripple of status re-evaluation must continue until a parent element is reached that does not evaluate to a different status or until there are no higher level course elements. All status re-evaluations must be completed before the next CMP rule is evaluated.

4.4 Prerequisites

Prerequisites for a given Course Element are defined in a logical expression (see section 4.2.3). If the logical expression evaluates as true, then the student may begin the Course Element, if it evaluates False, the student is prohibited from beginning the Element.

4.4.1 Simple Prerequisites

Simple prerequisites are based on the status of a single Course Element. Many fairly sophisticated course navigation schemes can be constructed with simple prerequisites. Perhaps the most common is the sequential course. Assume there are four AU's, and the developer wants them to be taken in sequence.



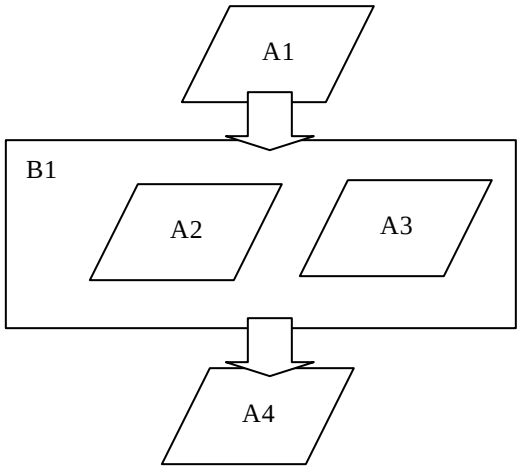
Sequential Course

The file fragments below show how the sequential path is forced on the student with prerequisite logic. There is no prerequisite for A1, but there are prerequisites for A2, A3 and A4, so A1 must be taken first. After passing or completing A1, the only AU for which the student has met the prerequisites, is A2. So A2 must be taken second. After passing A2, the only new AU for which the student is now qualified is A3. And so forth..

AICC Prerequisites File

```
Structure_Element, Prerequisite
A2, A1
A3, A2
A4, A3
```

More complex course structures may require the creation of blocks. Simple prerequisites can still be used to enforce a desired sequence. Assume there are four AUs. The first AU is an introduction that must be taken before any others. AUs A2 and A3 can be taken in any order, but AU A4 requires the completion of A2 and A3 (Block B1).



More Complex Course

AICC Course Structure File

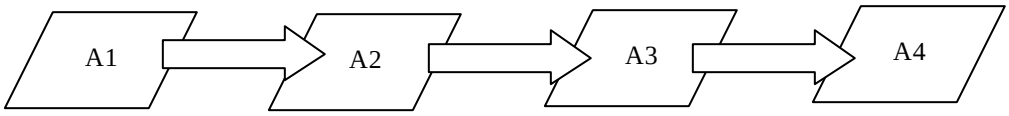
```
block, member, member, member
root, A1, B1, A4
B1, A2, A3
```

AICC Prerequisites File

```
Structure_Element, Prerequisite
B1, A1
A4, B1
```

4.4.2 Complex Prerequisites

Complex prerequisites allow the use of complex logical expressions for the prerequisite column in the Prerequisite table. As an example of some of the additional capabilities possible with complex prerequisites, return to the example of a sequential course of 4 AU's illustrated below..



Sequential Course

In this case however, assume that the course designer does not want the student to revisit any AU after it is passed. With complex prerequisites, you can force a linear sequence, and prevent the review of a previous AU. The following file fragments show how this may be done.

Notice the A1 prerequisite is that A1 not be passed. As soon as A1 is passed, the prerequisite cannot be met. The student is "locked out."

AICC Course Structure File

```
block, member, member, member
root, A1, B1, A4
B1, A2, A3
```

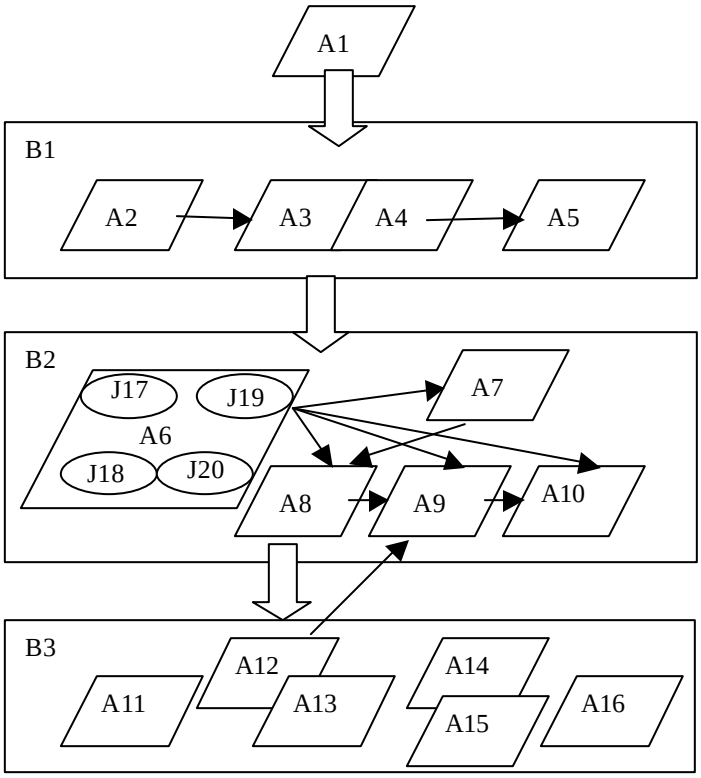
AICC Prerequisites File

```
Structure_Element, Prerequisite
A1, ~(A1=p)
A2, ~(A2=p)
A3, ~A3=p
B1, A1
A4, B1=p & ~A4=p
```

4.4.3 Complex Sequencing

The more common use of complex prerequisites is to allow complex navigation schemes to be described.

In the course shown in the figure below, the structure is reflected in the file fragments that follow.



Complex Navigation

Inside Block 1, the AUs must be taken sequentially. This can be forced with prerequisites. A3 and A4 need to be seamlessly linked together so the student takes both in a single session. This can be done with Completion Requirements,

Block B2 has B1 as a prerequisite. This means that no element in B2 can be started until all elements in B1 are Passed or Completed. There may be additional prerequisites defined for elements in B2, but their definition is always additive. For instance, A8 has A7 as a prerequisite in the table. Though not stated explicitly, this is the equivalent of a prerequisite of (A7 & B1). Similarly, A6 is not listed on the table as having any prerequisite. But because it is part of B2, it has the implicit prerequisite of B1.

Block B2 includes a pre-test -- A6. The student may select the pretest or take the learning activities in sequence starting with A7. There are four objectives in the pre-test -- J17, J18, J19, and J20. The course developer has decided that passing an objective in the pre-test allows the student to skip the AU associated with that objective. J17 is associated with A7, J18 with A8, and so forth. The Completion Requirements Table shows that these AUs are considered passed when the objectives are passed. Notice in the Completion Requirements Table that the Block (B2) is considered passed when A7 through A10 are passed. A6 status is not relevant to the completion of the block.

In Block B3, the AU's can be taken in any order. Passing or Completing any 4 of the six AU's results in passing the Block. This is shown in the Completions.Requirement. If the student fails A12, he is forced back to A9. This is shown in the Completions for B3. After completing A9, he can again take any AU in Block B3.

AICC Course Structure File

```
block, member, member, member, member, member, member
root, A1, B1, B2, B3
B1, A2, A3, A4, A5
B2, A6, A7, A8, A9, A10
B3, A11, A12, A13, A14, A15, A16
```

AICC Prerequisites File

```
Structure_Element, Prerequisite
B1, A1
A3, A2
A4, A3
A5, A4
B2, B1
A8, A7
A9, A8
A10, A9
B3, B2
```

The student must begin with A1. Taking any AU in Block 2 requires passing Block 1. Beginning any AU in Block 3 requires passing Block 2. These rules are shown in the Prerequisites File..

AICC Objectives Relationships File

```
Course_Element, Member, Member, Member, Member
A6, J17, J18, J19, J20
```

AICC Completion Requirements File.

```
Structure_Element, Requirement, Result, Next, Return
A3, A3=passed, ,A4
A7, A7=passed | J17=passed, passed
A8, A8=passed | J18=passed, passed
A9, A9=passed | J19=passed, passed
A10, A10=passed | J20=passed, passed
B2, A7=passed & A8=passed & A9=passed & A10=passed, passed
A12, A12=failed, failed, A9, A12
B3, 4*{A11, A12, A13, A14, A15, A16}, passed
```

4.5 Tracking Non-Conforming/Non-Communicating Assignable Units in a Course

Courses may have AU's that are non-conforming or non-communicating (i.e. "Dumb Content"). Such AU's have not implemented any of the existing communication bindings and do not report data to the CMI. CMI systems are required to support this type of content in course.

4.5.1 Web Environment Conformance Requirements

In web environments (Which includes HACP and API bindings), the CMI must launch all non-conforming/non-communicating AU's. Since no data is reported, the CMI requirements for setting the AU's status or other data are undefined. Such undefined behavior is CMI implementation specific.

Note that conforming AU's in the API binding may communicate but not report *Core.Lesson Status*. The behavior of the CMI with regards to determining status is also undefined (and CMI implementation specific).

Future versions of this specification may define specific behaviors for both instances.

4.5.2 File-based Conformance Requirements

In the Windows environment (file-based), the CMI must launch all non-conforming/non-communicating AU's in the course if such content can be "synchronously launched". (See section 5.3 for a description of a single-process launch).

Since no data is reported, the CMI requirements for setting the AU's status or other session data are undefined. Such undefined behavior is CMI implementation specific.

Future versions of this specification may define specific behavior for this case.

5.0 Communicating via Files (The File Binding)

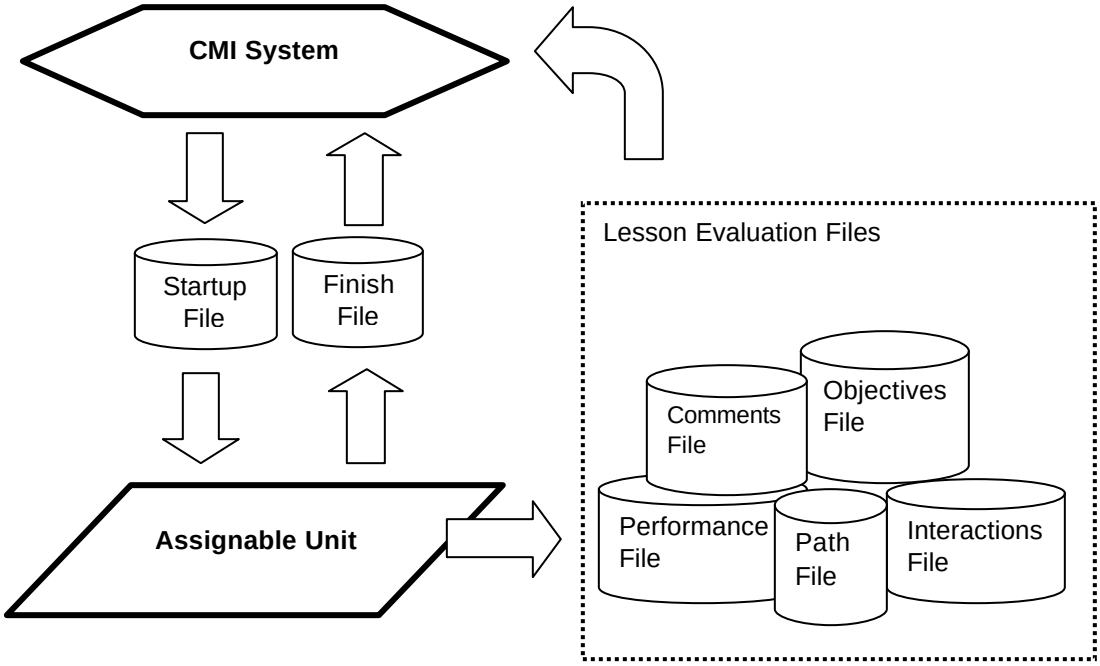
This chapter defines the File binding to the communication data model in Chapter 2 (i.e. "file-based communication"). It defines the following:

- The environment in which the File binding operates
- How the CMI launches Assignable Units (AUs)
- How the File binding is used by AUs to communicate with the CMI system.
- Conformance requirements for this binding.
- Which elements from the data model described in Chapter 2 may be used by the File binding (Including which files specific elements are located in and the format of those files).

Although many of the data elements in the communication data model have different names in the file-based communication, there are no new data elements appearing in this chapter.

5.1 Conceptual Model

In the File binding (i.e. "file-based communication"), the Assignable Unit (AU) communicates with the CMI using text files (See figure below). The CMI system writes a "Startup" file (for the AU to read), launches the AU process, suspends execution (waits) until the AU process terminates, and reads the "Finish" file created by the AU. Based on information obtained from the Startup file, the AU can obtain launch parameters, previous state information, and determine where to write its Finish file (and other output files) for the CMI to read. Since data files are used for communication, there are rules for when and where these data files are written, read, and deleted.



5.2 Operating Environment

The operating environment for this binding is the Microsoft Windows™ Operating environment. (Other operating environments may be included in the future).

5.3 Launching an Assignable Unit

The method for launching an Assignable Unit (AU) is a simple "synchronous" launch. The CMI system acts as a "Router" program and uses the operating system to launch another program (i.e. the AU), creating a new process. Immediately prior to launch, the CMI writes a "startup" data file for the AU. The CMI then launches the AU and "waits" until the AU completes execution. Upon termination of the AU's process, the CMI reads the text file(s) output from the AU and resumes execution (e.g. launches next AU assigned, refreshes menu status, etc.).

This process assumes the following:

- Both the CMI and AU programs are in files located on the local file system (either a disk volume provided by a LAN fileserver or a local disk drive.)
- Both the CMI and the AU programs are local processes running on the student's computer workstation

The launch sequence of an AU is as follows:

1. The CMI writes the Startup file to a pre-determined location (see section 5.4.1)
2. The CMI launches the AU application using the Windows CreateProcess() function or similar Windows function. (This is commonly called a Windows "command line" type launch).
3. The CMI "waits" until the AU application process has terminated. (The CMI monitors the AU process created)
4. As the AU starts up, it reads the Startup file and then immediately deletes it.
5. Prior to exit the AU writes the finish file (and other evaluation files) to locations specified in the startup file by the CMI.
6. The AU exits
7. The CMI reads the Finish file and then immediately deletes it. Other evaluation files are also read (if they exist) but are not necessarily deleted.
8. The CMI resumes execution.

AU Processes

An AU must be designed so that it can be launched in the windows (32 bit) environment using the windows CreateProcess() function (or a similar windows "command line" function). The CMI will monitor the created process to determine when the AU has terminated. The process created from this action may spawn other processes, but it must be the "primary process". The AU must close all of its other spawned processes before closing the originating process. If an AU does not do this, then the CMI may assume that the AU has terminated before it actually has.

CMI Launch example

The following Microsoft Visual Basic™ code example shows how a CMI could synchronously launch an AU using the technique described above.

Synchronous Launch example

```
' >>>> Step 1 – Use CreateProcess() Launch application <<<<<<
X = CreateProcessA(0&, cmdline$, 0&, 0&, 1&, NORMAL_PRIORITY_CLASS, 0&, working$, _
                   NameStart, NameOfProc)

' >> Step 2 - WaitforSingleObject(  ) - Wait until primary process is terminated <<
X = WaitForSingleObject(NameOfProc.hProcess, INFINITE)

' >> Step 3 – Destroy handle to the process <<
X = CloseHandle(NameOfProc.hProcess)
```

5.4 Method of Communication

Communication between the CMI and AUs is accomplished by reading/writing text files. The files used for this communication are described in the table below. There are 2 file formats used for these files *CMIFormatINI* and *CMIFormatCSV*. The table also indicates the format for each file. See data types *CMIFormatINI* and *CMIFormatCSV* (in section 9.0) for detailed descriptions of the formats used.

File	Description
Startup File	A text file written by the CMI for the AU to read at startup. Contains AU specific launch parameters, previous state information, and file locations for the AU to write output file(s). (see section 5.6.1)
Finish File	A text file containing information on student activity, performance, and AU state. Written by the AU prior to exit (see section 5.6.2). This file is the "complement" of the Startup File.
Comments File	A text file written by the AU that contains comments for the student. (See section 5.6.3)
Objectives File	A text file written by the AU that contains student performance to specific objectives (see section 5.6.4).
Path File	A text file written by the AU that records the path the student user navigated thru the AU (see section 5.6.5).
Interactions File	A text file written by the AU that contains detailed information on each student interaction measured (see section 5.6.6).
Performance File	A text file written by the AU that contains Learner performance information. (see section 5.6.7).

5.4.1 Startup File (Usage)

The Startup file is used by the CMI system to pass data to the AU. It is the only "input" file created by the CMI for the AU to read. The CMI system creates the Startup file just prior to the launch of the AU.

There are three methods available for the AU to determine the Startup file location :

1. An additional parameter containing the Startup file location/name is included in AU's command line.
2. The location of the Startup file location/name is found in the Windows environment variable "PARAM$CMI" (e.g. "PARAM$CMI=C:\Winnt\Temp\SomeStartupFileName.ext").
3. The location of the Startup file is in the "Windows directory" with a name of "PARAM.CMI" (e.g. "C:\WINDOWS\PARAM.CMI"). The Windows directory varies by workstation, it is discovered by the AU using the Microsoft Windows GetWinDir() function or by using the "windir" system environment variable.. Examples of this directory are "c:\windows" for Windows95/98 and "c:\winnt" for Windows NT, ME, XP, 2000.

The CMI system must support all 3 methods of Startup file location. Typically, most AUs use method #3.

Once the AU Application is initiated, it reads the Startup file created by the calling CMI system and then immediately deletes it.

The AU obtains the following information from the CMI via the Startup file:

- Where to write the Finish File
- Where to write lesson evaluation files (if any)
- Launch parameters
- Previous state (i.e. "Bookmarking") information
- Previous status information.

For a complete list of data elements contained in the Startup file and its format - see section 5.6.1.

5.4.2 Finish File (Usage)

The AU must create a Finish file containing data to be passed back to CMI so that the CMI system can update its student performance data (and perform any necessary display updates or routing activity). The CMI determines where the Finish file is to be written by the AU. The AU discovers this location via the communication data element *Core.Output File* which is contained in the Startup file - see section 5.6.1.

The AU writes the Finish file just prior to exit. The CMI system then reads the Finish file and immediately deletes it.

The CMI obtains the following information from the AU via the Finish file:

- Status updates
- AU session state (i.e. "Bookmarking") information to store

For a complete list of data elements contained in the Finish file and its format - see section 5.6.2.

5.4.3 Evaluation Files (Usage)

In addition to the Finish and Startup file, there is a group of optional files called the *Evaluation Files*. They are as follows:

- Comments File (see section 5.6.3)
- Objectives File (see section 5.6.4)
- Interactions File (see section 5.6.5)
- Path File (see section 5.6.6)
- Performance File (see section 5.6.7)

The following is true for each of the evaluation files:

- If the AU (and the CMI) supports the data elements contained in file, the AU will write them to the location specified in the Startup file.
- If the file already exists, the AU appends the data to that file. If the file does not exist, the file is created and the data deposited. The CMI system is responsible for management of these files.
- If the AU has the ability to create the evaluation file(s) but the CMI does not provide a file location, then the evaluation file(s) will not be written.

5.4.4 Error Conditions

To be determined.

5.5 Conformance Requirements

Conformance to the file binding may be looked at from two viewpoints, that of the Assignable Unit (AU) and that of the CMI. There are three levels of obligation described in this binding specification:

- Mandatory
- Optional
- Extension

Obligations for the AU and the CMI are different.

CMI Conformance

Mandatory means that the CMI shall read, delete, and create the indicated data file(s), properly store and use mandatory communication data elements.

Optional means that a conforming CMI may not respond at all indicated files, or optional communication data elements. A conforming CMI may support many options.

An *extension* is a file or data element that is not described in this specification. Extensions may be supported by a CMI. However, extension data elements (or files) may not perform the identical function as data elements (or files) defined in this specification; 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 CMI is non-conforming.

AU Conformance

Mandatory means that the AU shall read, delete, and create the indicated data file(s), and properly store and use the mandatory data elements.

Optional means that the AU may read or create data elements in the indicated data file(s), and properly store and use the data elements indicated as optional.

An *extension* is a file or data element that is not described in this specification. Extensions may be supported/used by an AU. However, extension data elements (or files) may not perform the identical function as data elements (or files) defined in this specification; 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 AU is non-conforming.

5.5.1 CMI Responsibilities

Launch and Communication

The CMI system shall do the following to launch an AU:

1. Write a Startup File
2. Synchronously launch the AU application (i.e. launch and "wait") using the operation system
3. Monitor the AU process until termination
4. Read the resulting Finish File
5. Delete the Finish immediately after reading it contents

The CMI shall support all 3 mechanisms of Startup file location (described in section 5.4). The CMI must support all the data elements described for this binding as mandatory (described in section 5.6). The CMI may support the optional data elements (and files). The CMI may also support extensions not defined in this specification 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 AU not using the extensions.

Sequencing
An AU assignable unit may only be launched by a CMI. An AU may not itself launch other assignable units. An assignable unit must, at a minimum, be able to:

1. Be synchronously launched (as described in sections5.3 and 5.4)
2. Read, write, and delete the required communication file(s) (as described in sections 5.3 and 5.4)

Flow control – moving from one the AU object to another – is assumed to be the responsibility of the CMI and not within the assignable unit (AU) itself. This is conceptually important because AU reuse cannot really happen if the AU has embedded information that is context specific to the course. In this context, flow control means that the decision of what AU (the AU) will next be presented to the student is made by the CMI. (This recognizes that some AU's may make decisions—that is, branch – within itself, but that kind of internal flow is hidden from the CMI.)

The determination of which AU(s) the student is routed to is determined solely by the CMI and is defined in large part by the Course Structure description (Chapter 3). Chapter 3 defines information about the AU that is context specific to the course (e.g., the default sequence of AU's, and prerequisites or completion requirements that might alter the delivery path.)

5.5.2 Assignable Unit (AU) Responsibilities

Launch and Communication
An assignable unit must, at a minimum, be able to do the following:

1. Have the ability to be synchronously launched (as described in sections5.3 and 5.4)
2. Read (and delete) the Startup File, and write the Finish file(as described in sections 5.3, 5.4, and 5.6)
3. Support all the following communication data elements (listed in the tables below)

Startup File – AU mandatory data elements

Group Name or Keyword	Communication Data Model Name	Section
[Core]	Core	2.1
Output_File	Core.Output File	2.1.3

Finish File – AU mandatory data elements

Group Name or Keyword	Communication Data Model Name	Section
[Core]	Core	2.1
Lesson_Location	Core.Lesson Location	2.1.4
Lesson_Status	Core.Lesson Status	2.1.6
Score	Core.Score	2.1.10
Time	Core.Session_Time	2.1.12

Sequencing
An AU may not itself launch other assignable units

5.6 Communication Data Model Mapping

This section contains the mapping of the communication data model elements (defined in section 2.0) to the file (file-based communication) binding. The contents of these files are defined in this section. The files used for communication are as follows:

- Startup File
- Finish File
- Comments File
- Objectives File
- Path File
- Interactions File
- Performance File

The following is defined for each of the above files:

- A description of the file's purpose
- A list of communication data model elements used
- The file's data format
- An example

5.6.1 Startup File

Purpose
The Startup file is used by the CMI system to pass data to the AU. (See section 5.4.1).

Data Model Elements
The following table describes the Group and Keywords used by the Startup file with corresponding data model names, references, and Mandatory/Required designations. For specific usage of a data element refer to the corresponding section in the Chapter 2.0 *Communication Data Model*. Note that ***n*** indicates an array index.

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
[Core]	Core	2.1	Mandatory
Student_ID	Core.Student Id	2.1.1	Mandatory
Student_Name	Core.Student Name	2.1.2	Mandatory
Output_File	Core.Output File	2.1.3	Mandatory
Lesson_Location	Core.Lesson Location	2.1.4	Mandatory
Credit	Core.Credit	2.1.5	Mandatory
Lesson_Status	Core.Lesson Status	2.1.6	Mandatory
	Core.Entry	2.1.8	Mandatory
Path	Core.File Path	2.1.9	Mandatory
Score	Core.Score	2.1.10	Mandatory
	Core.Score.Raw	2.1.10	Mandatory
	Core.Score.Max	2.1.10	Mandatory
	Core.Score.Min	2.1.10	Mandatory
Time	Core.Total_Time	2.1.12	Mandatory
Lesson_Mode	Core.Lesson Mode	2.1.13	Optional
[Core_Lesson]	Suspend Data	2.1	Mandatory
[Core Vendor]	Launch Data	2.3	Mandatory
[Comments]	Comments From LMS	2.6	Optional
[Evaluation]	Evaluation	2.7	Optional
Course_ID	Evaluation.Course ID	2.7.2	Optional
Comments_File	Evaluation.Comments_File	2.7.1	Optional
Interactions_File	Evaluation.Interactions_File	2.7.3	Optional
Objectives_Status_File	Evaluation.Objective_Status_File	2.7.4	Optional
Path_File	Evaluation.Path_File	2.7.5	Optional
Performance_File	Evaluation.Performance_File	2.7.6	Optional

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
[Objectives_Status]	Objectives	2.8	Optional
J_ID.***n***	Objectives.ID	2.8.1	Optional
J_Score.***n***	Objectives.Score	2.8.2	Optional
	Objectives.Score.Raw	2.8.2	Optional
	Objectives.Score.Max	2.8.2	Optional
	Objectives.Score.Min	2.8.2	Optional
J_Status.***n***	Objectives.Status	2.8.3	Optional
[Student_Data]	Student Data	2.9	Optional
Attempt_Number	Student Data.Attempt Number	2.9.1	Optional
Mastery_Score	Student Data.Mastery Score	2.9.3	Optional
Max_Time_Allowed	Student Data.Max Time Allowed	2.9.3	Optional
Time_Limit_Action	Student Data.Time Limit Action	2.9.4	Optional
	Student Data.		
Lesson_Status.***n***	Student Data.Sessions Journal.Lesson Status	2.9.7.2	Optional
Score.***n***	Student Data.Sessions Journal.Lesson Score	2.9.7.1	Optional
	.Raw	2.9.7.1	Optional
	.Max	2.9.7.1	Optional
	.Min	2.9.7.1	Optional
[Student_Demographics]	Student Demographics	2.13	Optional
City	Student Demographics.City	2.13.1	Optional
Class	Student Demographics.Class	2.13.2	Optional
Company	Student Demographics.Company	2.13.3	Optional
Country	Student Demographics.Country	2.13.4	Optional
Experience	Student Demographics.Experience	2.13.5	Optional
Familiar_Name	Student Demographics.Familiar Name	2.13.6	Optional
Instructor_Name	Student Demographics.Instructor Name	2.13.7	Optional
Job_Title	Student Demographics.Title	2.13.12	Optional
Native_Language	Student Demographics.Native Language	2.13.8	Optional
State	Student Demographics.State	2.13.9	Optional
Street_Address	Student Demographics.Street Address	2.13.10	Optional
Telephone	Student Demographics.Telephone	2.13.11	Optional
Years_Experience	Student Demographics.Years Experience	2.13.13	Optional
[Student_Preferences]	Student Preference	2.1	Optional
Audio	Student Preference.Audio	2.10.1	Optional
Language	Student Preference.Language	2.10.2	Optional
Lesson_Type	Student Preference.Lesson Type	2.10.3	Optional
Speed	Student Preference.Speed	2.10.4	Optional
Text	Student Preference.Text	2.10.5	Optional
Text_Color	Student Preference.Text Color	2.10.6	Optional
Text_Location	Student Preference.Text Location	2.10.7	Optional
Text_Size	Student Preference.Text Size	2.10.8	Optional
Video	Student Preference.Video	2.10.9	Optional
Window.1	Student Preference.Windows	2.10.10	Optional

File Format

The Startup file is text formatted as datatype *CMIFormatINI.* (see section 9.0 - Datatypes)

Example

An example of a typical Startup file is show below

Startup File example

```
;
; Startup File
;
[Core]
     ; Comment
   Student_ID = XYZ_1234
   Student_Name = Hyde, Jackson Q.
   Output_File = C:\Windows\Temp\outparam.cmi
   Lesson_Location = 45
   Credit = CREDI
   Lesson_Status = INCOMPLETE
   Score =
   Time = 0000:04:30.34
   Lesson_Mode = Normal
```

Startup File example

```
; Core_lesson is free-form group
[CORE_LESSON]

my lesson state data – 11111111111111111110000000000000000000000000

11111111111111111110000000000000000000000000 – end my lesson state data
; Core_vendor is also a free-form group
[Core_Vendor]

My Start up parameters
45,67,78,RR
End of My startup parameters

[Evaluation]
Course_ID = {}

Comments_File
Interactions_File
Objectives_Status_File
Path_File
Performance_File

[Student_Data]
Mastery_Score = 100
```

5.6.2 Finish File

Purpose
The Finish file is used by the AU to pass data to the CMI. (See section 5.4.2).

Data Model Elements
The following table describes the Group and Keywords used by the Finish file with corresponding data model names, references, and Mandatory/Required designations. For specific usage of a data element refer to the corresponding section in the Chapter 2.0 *Communication Data Model*. Note that ***n*** indicates an array index.

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
[Core]	Core	2.1	Mandatory
Lesson_Location	Core.Lesson Location	2.1.4	Mandatory
Lesson_Status	Core.Lesson Status	2.1.6	Mandatory
	Core.Exit	2.1.7	Mandatory
Score	Core.Score	2.1.10	Mandatory
	Core.Score.Raw	2.1.10	Mandatory
	Core.Score.Max	2.1.10	Mandatory
	Core.Score.Min	2.1.10	Mandatory
Time	Core.Session_Time	2.1.12	Mandatory
[Core_Lesson]	Suspend Data	2.1	Mandatory
[Comments]	Comments From Learner	2.4	Optional
[Objectives_Status]	Objectives	2.8	Optional
J_ID.***n***	Objectives.ID	2.8.1	Optional
J_Score.***n***	Objectives.Score	2.8.2	Optional
	Objectives.Score.Raw	2.8.2	Optional
	Objectives.Score.Max	2.8.2	Optional
	Objectives.Score.Min	2.8.2	Optional
J_Status.***n***	Objectives.Status	2.8.3	Optional
[Student_Data]	Student Data	2.9	Optional
Tries_During_Lesson	Student Data.Tries During Lesson	2.9.5	Optional
Try_Score.***n***	Student Data.Tries.Try_Score	2.9.2	Optional
Try_Time.***n***	Student Data.Tries.Try_Time	2.9.2	Optional
Try_Status.***n***	Student Data.Tries.Status	2.9.2	Optional
[Student_Preferences]	Student Preference	2.1	Optional
Audio	Student Preference.Audio	2.10.1	Optional

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
Language	Student Preference.Language	2.10.2	Optional
Lesson_Type	Student Preference.Lesson Type	2.10.3	Optional
Speed	Student Preference.Speed	2.10.4	Optional
Text	Student Preference.Text	2.10.5	Optional
Text_Color	Student Preference.Text Color	2.10.6	Optional
Text_Location	Student Preference.Text Location	2.10.7	Optional
Text_Size	Student Preference.Text Size	2.10.8	Optional
Video	Student Preference.Video	2.10.9	Optional
Window***n***	Student Preference.Windows	2.10.10	Optional

File Format
The Finish file is text formatted as datatype *CMIFormatINI.* (See section 5.4.1 and Chapter 9.0 - Datatypes)

Example
An example of a typical Finish file is show below:

Finish File example

```
;
; Finish File
;
[Core]
   Lesson_Location = 87
   Lesson_Status = C
   Score =
   Time = 00:02:30

[CORE_LESSON]

my lesson state data – 11111111111111111110000000000000000001110000


11111111111111111110000000000011100000000 – end my lesson state data

[COMMENT]

<1><L.Slide#2> This slide has the fuel listed in the wrong units <e.1>
```

5.6.3 Comments File

Purpose
This file contains freeform feedback from the student (recorded by the AU). It is a duplicate of the [Comments] group that is passed to the CMI system in the Finish file. If a CMI system receives data from the AU in both [Comments] group and the Comments File, the CMI must save the data from the Comments File and discard the [Comments] group data.

Data Model Elements
The following table identifies the Comment File's Fields, Data Model Names, and Data Model Section reference.

CSV File Field Identifier	Communication Data Model Name	Section
Course_ID	Itemized Comments From Learner.Course_ID	2.5.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Itemized Comments From Learner.Lesson_ID	2.14
Date	Itemized Comments From Learner.Date	2.5.3
Time	Itemized Comments From Learner.Time	2.5.7
Location	Itemized Comments From Learner.Location	2.5.6
Comment	Itemized Comments From Learner.Content	2.5.1

File Format
The Comments file is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules) . All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

Example
An example of a typical Comments file is show below:

Comments File example

```
Location ,Comment ,Course_ID, Student_ID, Lesson_ID, Date, Time
Slide #6, "The color of indicator is wrong",APU101,User03,APU-START4, 2003/01/23 , 12:45:45
Slide #6, "The color of indicator is wrong",APU101,User03,APU-START4, 2003/01/23 , 12:45:45
```

5.6.4 Interactions File

Purpose
All of the items in this file are related to a recognized and recorded input from the student (recorded by the AU). Normally, the interactions recorded are student responses to a question. (See sections listed in table below for description of the data elements recording student interactions)

Data Model Elements
The following table identifies the Interactions File's Fields, Data Model Names, and Data Model Section reference.

CSV File Field Identifier	Communication Data Model Name	Section
Course_ID	Evaluation.Course_ID	2.7.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Lesson_ID	2.14
Date	Interactions.Date	2.11.3
Time	Interactions.Time	2.11.4
Interaction ID	Interactions.ID	2.11.1
Objective ID	Interactions.Objectives	2.11.2
Type Interaction	Interactions.Type	2.11.5
Correct Response	Interactions.Correct Responses	2.11.6
Student Response	Interactions.Student Response	2.11.8
Result	Interactions.Result	2.11.9
Weighting	Interactions.Weighting	2.11.7
Latency	Interactions.Latency	2.11.10

File Format
The Interactions file is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules) . All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.
Example
An example of a typical Interactions file is show below:

Interactions File example

```
"course_id","student_id","lesson_id","date","time","interaction_id","objective_id",
"type_interaction","correct_response","student_response","result","weighting","latency"
"A340ft-2","jqh085","APU1","2004/01/15","15:14:23",37,ft1016,C,A,C,W,, 00:00:3
"A340ft-2","wam016","APU1","2004/01/15","15:14:23",38,ft2223,t,t,t,,, 00:00:01
"A340ft-2","dag085","APU1","2004/01/15","15:14:23",39,ft1134,C,B,B,C,, 00:00:02
"A340ft-2","trd018","APU1","2004/01/15","15:14:23",40,ft1156,C,C,C,C,, 00:00:04
```

5.6.5 Objectives Status File

Purpose
This file contains information on how the student has performed on objectives related to the AU. The performance may be related to previous sessions in the AU, or to the student user's performance in other AU's (in the same course) related to the same objectives. These objectives are only those associated with the current launching AU, not all the objectives in the course or curriculum.

Data Model Elements
The following table identifies the Objective Status File's Fields, Data Model Names, and Data Model Section reference.

CSV File Field Idenifier	Communication Data Model Name	Section
Course_ID	Evaluation.Course_ID	2.7.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Lesson_ID	2.14
Date	Objectives.Date	2.8.4
Time	Objectives.Time	2.8.5
Objective ID	Objectives.ID	2.8.1
Score	Objectives.Score	2.8.2
Status	Objectives.Status	2.8.3
Mastery Time	Objectives.Mastery Time	2.8.6

File Format
The Objectives Status is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules) . All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

Example
An example of a typical Objectives Status file is show below:

Objectives Status File example

```
COURSE_ID , STUDENT_ID, LESSON_ID, DATE , TIME, OBJECTIVE_ID, SCORE, STATUS,
MASTERY_TIME
"MD80-2","STU1009","APU1","1994/01/15","10:14:23","APU1684",3,, "passed","00:02:37"
```

5.6.6 Path File

Purpose
To provide a mechanism to record the "paths" a student use took during AU session(s). The paths recorded are generally the order in which the student navigates through the AU. (See sections listed in table below for descriptions of the data elements recording path information)
.

Data Model Elements
The following table identifies the Path File's Fields, Data Model Names, and Data Model Section reference.

CSV File Field Identifier	Communication Data Model Name	Section
Course_ID	Evaluation.Course_ID	2.7.2
Student_ID	Core.Student Id	2.1.1

CSV File Field Identifier	Communication Data Model Name	Section
Lesson_ID	Lesson_ID	2.14
Date	Paths.Date	2.12.2
Time	Paths.Time	2.12.3
Element Location	Paths.Location ID	2.12.1
Status	Paths.Status	2.12.4
Why_Left	Paths.Why Left	2.12.5
Time_in_Element	Paths.Time in Element	2.12.6

File Format
The Path file is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules) . All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

Example
An example of a typical Path file is show below:

Path Status File example

```
course_id, student_id, lesson_id ,date, time, element_location , status ,
why_left , time_in_element
"course6","stu2310","first1","2003/06/05","14:10:31","A","P","S","00:00:24"
"course6","stu2310","first1","2003/06/05","14:10:55","E","P","S","00:01:06"
"course6","stu2310","first1","2003/06/05","14:12:01","A","I","L","00:02:24"
"course6","stu2310","first1","2003/06/05","14:13:25","B","P","S","00:00:54"
"course6","stu2310","first1","2003/06/05","14:14:19","D","P","L","00:02:40"
"course6","stu2310","first1","2003/06/05","14:16:59","E","P","S","00:03:03"
"course6","stu2310","first1","2003/06/05","14:20:02","F","P","E","00:02:12"
```

5.6.7 Performance File

Purpose
To record simulation-specific data from AU session(s) for later analysis.

Data Model Elements
Not applicable. The performance file data is developer-defined.

File Format
The Path file is text. The formatting of this text is developer-defined.

Example
Not applicable.

6.0 Communicating via HTTP (The HACP Binding)

This chapter defines the HTTP/S-based AICC/CMI Protocol (HACP) binding to the communication data model in Chapter 2.0. It defines the following:

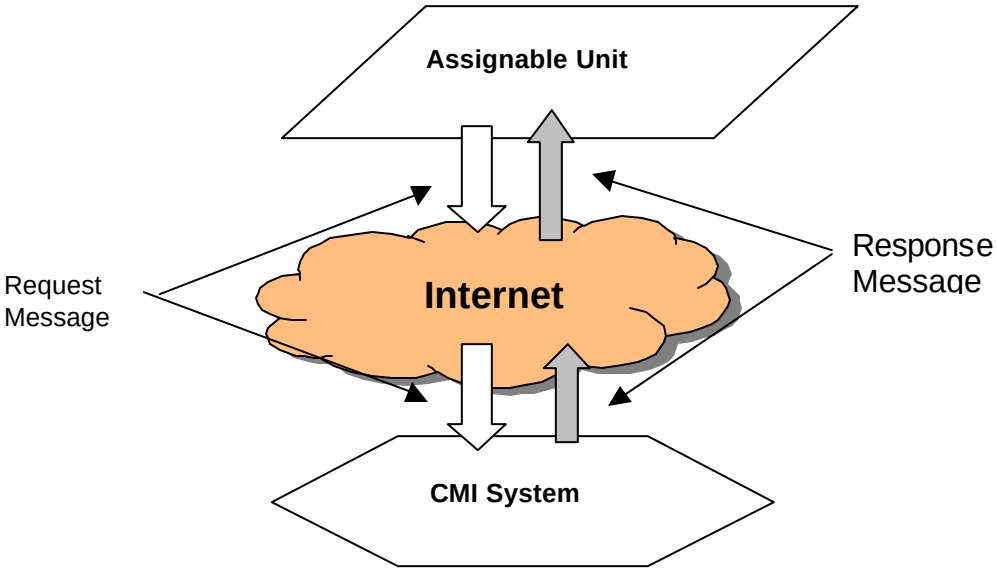
- The environment in which the HACP binding operates
- How the CMI launches Assignable Units (AUs)
- How the HACP binding is used by AUs to communicate with the CMI system. (Using HTTP/S messages)
- Conformance requirements for this binding.
- Which elements from the data model described in Chapter 2.0 may be used by the HACP binding (Including the HTTP/S messages specific elements are located in and the format of those HTTP/S messages.)

Although many of the data elements in the communication data model have different names in the HACP binding, there are no new data elements appearing in this chapter.

6.1 Conceptual Model

In the HACP binding, the Assignable Unit (AU) communicates with the CMI using a series of HTTP/S messages (See figure below). The assignable unit (AU) is launched by the CMI redirecting the web browser to a URL. The AU always initiates the communication with a message to get data or send data to the CMI. The CMI listens for and responds to message requests from the AU. For every AU "request" message there is a CMI "response" message.

HTTP is client/server protocol. There is a client program (usually a Web Browser) making requests and a server program (a Web Server) responding to the requests. With HTTP/S protocol, client and server programs may be running on the same computer or on different computers at different locations. Some portions of the CMI run as part of the Web Server (i.e. an HTTP/S server) and other portions (The student User interface) run as part of the Web Browser (This is also true for assignable units).



6.2 Operating Environment

The operating environment for this binding is the HTTP (Hyper-Text Transfer Protocol) client and server environment(s) (including Secure HTTP [a.k.a. HTTPS]). The HTTP "client" is typically a web-browser. Please refer to following document for more information about HTTP: *RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0*.

6.3 Launching an Assignable Unit

The CMI provides an interface for the learner. The CMI launches the AU by dynamically appending parameters to URL (Uniform resource locator) where the AU is located and directing the web browser to launch this specially modified URL.

This process assumes the following:

- The CMI user interface is operating within a web-browser
- The AU is initiated from the web browser

The launch sequence of an AU is as follows:

1. Student selects an Assignable Unit (AU) to launch from the CMI's user's interface (Menu)
2. The CMI appends startup parameters to the URL location of the AU and directs the web-browser to this "modified" URL (see 6.3.1 The "Launch URL" - below).
3. The AU starts execution and retrieves the Query String from the web-browser, parses the startup parameters and sends a message to the CMI requesting startup and/or previous state data. This message is called a "GetParam" (see section 6.4.4). The "GetParam" message is always the first message issued by the AU.
4. The CMI receives the "GetParam" request and sends startup data.
5. During the rest of the AU session, the AU sends message(s) CMI reporting student performance. These messages are the "PutParam" message (see section 6.4.5) and other "optional" messages (see section 6.4.6). The AU must send at least one "PutParam" message prior to exit.
6. Just prior to exiting, the AU sends a message to the CMI indicating that the AU session has terminated. This message is called a "ExitAU" (see section 6.4.7)

6.3.1 The "Launch URL"

The Launch URL is dynamically created by the CMI in order to launch the AU. The structure of the launch URL is as follows:

*{URL to Assignable Unit}***?***{CMI generated query string}*

The (CMI generated) query string is separated from the Assignable Unit's URL by "?" (Question mark). The URL to the AU is the value of *Course Elements.Filename* corresponding to the AU in the course structure (see section 3.4.7).. The query string is composed of name/value pairs (i.e. name=value) separated by ampersands ("&"s). All values are url-encoded (see section 6.4.1.1) and must be url-decoded prior to interpretation. The value of *Course Elements.Web Launch Parameters* is appended to the CMI generated querystring.

The query string has the following structure:

aicc_sid=*{CMI generated session ID}***&aicc_url=***{URL to receive AU messages}***&***[AU specific launch parameters }*

AU specific launch parameters are obtained by the CMI from the course structure from *Course Elements.Web Launch Parameters* data element (for the AU being launched). See section 3.4.14 for description and format. The other name/value pairs in the query string are described in the table below:

Launch parameter (Name/Value pairs) generated by the CMI

Name	Value Usage/Description	Value Data Type (see Datatypes 9.0)	Obligation
AICC_SID	A string generated by the CMI (prior to AU launch) that uniquely identifies the AU session among all other active AU sessions The Assignable Unit uses this value to identify its session when making requests to the CMI system. This value must be contained in request messages made by the AU. (See section 6.4.2)	CMIIdentifier (URL-Encoded)	Mandatory
AICC_URL	The URL where the AU is to send its HACP request messages.	CMIurl (URL-Encoded)	Mandatory

6.4 Method of Communication

The method of communication utilizes specially formatted HTTP messages. HTTP is a symmetric protocol. For every request, there is a response. The AU sends "request" messages to the CMI and the CMI sends "response" messages back. The AU initiates all communication to the CMI. There are 8 types of request messages that an AU can make to the CMI, they are described in the table below.

HACP Message Type	Description
GetParam	In response to this AU request message the CMI sends a response message that contains AU specific launch parameters, previous state information, and a acknowledgement (see section 6.6.1)
PutParam	This AU request message sends information on student activity, performance, and AU state to the CMI system. The CMI receives this information and send an acknowledgement as a response. (see section 6.6.2).
PutComments	This AU request message sends information that contains written "comments" made by the student to the CMI system. The CMI receives this information and sends an acknowledgement as a response. (see section 6.6.3).
PutObjectives	This AU request message sends information that contains student performance (to specific objectives) to the CMI system. The CMI receives this information and sends an acknowledgement as a response. (see section 6.6.4).
PutPath	This AU request message sends information to the CMI with regards to the path the student user navigated thru the AU to the CMI system. The CMI receives this information and sends an acknowledgement as a response. (see section 6.6.5).
PutInteractions	This AU request message sends information that contains detailed data on each student interaction measured to the CMI system. The CMI receives this information and sends an acknowledgement as a response. (see section 6.6.6).
PutPerformance	This AU request message sends information that contains Learner performance information to the CMI system. The CMI receives this information and sends an acknowledgement as a response. (see section 6.6.7).
ExitAU	This AU request message sends this message to terminate the AU session. The CMI receives this message and sends an acknowledgement as a response. (see section 6.6.8).

HACP Message Sequence Rules

In a (HACP) communication session with the CMI, the AU must meet the following message sequence rules:

Rule #1 - The first HACP message issued must be a GetParam.
Rule #2 - The last HACP message issued must be an ExitAU.
Rule #3 - At least one PutParam message must be issued prior to an Exit AU message.
Rule #4 - No HACP messages can be issued after a successfully issued ExitAU message.

6.4.1 HACP Transport Mechanism

All HACP messages are sent/received using HTTP/S protocol. (See RFC1945 for a detailed description of HTTP protocol.) The HACP message data are contained in the "entity-body" of HTTP request and response messages.

The AU is the "client" (initiates all communication) and the CMI is the "server" (responds to requests). The AU sends messages to the URL location indicated in the "AICC_URL" launch parameter. (See section 6.3). The "AICC_SID" launch parameter (also in Section 6.3) is used by the AU in the body of request messages to identify the AU session to the CMI.

AU request messages are sent to the CMI system via HTTP messages using the POST method (the GET method is not allowed for HACP communication). The content-type of request messages is "application/x-www-form-urlencoded".

The CMI responds to a HTTP/POST message with a HTTP response message. The content-type of response messages is "text/plain".

6.4.1.1 URL-Encoding/Decoding

All name/value pairs in HACP request messages (see section 6.4.2) and URL launch parameters (see section 6.3.1) require that values (and sometimes names) be url-encoded. Url-encoding is used for data transport purposes only. Once url-encoded data is received/read, it must be url-decoded prior to interpreting the data.

The rules for url-encoding are as follows:

Rule #1 - Spaces are converted to the "+" (Plus sign) or "%20"

Rule #2 - All "unsafe" characters, control characters, and "upper ASCII" characters (see table below) must always be encoded in an escape sequence. An escape sequence is a "%" (percent sign) followed by 2 hexadecimal digits. The BNF notation (see sections 10.0 and 10.2) for an escape sequence is as follows:

"%" HEX HEX

For example, "%3F" would represent a url-encoding of "?" (Question mark) character.

Table of ASCII characters that must be encoded

Characters That must be encoded	BNF Notation (see sections 10.0 and 10.2)
Unsafe Characters	"/" | ";" | "?" | "{" | "}" | "|" | "\" | "^" | "~" | "[" | "]" | "`" | "%" | "#" | ">" | "<" | <">
Control Characters	CTL
"Upper ASCII" characters (per ISO-8859)	EXTENDED

Rule #3 - Any other characters may be encoded in an escape sequence (if desired).

The rules for url-decoding are as follows:

Rule #1 - "+" (Plus signs) are converted to spaces

Rule #2 - All characters encoded in escape sequences must be decoded.

Rule #3 - All other characters remain unchanged.

6.4.2 HACP Request Message Format

HACP request message are HTTP request messages with the following properties:

The content-type is “application/x-www-form-urlencoded”. This content-type follows a convention called “name/value pairs”. The name is separated from the value by `=' and name/value pairs are separated from each other by `&'. (e.g. name1=value2&name2=value2). All names and values are url-encoded (see section 6.4.1.1).

The entity-body is composed of the following name/value pairs (depicted in the table below):

Name/Value pairs in a HACP request message

Name	Value Usage/Description	Value Data Type (see Datatypes 9.0)	Obligation
command	Defines request message type.	HacpCommand (URL encoded)	Mandatory
version	Version of the CMI Specification.	CMIVersionNumber (URL encoded)	Mandatory
session_id	This is a string that uniquely identifies the AU session among all other active AU sessions The Assignable Unit uses this value to identify its session when making requests to the CMI system. The value used for session_id is passed to the AU by the CMI via the *AICC_SID* launch parameter. (See section 6.3.1)	CMIIdentifier (URL encoded)	Mandatory
AU_password	AU specific password. This value must match the corresponding value for *Course Elements.AU Password* (See 3.4.15) in the course structure.	CMIString255CSV (URL encoded)	Optional
AICC_Data	Data being sent to the CMI system.	See AICC_Data format for each message in section 6.6. (URL encoded)	Mandatory for all messages except GetParam and ExitAU

Additional usage rules for the name/value pairs in the entity-body are as follows:

- All names and values are url-encoded.
- Values must url-decoded prior to use.
- All names are case-insensitive.
- Each of the name/value pairs can be in any sequence
- If an optional value is to be omitted, the name must also be omitted.

The following is an example of a GetParam request message (See section 6.6 for examples of each message type):

GetParam Request Message- example
`command=getparam&version=4%2E0&session_id=xyz123`

6.4.3 HACP Response Message Format

HACP response message are HTTP response messages with the following properties:

The content-type is “text/plain”.

The data is arranged in format similar to “name/value pairs”. The name is separated from the value by `=' and name/value pairs are separated from each other by carriage return/linefeed end-of-line markers (e.g. name1=value1*{carriage return/linefeed}*name2=value2). The order of the name/value pairs is significant.

The entity-body is composed of the following name/value pairs (depicted in the table below):

Name/Value pairs in a HACP response message

Name	Value Usage/Description	Value Data Type (See Datatypes 9.0)	Obligation
error	Error Number	HacpErrorNumber	Mandatory
error_text	Error description	HacpErrorText	Optional
version	Version of the CMI Specification.	CMIVersionNumber	Optional
aicc_data	Data sent from the CMI system. See section 6.6.1	CMIFormatINI	Mandatory only for GetParam messages

The following (additional) usage rules apply to response message format:

- Leading and trailing white space (Tab, space) is allowed before and after the name, value and "=" (equals sign).
- The value data of aicc_data begins as the first non-white space character after the "=" and continues until the end of the entity-body buffer.
- The value for all other names begins as the first non-white space character after the "=" and continues until the last non-white character before the carrige return/linefeed.
- The order of the name/value pairs is significant.
- The name, in the name/value pair is not case sensitive.
- If an optional value is to be omitted, the name must also be omitted.

See *HACP_RESPONSE* in section 10.4 for BNF notation depicting a HACP response message format

The following is an example of a GetParam response message (See section 6.6 for examples of each message type):

GetParam Response Message- example

```
error=0
error_text=Successful
aicc_data=
[Core]
   Student_ID = XYZ_1234
   Student_Name = Hyde, Jackson Q.
   Lesson_Location = 45
   Credit = CREDIT
   Lesson_Status = INCOMPLETE
   Score =
   Time = 00:04:30

[CORE_LESSON]

my lesson state data – 11111111111111111110000000000000000000000000

11111111111111111110000000000000000000000000 – end my lesson state data

[Core_Vendor]

My Start up parameters
45,67,78,RR
End of My startup parameters

[Evaluation]
Course_ID = {}

[Student_Data]
 Mastery_Score = 100
```

6.4.4 GetParam Request

The GetParam request message is used by the CMI system to pass data to the AU. It is the only request message that the CMI returns actual data (in addition to simple message acknowledgement) for the AU to read.

The AU must issue the GetParam request prior to any other messages in an AU session.

The AU obtains the following information from a CMI response to a GetParam request:

- Launch parameters
- Previous state (i.e. "book marking") information
- Previous status information.

Typically, an AU will issue only one GetParam request during an AU session. However, an AU may issue additional GetParam requests prior to session end. If an AU issues multiple GetParam requests (during an AU session), the following rules apply:

Rule #1 - If a GetParam request is issued after a PutParam request, the GetParam response will include updated values for the following communication data elements (if set by the PutParam request):
Suspend_Data (see section 2.10)
Core.Lesson Location (see section 2.1.4)
All other data elements contained the GetParam response remain static during an AU session.

For a complete list of data elements contained in the GetParam responses message and the format of both request and response - see section 6.6.1.

6.4.5 PutParam Request

The PutParam request is used to report data to the CMI. The AU must issue a PutParam request containing data to be passed back to CMI so that the CMI system can update its student performance data (and perform any necessary display updates or routing activity).

The AU must issue at least one PutParam request prior to end of the AU session.

The CMI receives the following information from the AU via the PutParam Request:

- Status updates
- AU session state (i.e. "Book marking") information to store

Typically, an AU will issue only one PutParam request during an AU session. However, an AU may issue additional PutParam requests prior to session end. If an AU issues multiple PutParam requests (during an AU session), the following rules apply:

Rule #1 - Additional PutParam requests replace the data from prior PutParam requests. Only the data in the final PutParam request is recorded by the CMI and used to evaluate the AU session results.

For a complete list of data elements contained in the PutParam request message and the format of both the request and response messages - see section 6.6.2.

6.4.6 Optional Messages

In addition to GetParam and PutParam messages, there is a group of optional request messages that an AU may send. These request messages are as follows:

- PutComments (see section 6.6.3)
- PutObjectives (see section 6.6.4)

- PutInteractions (see section 6.6.5)
- PutPath (see section 6.6.6)
- PutPerformance (see section 6.6.7)

The following is true for each of the above request messages:

- If the AU supports the data elements defined for any of the above request messages, the AU will send that request message to the CMI system.
- If CMI receives any of the above messages, it will send response message to the AU *even if it does not support any of the data elements contained in the message.*
- If multiple messages are made during the an AU session, all new data is "additive" and stored by the CMI. Data that is duplicated in multiple messages during an AU session is discarded by the CMI.

6.4.7 ExitAU Message

The AU must issue an ExitAU request to notify the CMI system that the AU session is over. The ExitAU is the last message that is issued in an AU session. For the format of both the ExitAU request and response messages - see section 6.6.8.

6.4.8 Error Conditions

Error handling is an AU responsibility. Every response message (provided by the CMI) will contain an error code. There are 4 HACP error conditions currently defined. They are indicated in the table below. All AU corrective action is implementation dependent (possible actions depicted below are provided for information only).

HACP Error Conditions

Error_Code	Error_Text	Description	Possible AU corrective action
0	Successful	Message successfully received by the CMI system	None.
1	Invalid Command	The message type was not valid. (See datatype *HacpErrorCommand* in section 9.0 for legal vocabulary.)	Try again with a valid message type. If error persists display error message to user.
2	Invalid AU password	The AU had a Password associated with it (See *Course Elements. AU Password*) in the course structure and the AU failed to issued a matching value in the request message.	Display message to user that password is incorrect and to contact technical/admin support.
3	Invalid Session ID	The AU did not provide the proper AICC_SID (see section 6.3) for the AU session. The AU either was unable to parse the Launch parameters properly or the CMI provided an invalid AICC_SID.	Send message again. If error persists display message to user that AICC_SID is incorrect and to contact technical/admin support.

Since HACP is based on HTTP/S protocol, HTTP-specific errors may also occur (such as server "time-out", etc.). In this case, the HTTP response message may come directly from the HTTP server (instead of a valid HACP response from the CMI). See RFC1945 for possible HTTP server error codes, their meaning, and the format of the HTTP error response message.

6.5 Conformance Requirements

Conformance to the HACP binding may be looked at from two viewpoints, that of the Assignable Unit (AU) and that of the CMI. There are three levels of obligation described in this binding specification:

- Mandatory
- Optional
- Extension

Obligations for the AU and the CMI are different.

CMI Conformance

Mandatory means that the CMI shall receive all HACP messages, send an acknowledgement of receipt of those messages (or send data elements requested by the AU), and properly store and use mandatory data elements.

Optional means that a conforming CMI must receive all HACP messages types (and send an acknowledgement of successful receipt) but may not store or use all data, or optional communication data elements. A conforming CMI may support many options.

An *extension* is a data element that is not described in this specification. Extensions may be supported by a CMI. However, extension data elements may not perform the identical function as data elements defined in this specification; 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 CMI is non-conforming.

AU Conformance

Mandatory means that the AU shall issue the indicated HACP messages, and properly store and use the mandatory data elements. Furthermore, the indicated HACP messages will be properly formatted and sent to the CMI system.

Optional means that the AU shall issue the mandatory HACP messages, may issue the optional HACP messages, and may use or support the indicated data elements in either. Furthermore, all HACP messages will be properly formatted and sent to the CMI system.

An *extension* is a data element that is not described in this specification. Extensions may be supported/used by an AU. However, extension data elements may not perform the identical function as data elements defined in this specification; 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 AU is non-conforming. Extension data elements must be passed within existing HACP message types defined.

6.5.1 CMI Responsibilities

Launch and Communication

The CMI system shall do the following to launch an assignable unit (AU):

1. Append launch parameters to the URL location of the AU
2. Redirect the web-browser to the modified URL
3. Listen for AU requests
4. Issue response messages for AU requests

The CMI must support all the data elements described for this binding as mandatory (described in section 6.6). The CMI may support the optional data elements. The CMI may also support extensions not defined in this specification 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 AU not using the extensions.

Sequencing
An assignable unit (AU) may only be launched by a CMI. An AU may not itself launch other assignable units. An assignable unit must, at a minimum, be able to do the following:

1. Have the ability to be launched from a web browser (as described in sections 6.3)
2. Parse Launch parameters (as described in sections 6.3)
3. Issue the minimum required HACP message requests in the required sequence (as described in sections 6.3 and 6.4).

Flow control – moving from one AU to another – is assumed to be the responsibility of the CMI and not the AU itself. This is conceptually important because AU reuse cannot really happen if the AU has embedded information that is context specific to the course. In this context, flow control means that the decision of what AU will next be presented to the student is made by the CMI. (This recognizes that some AU's may make decisions—that is, branch – within themselves, but that kind of internal flow is hidden from the CMI.)

The determination of which AU(s) the student is routed to is determined solely by the CMI and is defined in large part by the Course Structure description (Chapter 3). Chapter 3 defines information about the AU that is context specific to the course (e.g., the default sequence of AU's, and prerequisites or completion requirements that might alter the delivery path.)

6.5.2 Assignable Unit (AU) Responsibilities

Launch and Communication
An assignable unit must, at a minimum, be able to do the following:

1. Have the ability to be launched from a web browser (as described in sections 6.3)
2. Parse Launch parameters (as described in sections 6.3)
3. Issue the minimum required HACP message requests in the required sequence (as described in sections 6.3 and 6.4).
4. Support all the following communication data elements (listed in the tables below)

GetParam (Response) – AU mandatory data elements

Group Name or Keyword	Communication Data Model Name	Section

PutParam (Request) – AU mandatory data elements

Group Name or Keyword	Communication Data Model Name	Section
[Core]	Core	2.1
Lesson_Location	Core.Lesson Location	2.1.4
Lesson_Status	Core.Lesson Status	2.1.6
Score	Core.Score	2.1.10
Time	Core.Session_Time	2.1.12

The AU must support all the data elements described for this binding as mandatory (above). The AU may support the optional data elements. The AU may also support extensions not defined in this specification 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 CMI not using the extensions.

Sequencing
An AU may not itself launch other assignable units

6.6 Communication Data Model Mapping

This section contains the mapping of the communication data model elements (defined in section 2.0) to the HACP binding. The contents of the HACP request and response messages are defined in this section. The request messages used by the AU for communication are as follows:

- GetParam
- PutParam
- PutComments
- PutObjectives
- PutPath
- PutInteractions
- PutPerformance

The following is defined for each of the above message types:

- A description of the message's purpose
- A list of communication data model elements used
- The format of the data contained in AICC_DATA name/value pair (if any)
- An example of request and response messages

6.6.1 GetParam (Messages)

Purpose
The GetParam request is used by the CMI system to pass data to the AU. (See section 6.4.1).

Data Model Elements
The following table describes the Group and Keywords used by the GetParam response message with corresponding data model names, references, and Mandatory/Required designations. For specific usage of a data element refer to the corresponding section in the Chapter 2.0 *Communication Data Model*. Note that ***n*** indicates an array index.

Data Model Elements (Response Message)

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
[Core]	Core	2.1	Mandatory
Student_ID	Core.Student Id	2.1.1	Mandatory
Student_Name	Core.Student Name	2.1.2	Mandatory
Lesson_Location	Core.Lesson Location	2.1.4	Mandatory
Credit	Core.Credit	2.1.5	Mandatory
Lesson_Status	Core.Lesson Status	2.1.6	Mandatory
	Core.Entry	2.1.8	Mandatory
Score	Core.Score	2.1.10	Mandatory
	Core.Score.Raw	2.1.10	Mandatory
	Core.Score.Max	2.1.10	Optional
	Core.Score.Min	2.1.10	Optional
Time	Core.Total_Time	2.1.12	Mandatory
Lesson_Mode	Core.Lesson Mode	2.1.13	Optional
[Core_Lesson]	Suspend Data	2.1	Mandatory
[Core Vendor]	Launch Data	2.3	Mandatory
[Comments]	Comments From LMS	2.6	Optional
[Evaluation]	Evaluation	2.7	Optional
Course_ID	Evaluation.Course ID	2.7.2	Optional
[Objectives_Status]	Objectives	2.8	Optional
J_ID.***n***	Objectives.ID	2.8.1	Optional
J_Score.***n***	Objectives.Score	2.8.2	Optional
	Objectives.Score.Raw	2.8.2	Optional
	Objectives.Score.Max	2.8.2	Optional

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
	Objectives.Score.Min	2.8.2	Optional
J_Status.***n***	Objectives.Status	2.8.3	Optional
[Student_Data]	Student Data	2.9	Optional
Attempt_Number	Student Data.Attempt Number	2.9.1	Optional
Mastery_Score	Student Data.Mastery Score	2.9.3	Optional
Max_Time_Allowed	Student Data.Max Time Allowed	2.9.3	Optional
Time_Limit_Action	Student Data.Time Limit Action	2.9.4	Optional
	Student Data.	2.9.7	Optional
Score.***n***	*Student Data.Sessions Journal.Lesson Score*	2.9.7.1	Optional
	.Raw	2.9.7.1	Optional
	.Max	2.9.7.1	Optional
	.Min	2.9.7.1	Optional
Lesson_Status.***n***	*Student Data.Sessions Journal.Lesson Status*	2.9.7.2	Optional
[Student_Demographics]	Student Demographics	2.13	Optional
City	Student Demographics.City	2.13.1	Optional
Class	Student Demographics.Class	2.13.2	Optional
Company	Student Demographics.Company	2.13.3	Optional
Country	Student Demographics.Country	2.13.4	Optional
Experience	Student Demographics.Experience	2.13.5	Optional
Familiar_Name	Student Demographics.Familiar Name	2.13.6	Optional
Instructor_Name	Student Demographics.Instructor Name	2.13.7	Optional
Job_Title	Student Demographics.Title	2.13.12	Optional
Native_Language	Student Demographics.Native Language	2.13.8	Optional
State	Student Demographics.State	2.13.9	Optional
Street_Address	Student Demographics.Street Address	2.13.10	Optional
Telephone	Student Demographics.Telephone	2.13.11	Optional
Years_Experience	Student Demographics.Years Experience	2.13.13	Optional
[Student_Preferenc es]	Student Preference	2.1	Optional
Audio	Student Preference.Audio	2.10.1	Optional
Language	Student Preference.Language	2.10.2	Optional
Lesson_Type	Student Preference.Lesson Type	2.10.3	Optional
Speed	Student Preference.Speed	2.10.4	Optional
Text	Student Preference.Text	2.10.5	Optional
Text_Color	Student Preference.Text Color	2.10.6	Optional
Text_Location	Student Preference.Text Location	2.10.7	Optional
Text_Size	Student Preference.Text Size	2.10.8	Optional
Video	Student Preference.Video	2.10.9	Optional
Window.1	Student Preference.Windows	2.10.10	Optional

AICC_Data Format (Request Message)

Not Applicable for GetParam request messages. If the aicc_data name/value pair is present in GetParam request messages, it is ignored by the CMI.

AICC_Data Format (Response Message)

The GetParam response message is formatted as datatype *CMIFormatINI* (See section 9.0 - Datatypes). All mandatory data elements (listed in the table above) must be included.

Example

An example of a typical set of GetParam request/response messages are shown below:

GetParam Request Message- example

```
command=GetParam&version=4.0&session_id=xyz123
```

GetParam Response Message - example

```
error=0
error_text=Successful
aicc_data=; line 1
; line 2
; line 3
[Core]
```

GetParam Response Message - example

```
     ; Comment
   Student_ID = XYZ_1234
   Student_Name = Hyde, Jackson Q.
   Lesson_Location = 45
   Credit = CREDI
   Lesson_Status = INCOMPLETE
   Score =
   Time = 0000:04:30.34
   Lesson_Mode = Normal

; Core_lesson is free-form group
[CORE_LESSON]

my lesson state data – 11111111111111111110000000000000000000000000

11111111111111111110000000000000000000000000 – end my lesson state data
; Core_vendor is also a free-form group
[Core_Vendor]

My Start up parameters
45,67,78,RR
End of My startup parameters

[Evaluation]
Course_ID = {}

[Student_Data]
Mastery_Score = 100
```

6.6.2 PutParam (Messages)

Purpose
The PutParam request is used by the AU to pass (mandatory and optional) data to the CMI (See section 6.4.2).

Data Model Elements
The following table describes the Group and Keywords used by the PutParam request with corresponding data model names, references, and Mandatory/Required designations. For specific usage of a data element refer to the corresponding section in the Chapter 2.0 *Communication Data Model*. Note that ***n*** indicates an array index.

Data Model Elements (Request Message)

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
[Core]	Core	2.1	Mandatory
Lesson_Location	Core.Lesson Location	2.1.4	Mandatory
Lesson_Status	Core.Lesson Status	2.1.6	Mandatory
	Core.Exit	2.1.7	Mandatory
Score	Core.Score	2.1.10	Mandatory
	Core.Score.Raw	2.1.10	Mandatory
	Core.Score.Max	2.1.10	Mandatory
	Core.Score.Min	2.1.10	Mandatory
Time	Core.Session_Time	2.1.12	Mandatory
[Core_Lesson]	Suspend Data	2.1	Mandatory
[Comments]	Comments From Learner	2.4	Optional
[Objectives_Status]	Objectives	2.8	Optional
J_ID.***n***	Objectives.ID	2.8.1	Optional
J_Score.***n***	Objectives.Score	2.8.2	Optional
	Objectives.Score.Raw	2.8.2	Optional
	Objectives.Score.Max	2.8.2	Optional
	Objectives.Score.Min	2.8.2	Optional
J_Status.***n***	Objectives.Status	2.8.3	Optional
[Student_Data]	Student Data	2.9	Optional
Tries_During_Lesson	Student Data.Tries During Lesson	2.9.5	Optional

Group Names and Keywords	Communication Data Model Name	Section	CMI Obligation
Try_Score.***n***	Student Data.Tries.Try_Score	2.9.2	Optional
Try_Time.***n***	Student Data.Tries.Try_Time	2.9.2	Optional
Try_Status.***n***	Student Data.Tries.Status	2.9.2	Optional
[Student_Preferences]	Student Preference	2.1	Optional
Audio	Student Preference.Audio	2.10.1	Optional
Language	Student Preference.Language	2.10.2	Optional
Lesson_Type	Student Preference.Lesson Type	2.10.3	Optional
Text	Student Preference.Text	2.10.5	Optional
Text_Color	Student Preference.Text Color	2.10.6	Optional
Text_Location	Student Preference.Text Location	2.10.7	Optional
Text_Size	Student Preference.Text Size	2.10.8	Optional
Video	Student Preference.Video	2.10.9	Optional
Window***n***	Student Preference.Windows	2.10.10	Optional

AICC_Data Format (Request Message)
The PutParam request message is formatted as datatype *CMIFormatINI* (See section 9.0 - Datatypes). All mandatory data elements (listed in the table above) must be included. All data is url-encoded and must be decoded prior to interpretation.

AICC_Data Format (Response Message)
Not Applicable for PutParam response messages. If the aicc_data name/value pair is present in PutParam response messages, it is ignored by the AU.

Example
An example of a typical PutParam (request/reponse) message sequence and AICC_DATA are show below:

AICC_DATA example (for a PutParam request) prior to URL-encoding

```
;
; Finish File
;
[Core]
   Lesson_Location = 87
   Lesson_Status = C
   Score =
   Time = 00:02:30

[CORE_LESSON]

my lesson state data – 11111111111111111110000000000000000001110000


11111111111111111110000000000000111000000000 – end my lesson state data

[COMMENTS]

<1><L.Slide#2> This slide has the fuel listed in the wrong units <e.1>
```

PutParam Request Message- example

```
command=PutParam&version=4.0&session_id=345678&AICC_data=%3B%0D%0A%3B%20Finish%20File%0
D%0A%3B%0D%0A%5BCore%5D%0D%0A%20%20%20Lesson_Location%20%3D%2087%0D%0A%20%20%20Lesson_S
tatus%20%3D%20C%0D%0A%20%20%20Score%20%3D%20%0D%0A%20%20%20Time%20%3D%2000%3A02%3A30%0D
%0A%0D%0A%5BCORE_LESSON%5D%0D%0A%0D%0Amy%20lesson%20state%20data%20%2D%2011111111111111
11111100000000000000000001110000%0D%0A%0D%0A%0D%0A11111111111111111111000000000001110000000
00%20%2D20end%20my%20lesson%20state%20data%0D%0A%0D%0A%5BCOMMENT%5D%0D%0A%0D%0A%3C1%3E%
3CL.Slide%232%3E%20This%20slide%20has%20the%20fuel%20listed%20in%20the%20wrong%20units%
20%3Ce.1%3E%0D%0A
```

PutParam Response Message - example
error=0 error_text=Successful

6.6.3 PutComments (Messages)

Purpose
The PutComments request sends data containing freeform feedback from the student (recorded by the AU) to the CMI. It is a duplicate of the [Comments] group that is passed to the CMI system in PutParam request(s).

NOTE: If a CMI system receives data from the AU in both [Comments] group (PutParam request) and the PutComments request in the same AU session, then CMI must retain the data from the PutComments request and discard the [Comments] group data from the PutParam request(s).

Data Model Elements
The following table identifies the Comment File's Fields, Data Model Names, and Data Model Section reference.

Data Model Elements (Request Message)

CSV File Field Identifier	Communication Data Model Name	Section
Course_ID	Itemized Comments From Learner.Course_ID	2.5.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Itemized Comments From Learner.Lesson_ID	2.5.5
Date	Itemized Comments From Learner.Date	2.5.3
Time	Itemized Comments From Learner.Time	2.5.7
Location	Itemized Comments From Learner.Location	2.5.6
Comment	Itemized Comments From Learner.Content	2.5.1

AICC_Data Format (Request Message)
The AICC_DATA value is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules). All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

The value of AICC_DATA is url-encoded and must be decoded prior to interpretation.

AICC_Data Format (Response Message)
Not Applicable for PutComments response messages. If the aicc_data name/value pair is present in PutComments response messages, it is ignored by the AU.

Example
An example of a typical PutComments (request/reponse) message sequence and AICC_DATA are show below:

AICC_DATA example (for a PutComments request) prior to URL-encoding
"course_id","student_id","lesson_id","date","time","location", "comment" "A380FT-1","jqh2003","APU1","2006/01/15",00:14:23 ,frame3, "I think that the word received is not spelled correctly. The reason I'm not sure is because of the colors used for the background and foreground text colors. Purple on orange is really hard to read sometimes." "A380FT-1","jqh2003","APU1","2006/01/15",00:14:36, frame16, "Why did you change colors? I was just getting used to purple on orange."

PutComments Request Message - example

```
Aicc_data=%22course_id%22%2C%22student_id%22%2C%22lesson_id%22%2C%22date%22%2C%22time%22%2C%22
location%22%2C%20%22comment%22%0D%0A%22A380FT%2D1%22%2C%22jqh2003%22%2C%22APU1%22%2C%222006%2F
01%2F15%22%2C00%3A14%3A23%20%2Cframe3%2C%20%22I%20think%20that%20the%20word%20received%20is%20
not%20spelled%20correctly.%20The%20reason%20I%27m%20not%20sure%20is%20because%20of%20the%20col
ors%20used%20for%20the%20background%20and%20foreground%20text%20colors.%20%20%20Purple%20on%20
orange%20is%20really%20hard%20to%20read%20sometimes.%22%0D%0A%22A380FT%2D1%22%2C%22jqh2003%22%
2C%22APU1%22%2C%222006%2F01%2F15%22%2C00%3A14%3A36%2C%20frame16%2C%20%22Why%20did%20you%20chan
ge%20colors%3F%20I%20was%20just%20getting%20used%20to%20purple%20on%20orange.%22&version=4.0&c
ommand=PutComments&session_id=McKim109
```

PutComments Response Message - example
error=0 error_text=Successful

6.6.4 PutInteractions (Messages)

Purpose
All of the items in this file are related to a recognized and recorded input from the student (recorded by the AU). Normally, the interactions recorded are student responses to a question. (See sections listed in table below for description of the data elements recording student interactions)

Data Model Elements
The following table identifies the Interactions File's Fields, Data Model Names, and Data Model Section reference.

Data Model Elements (Request Message)

CSV File Field Idenifier	Communication Data Model Name	Section
Course_ID	Evaluation.Course_ID	2.7.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Lesson_ID	2.14
Date	Interactions.Date	2.11.3
Time	Interactions.Time	2.11.4
Interaction ID	Interactions.ID	2.11.1
Objective ID	Interactions.Objectives	2.11.2
Type Interaction	Interactions.Type	2.11.5
Correct Response	Interactions.Correct Responses	2.11.6
Student Response	Interactions.Student Response	2.11.8
Result	Interactions.Result	2.11.9
Weighting	Interactions.Weighting	2.11.7
Latency	Interactions.Latency	2.11.10

AICC_Data Format (Request Message)
The AICC_DATA value is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules). All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

The value of AICC_DATA is url-encoded and must be decoded prior to interpretation.

AICC_Data Format (Response Message)
Not Applicable for PutInteractions response messages. If the aicc_data name/value pair is present in PutInteractions response messages, it is ignored by the AU.

Example
An example of a typical PutInteractions (request/response) message sequence and AICC_DATA are show below:

AICC_DATA example (for a PutInteractions request) prior to URL-encoding

```
"course_id","student_id","lesson_id","date","time","interaction_id","objective_id",
"type_interaction","correct_response","student_response","result","weighting","latency"
"A340ft-2","jqh085","APU1","2004/01/15","15:14:23",37,ft1016,C,A,C,W,, 00:00:3
"A340ft-2","wam016","APU1","2004/01/15","15:14:23",38,ft2223,t,t,t,,, 00:00:01
"A340ft-2","dag085","APU1","2004/01/15","15:14:23",39,ft1134,C,B,B,C,, 00:00:02
"A340ft-2","trd018","APU1","2004/01/15","15:14:23",40,ft1156,C,C,C,C,, 00:00:04
```

PutInteractions Request Message- example

```
Command=PutInteractions&AICC_data=%22course_id%22%2C%22student_id%22%2C%22lesson_id%22%2C%
22date%22%2C%22time%22%2C%22interaction_id%22%2C%22objective_id%22%2C%22type_interaction%2
2%2C%22correct_response%22%2C%22student_response%22%2C%22result%22%2C%22weighting%22%2C%22
latency%22%0D%0A%22A340ft%2D2%22%2C%22jqh085%22%2C%22APU1%22%2C%222004%2F01%2F15%22%2C%221
5%3A14%3A23%22%2C37%2Cft1016%2CC%2CA%2CC%2CW%2C%2C%2000%3A00%3A3%0D%0A%22A340ft%2D2%22%2C%
22wam016%22%2C%22APU1%22%2C%222004%2F01%2F15%22%2C%2215%3A14%3A23%22%2C38%2Cft2223%2Ct%2Ct
%2Ct%2C%2C%2C%2000%3A00%3A01%0D%0A%22A340ft%2D2%22%2C%22dag085%22%2C%22APU1%22%2C%222004%2
F01%2F15%22%2C%2215%3A14%3A23%22%2C39%2Cft1134%2CC%2CB%2CB%2CC%2C%2C%2000%3A00%3A02%0D%0A%
22A340ft%2D2%22%2C%22trd018%22%2C%22APU1%22%2C%222004%2F01%2F15%22%2C%2215%3A14%3A23%22%2C
40%2Cft1156%2CC%2CC%2CC%2CC%2C%2C%2000%3A00%3A04&Version=4.0&session_id=xavier123
```

PutInteractions Response Message - example

```
error=0
error_text=Successful
```

6.6.5 PutObjectives (Messages)

Purpose

This file contains information on how the student has performed on objectives related to the AU. The performance may be related to previous sessions in the AU, or to the student user's performance in other AU's (in the same course) related to the same objectives. These objectives are only those associated with the current launching AU, not all the objectives in the course or curriculum.

Data Model Elements

The following table identifies the Objective Status File's Fields, Data Model Names, and Data Model Section reference.

Data Model Elements (Request Message)

CSV File Field Idenifier	Communication Data Model Name	Section
Course_ID	Evaluation.Course_ID	2.7.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Lesson_ID	2.14
Date	Objectives.Date	2.8.4
Time	Objectives.Time	2.8.5
Objective ID	Objectives.ID	2.8.1
Score	Objectives.Score	2.8.2
Status	Objectives.Status	2.8.3
Mastery Time	Objectives.Mastery Time	2.8.6

AICC_Data Format (Request Message)

The AICC_DATA value is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules). All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must

have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

The value of AICC_DATA is url-encoded and must be decoded prior to interpretation.

AICC_Data Format (Response Message)
Not Applicable for PutObjectives response messages. If the aicc_data name/value pair is present in PutObjectives response messages, it is ignored by the AU.

Example
An example of a typical PutObjectives (request/reponse) message sequence and AICC_DATA are show below:

AICC_DATA example (for a PutObjectives request) prior to URL-encoding

```
COURSE_ID, STUDENT_ID, LESSON_ID, DATE, TIME, OBJECTIVE_ID, SCORE, STATUS, MASTERY_TIME
"MD80-2","STU1009","APU1","2004/01/15","10:14:23","APU1684",3,, "passed","00:02:37"
```

PutObjectives Request Message- example

SESSION_ID=LEZAT1993&COMMAND=PUTOBJECTIVES&AICC_DATA=COURSE_ ID%2C%20STUDENT_ID%2C%20LESSON_ID%2C%20DATE%2C%20TIME%2C%20OBJECTIVE_ID%2C%20SCORE%2C%20STATUS %2C%20MASTERY_TIME%0D%0A%22MD80%2D2%22%2C%22STU1009%22%2C%22APU1 %22%2C%222004%2F01%2F15%22%2C%2210%3A14%3A23%22%2C%22APU1684%22%2C3%2C%2C%20%22PASSED%22%2C%2200%3A02%3A37%22&VERSION=4.0

PutObjectives Response Message - example

```
error=0
error_text=Successful
```

6.6.6 PutPath (Messages)

Purpose
To provide a mechanism to record the "paths" a student use took during AU session(s). The paths recorded are generally the order in which the student navigates through the AU. (See sections listed in table below for descriptions of the data elements recording path information)
.

Data Model Elements
The following table identifies the PutPath request message Fields, Data Model Names, and Data Model Section reference.

Data Model Elements (Request Message)

CSV File Field Identifier	Communication Data Model Name	Section
Course_ID	Evaluation.Course_ID	2.7.2
Student_ID	Core.Student Id	2.1.1
Lesson_ID	Lesson_ID	2.14
Date	Paths.Date	2.12.2
Time	Paths.Time	2.12.3
Element Location	Paths.Location ID	2.12.1
Status	Paths.Status	2.12.4
Why_Left	Paths.Why Left	2.12.5
Time_in_Element	Paths.Time in Element	2.12.6

AICC_Data Format (Request Message)
The AICC_DATA value is text formatted as datatype *CMIFormatCSV* (See *CMIFormatCSV* in section *9.0 – Datatypes* for a detailed description of formatting rules). All *CSV File Field Identifiers* listed above must be present in the header row, even if a specific field is not supported/used by the CMI. All unsupported data elements are represented as empty strings. Note that field identifiers identify field position (i.e. "columns") in a record (i.e. "row") and can be in any order. Custom fields may be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

The value of AICC_DATA is url-encoded and must be decoded prior to interpretation.

AICC_Data Format (Response Message)
Not Applicable for PutPath response messages. If the aicc_data name/value pair is present in PutPath response messages, it is ignored by the AU.

Example
An example of a typical PutPath (request/reponse) message sequence and AICC_DATA are show below:

AICC_DATA example (for a PutPath request) prior to URL-encoding

```
"course_id","student_id","lesson_id","date","time","element_location","status","why_left","ti
me_in_element"
"course6","stu2310","first1","2003/06/05","14:10:31","page1","P","S","00:00:24"
"course6","stu2310","first1","2003/06/05","14:10:55"," page2","P","S","00:01:06"
"course6","stu2310","first1","2003/06/05","14:12:01"," page3","I","L","00:02:24"
"course6","stu2310","first1","2003/06/05","14:13:25"," page4","P","S","00:00:54"
"course6","stu2310","first1","2003/06/05","14:14:19"," page5","P","L","00:02:40"
"course6","stu2310","first1","2003/06/05","14:16:59"," page6","P","S","00:03:03"
"course6","stu2310","first1","2003/06/05","14:20:02"," page7","P","E","00:02:12"
```

PutPath Request Message

```
aicc_data=%22page2%22%2C%22P%22%2C%22S%22%2C%2200%3A01%3A06%22%0D%0A%22course6%22%2C%22stu231
0%22%2C%22first1%22%2C%222003%2F06%2F05%22%2C%2214%3A12%3A01%22%2C%22%20page3%22%2C%22I%22%2C
%22L%22%2C%2200%3A02%3A24%22%0D%0A%22course6%22%2C%22stu2310%22%2C%22first1%22%2C%222003%2F06
%2F05%22%2C%2214%3A13%3A25%22%2C%22%20page4%22%2C%22P%22%2C%22S%22%2C%2200%3A00%3A54%22%0D%0A
%22course6%22%2C%22stu2310%22%2C%22first1%22%2C%222003%2F06%2F05%22%2C%2214%3A14%3A19%22%2C%2
2%20page5%22%2C%22P%22%2C%22L%22%2C%2200%3A02%3A40%22%0D%0A%22course6%22%2C%22stu2310%22%2C%2
2first1%22%2C%222003%2F06%2F05%22%2C%2214%3A16%3A59%22%2C%22%20page6%22%2C%22P%22%2C%22S%22%2
C%2200%3A03%3A03%22%0D%0A%22course6%22%2C%22stu2310%22%2C%22first1%22%2C%222003%2F06%2F05%22%
2C%2214%3A20%3A02%22%2C%22%20page7%22%2C%22P%22%2C%22E%22%2C%2200%3A02%3A12%22&
command=PutPath&version=4.0&session_id=345699&
```

PutPath Response Message - example

```
error=0
error_text=Successful
```

6.6.7 PutPerformance (Messages)

Purpose
To record simulation-specific data from AU session(s) for later analysis.

Data Model Elements
Not applicable. PutPerformance data is developer-defined.

AICC_Data Format (Request Message)
The formatting of the data is developer-defined. All data is url-encoded and must be decoded prior to interpretation.

AICC_Data Format (Response Message)
Not Applicable for PutPerformance response messages. If the aicc_data name/value pair is present in PutPerformance response messages, it is ignored by the AU.

Example
Not applicable.

6.6.8 ExitAU (Messages)

Purpose
To notify the CMI of AU session termination

Data Model Elements
Not Applicable.

AICC_Data Format (Request Message)
Not Applicable for ExitAU request messages. If the aicc_data name/value pair is present in ExitAU request messages, it is ignored by the CMI.

AICC_Data Format (Response Message)
Not Applicable for ExitAU response messages. If the aicc_data name/value pair is present in ExitAU response messages, it is ignored by the AU.

Example

ExitAU Request Message- example
`command=ExitAU&version=4.0&session_id=xyz123`

ExitAU Response Message - example
`Error=0` `Error_text=Successful`

7.0 Communicating via API (The API Binding)

This chapter defines Application Programming Interface (API) binding to the communication data model in Chapter 2.0. It defines the following:

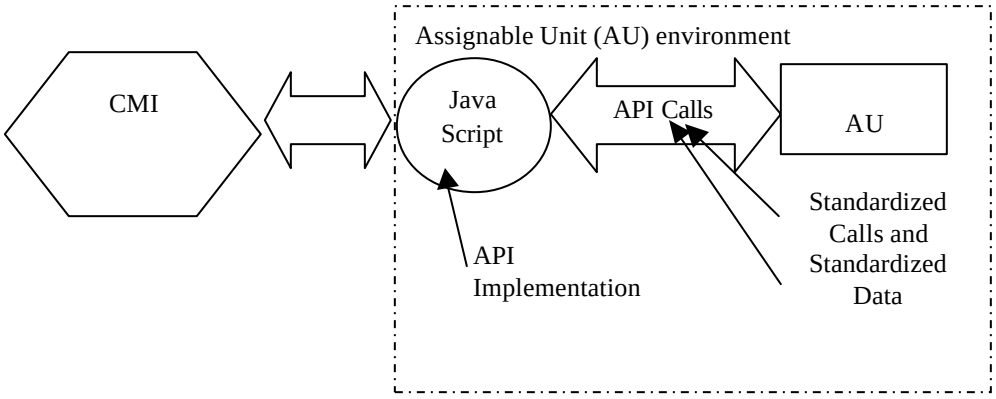
- The environment in which the API operates
- How the CMI launches Assignable Units (AUs)
- How the API is used by AUs to communicate with the CMI system.
- Conformance requirements for this binding.
- Which elements from the data model described in Chapter 2.0 may be used by the API.

Although some of the data elements in the communication data model may have different names in the API binding, there are no new data elements appearing in this chapter.

7.1 Conceptual Model

In the API binding, the AU will communicate using the widely supported JavaScript calling conventions. JavaScript was selected as the method for implementing this API since nearly all browser platforms natively support it. This binding 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 JavaScript object with the CMI is outside the scope of this specification. Implementations of the communications of the JavaScript object with the CMI may vary from product to product."



The Assignable Unit (AU) initiates all communication (after it is launched by the CMI). This communication model makes no provision for communication initiated by the CMI to the AU.

7.2 Operating Environment

The operating environment for this binding is a Web-Browser with JavaScript support.

7.3 Launching an Assignable Unit

Environment
As depicted in the conceptual model, a CMI implements an API *in the assignable unit's environment*. The AU's implementer incorporates in the AU the ability to discover and communicate with an API implementation. A CMI or the front-end to an AU (assignable unit) repository (local or remote) provides an interface for the learner. The CMI either delivers an assignable unit to the learner and starts it, or launches a URI to initiate the AU. An assignable unit has integrated procedures to locate an API implementation.

Sequence of operations
The CMI initiates the launch of an assignable unit. As the AU starts up, it searches for the API implementation. After verifying that the API implementation is accessible in the AU's environment, the AU invokes the API implementation through the instance that has been located.

The AU might not communicate further with the API implementation for some time. All subsequent communication is part of this communication session until it is ended. The AU may request data through the API implementation. Through the API implementation, the CMI returns the requested data or a message identifying an error condition.

While running, the AU may send or set data model data elements for storage across communication sessions. The CMI may use data elements or other data in reports on a learner's status with that AU. The AU may elicit a more detailed error message. The AU may continue communicating in this fashion, requesting and sending data until a learner finishes a AU, a learner terminates the communication session before finishing, or the communication session is abnormally terminated (e.g., loss of power, system crash). In the first two cases, the AU tells the API implementation that it is closing the communication session. In the last case, the CMI will not receive a signal through the API implementation that the communication session is closed. CMI behavior is this case is currently undefined in this Guideline.

A summary of the normal sequence of operations is as follows:

1. The CMI instantiates the API implementation in the assignable unit DOM and initiates launch of an assignable unit.
2. The AU locates the API instance. (Note—This is a required action of the AU.)
3. The AU invokes the LMSInitialize communication session method of the API implementation prior to calling any other method. (Note—The use of this session method is a required action of the AU.)
4. If the AU invokes one or more data-retrieval requests through the API instance, the API returns the data or, in the case of an error, an empty string (""). The API sets an appropriate error status, either "0" for no error or an error code. The error status can be retrieved by the AU on request. Calls to retrieve data (data-transfer methods) are optional actions of the AU.
5. If the AU invokes one or more data-storage requests (LMSSetValue) through the API instance, the API either caches the data to send to the CMI later, or attempts to send the data to the CMI immediately. In either case the API instance returns an acknowledgement, either "true" or, in the case of an error, "false". The API sets an appropriate error status, either "0" for no error or an error code. The error status can be retrieved by the AU on request. Calls to store data (data-transfer methods) are optional actions of the AU.
6. If the AU invokes one or more of the predefined error handling methods through the API instance, the CMI responds appropriately with data or messages through the API instance. Error handling methods are optional actions of the AU. The API instance returns a value or message if a call is made.
7. The AU invokes the termination method of the API instance. (Note—The use of this session method is a required action of the AU.)
8. The API instance rejects any attempt by this instance of the AU to reinitialize the communication session.

7.4 Method of Communication

Communication between AU and CMI is accomplished by the AU invoking function calls (or methods) from the API object. The JavaScript API includes three kinds of methods:

- Session methods – used to mark the beginning and the end of communication between a the AU object and an API implementation.
- Data-transfer methods – used to transfer data model values between a the AU object and an API implementation.
- Error handling methods – used for auxiliary communications (e.g. error handling) between a the AU object and the API implementation.

The set of API function calls or methods consists of the following:

Session Methods
LMSInitialize("")
LMSFinish("")
Data-transfer Methods
LMSGetValue(parameter)
LMSSetValue(parameter, value)
LMSCommit(parameter)
Error handling Methods
LMSGetLastError("")
LMSGetErrorString(parameter)
LMSGetDiagnostic(parameter)

7.4.1 Parameters

The parameters in the API data-transfer methods 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 specification, the data model is always "CMI".

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. 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.

7.4.2 API General Rules

The following list summarizes the usage rules for the API.

- The function or method names are all case sensitive, and must always be expressed exactly as shown above.
- When a function's parameter is a data model element name, it is case sensitive. All data model element names are lower case.
- The first symbol in the data element name identifies the data model. For example, "cmi" indicates the AICC/CMI data model (described in this document). This expands the functionality of these API's by allowing the same API to be used with other data models. (However, the use of other data models is outside the scope of this document).
- There are three reserved keywords. These are all lower case and proceeded by an underscore.
  - _version
  - _children
  - _count
- When LMSGetValue is executed, it returns the last set value if there was one.

7.4.3 Arrays – Handling Lists

There are several data elements that appear in a list or an array. An example of this would be interactions. There may be more than one interaction covered in an AU, and a student may be allowed to perform an interaction 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.

All new array elements shall be added sequentially. The assignable unit shall not skip array numbers or leave empty array elements when constructing a list of array values.

The _count keyword can be used to determine the current number of records in the list. For instance, to determine the number of interactions records currently recorded, the following API would be used:

LMSGetValue("cmi.interactions._count")

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 interaction in a AU would be referred to as "cmi.interactions.0.result ". The result element in the fourth interaction would be referred to as "cmi.interactions.3.result". If a student experienced the first interaction twice, there could be two results associated with the first interaction. These would be identified as "cmi.interactions.0.result" and "cmi.interactions.0.result".

7.4.4 Session Methods

Session methods are used to initiate and terminate data communication between an API implementation and a single instance of an AU object (assignable unit) during a single communication session.

The API implementation may have one of three communication states. Each of these communications states are mutually exclusive and are as follows:

1. Not initialized
2. Running
3. Terminated:

The initial API communication state (before the AU object is launched) shall be "not initialized".

Session Methods	
LMSInitialize	**Description**: This function is used to initiate communication between an assignable unit and an API implementation. It indicates to the API adapter that the assignable unit is going to

Session Methods	

	communicate with the CMI. It allows the CMI to handle CMI specific initialization issues. It is called by the assignable unit before it can call any other API function.
	
	Behavior notes
	When the communication state is “not initialized” and initialization of communication succeeds, the API instance
	sets the communication state to "running";
	sets the error state to "0" (No error); and
	returns "true" to the calling content object.
	1. When the communication state is "not initialized" and initialization of communication fails, the API instance
	a) makes no change to the communication state;
	b) sets the error state to "101" (General exception); and
	c) returns "false" to the calling content object.
	2. When the communication state is "running", the API instance
	a) makes no change to the communication state;
	b) sets the error state to "101" (General exception); and
	c) returns "false" to the calling content object.
	3. When the communication state is "terminated", the API instance
	a) makes no change to the communication state;
	b) sets the error state to "301" (Not initialized); and
	c) returns "false" to the calling content object.
	
	Note: Additional and more specific error codes will be added in later versions of this standard.
	
	Syntax:
	return_value = LMSInitialize(parameter)
	
	Parameter: “”. An empty string must be passed for conformance to this specification. This parameter is reserved for future extensions.
	
	Return Value: String representing a Boolean “true” or “false”. A "true" result indicates that the initialization was successful and a "false" result indicates that it was not.
	
	Example:
	
	`var result = LMSInitialize(“”)`
	`if (result == “false”)`
	`(`
	`// Do some error handling`
	`)`
	`else`
	`(`
	`// Continue with the execution of the assignable unit`
	`)`

LMSFinish	**Description**: The assignable unit must call this when it has determined that it no longer needs to communicate with the CMI. If it successfully called LMSInitialize at any previous point. This call signifies two things: 1. The assignable unit can be assured that any data set using LMSSetValue() calls has been persisted by the CMI. 2. The assignable unit has finished communicating with the CMI. **Behavior notes** 1. When the communication state is "running" and terminating communication succeeds, the API instance a) “Commits” any data in cache b) sets the communication state to "terminated"; c) sets the error state to "0" (No error), and d) returns "true" to the calling content object. 2. When the communication state is "running" and termination of communication or committing the cache fails, the API instance a) makes no change to the communication state; b) sets the error state to "101" (General exception); and c) returns "false" to the calling content object. 3. When the communication state is "not initialized", the API instance a) makes no change to the communication state;

Session Methods	
	b) sets the error state to "301" (Not Initialized); and c) returns "false" to the calling content object. 4. When the communication state is "terminated", the API instance a) makes no change to the communication state; b) sets the error state to "101" (General exception); and c) returns "false" to the calling content object. Note: Additional and more specific error codes will be added in later versions of this standard. **Syntax:** return_value = LMSFinish(parameter) **Parameter**: "". An empty string must be passed for conformance to this specification. This parameter is reserved for future extensions. **Return Value**: String representing a Boolean "true" or "false". A "true" result indicates that the initialization was successful and a "false" result indicates that it was not. **Example** `var result = LMSFinish("")`

7.4.5 Data-Transfer Methods

Data-transfer methods are used to direct the storage and retrieval of data that is to be available within the current communication session.

Data-Transfer Methods	
LMSGetValue	**Description**: This function allows the AU (the assignable unit) to obtain information from the CMI. 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. **Syntax**: return_value = LMSGetValue(parameter) **Parameter**: datamodel.group.element Returns the value of the named element. datamodel._version The _version keyword is used to determine the version of the data model supported by the CMI. datamodel.element._count The _count keyword is used to determine the number of elements currently in an array. The count is the total number of elements in the array, not the index number of the last position in the array. datamodel.element._children The _children keyword is used to determine all the elements in a group or category that are supported by the CMI. **Return Value**: All return values are strings which can be converted to the appropriate type. LMSGetValue(datamodel.group.element) The return value is a string representing the current value of the requested element or group. LMSGetValue(datamodel._version) The return value is a string representing the version of the data model supported by the CMI. LMSGetValue(datamodel.group._children) The return value is a comma-separated list of all of the element names in the specified group or category that are supported by the CMI. If an element has no children, but is supported, an empty string ("") is returned. An empty string ("") is also returned if an element is not supported. A subsequent request for last error can determine if the

Data-Transfer Methods	

	element is not supported. The error "401 Not implemented error" indicates the element is not supported.
	LMSGetValue(datamodel.group._count)
	The return value is an integer that indicates the number of elements in an element list or array.
	
	Examples:
	`LMSGetValue("cmi.core.student_name")`
	A typical return value might be "Hyde, Jackson".
	
	`LMSGetValue("cmi.core.lesson_status")`
	A typical return value might be "incomplete".
	
	`LMSGetValue(cmi._version)`
	The current AICC CMI Guideline is version 4.0 of document CMI001. Therefore a return value of AICC CMI001 4.0 would be appropriate.
	
	`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.

LMSSetValue	**Description:** This function allows the assignable unit to send information to the API. The API may be designed to immediately forward the information to the CMI, 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 CMI when the LMSFinish call is executed by the AU.
	
	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:
	return_value = 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 specification for the element identified in the first parameter.
	
	Return Value: String representing a Boolean.
	A "true" result indicates that the function was successful and a "false" result indicates that it was not.
	
	Examples:
	`var result = LMSSetValue("cmi.core.score.raw", "95")`
	`Sets the cmi.core.score.raw to a value of 95.`

LMSCommit	**Description:** If the JavaScript object (or API implementation) is caching LMSSetValue values, this call requires that any values not yet sent to the CMI be sent.

Data-Transfer Methods	
	In some cases, the API implementation may send the set values to the CMI as soon as they are received, and not cache them locally. In such cases, this API is redundant and would result in no additional action from the API implementation. **Syntax**: result = LMSCommit(parameter) **Parameter**: "". An empty string must be passed for conformance to this specification. This parameter is reserved for future extensions. **Return Value**: String representing a Boolean. A "true" result indicates that the function was successful and a "false" result indicates that it was not. If an API implementation automatically sends a values to the CMI as soon as received, it shall return a "true" to this call. **Example**: `var result = LMSCommit("");` Requires that any cached values, previously set via assignable unit calls to LMSSetValue(), that have not been persisted by the CMI be persisted.

7.4.6 Error Handling Methods

Error handling methods are used for error handling and diagnostics.

All calls to the JavaScript instance result in the error status being set by the instance. This status may be determined using the Error Condition Methods. The rules for setting the error status are the following:

1. All successful calls result in a status of 0 being set.
2. All successful calls result in the error status being set as described in the LMSGetLastError return value.
3. All error condition method calls do not change the error status.

Error Handling Methods	
LMSGetLastError	**Description**: The assignable unit 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 support functions), the error code is reset in the API. The AU 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**: return_value = LMSGetLastError(parameter) **Parameter**: "". An empty string must be passed for conformance to this specification. This parameter is reserved for future extensions. **Return Value**: The return values are integer numbers that identify errors falling into the following categories: 100 General errors 200 Syntax errors 300 CMI errors 400 Data model errors The following codes are available for error messages: 0. No error 101. General exception 102. Server is busy. 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 402. Invalid Set Value, element is a CMI keyword 403. Element is read only 404. Element is write only

Error Handling Methods	
	405. Incorrect data type Additional codes may be added in future versions **Examples**: `var errorCode = LMSGetLastError("")`
LMSGetErrorString	**Description**: This function enables the AU to obtain a textual description of the error represented by the error code number. **Syntax**: return_value = LMSGetErrorString(parameter) **Parameter**: An integer number representing an error code. **Return Value**: A string that represents the verbal description of an error. **Examples**: `var errorString = LMSGetErrorString("403")` errorString should contain "Element is read only".
LMSGetDiagnostic	**Description**: This function enables vendor-specific error descriptions to be developed and accessed by the AU. These would normally provide additional helpful detail regarding the error. **Syntax**: return_value = 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. • "". An empty string. 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. **Examples**: `var moreInfo = LMSGetDiagnostic("403")` moreInfo could contain more vendor specific information on the "Element is read only" error.

7.5 Conformance Requirements

Conformance to this binding may be looked at from two viewpoints, that of the Assignable Unit (AU) and that of the CMI.

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

- Mandatory
- Optional
- Extension

Obligations for the AU and the CMI are different.

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

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

An *extension* is an API or data element that is not described in this specification. Extensions may be supported by a CMI. However, extension API's may not perform the identical 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 CMI is non-conforming.

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

Optional means that the AU 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 specification. The AU may support extensions. However, extension API's may not perform the identical 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 AU is non-conforming.

7.5.1 CMI Responsibilities

The mechanism described here assumes a clean separation between the API function calls used in the AU and the API implementation (or API object or JavaScript object or API instance). The API function calls are embedded in the AU. The API implementation is provided by the CMI when the AU is launched.

Launch
For browser and Web-based AU's, the CMI shall launch the AU from a browser window that contains the API implementation, or must provide a parent frame that contains the API implementation. This window shall contain a reference to the assignable unit (which is an URL).

Communication
The API implementation provided by the CMI 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 API implementation using the LMSGetValue function and modified using a LMSSetValue function. Regardless of implementation details, if a data element is supported by the CMI, 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 CMI shall support the ability of the AU to "get" and "set" the "communication" data elements defined as mandatory in this specification. "Support" means that when the AU executes an " LMSGetValue " on an element, a legal value of the proper format and type and range will be returned. When the AU 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 CMI may support the ability of the AU to "get" and "set" the optional data elements.

The CMI may also support extensions not defined in this specification 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 the AU not using the extensions.

CMI Conformance Requirements
- Supports the following transactions • LMSInitialize • LMSFinish • LMSGetValue • LMSSetValue • LMSCommit • LMSGetLastError • LMSGetErrorString • May support security transactions • LMSGetDiagnostic - Supports all mandatory elements • LMSGetValue shall succeed • LMSSetValue shall succeed - May support any or all optional elements • LMSGetValue may succeed • LMSSetValue may 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

Sequencing

Flow control – moving from one the AU object to another – is assumed to be the responsibility of the CMI and not within the assignable unit (AU) itself. This is conceptually important because AU reuse cannot really happen if the AU has embedded information that is context specific to the course. In this context, flow control means that the decision of what AU (the AU) will next be presented to the student is made by the CMI. (This recognizes that some AU's may make decisions—that is, branch – within itself, but that kind of internal flow is hidden from the CMI.

The determination of which AU(s) the student is routed to is determined solely by the CMI and is defined in large part by the Course Structure description (Chapter 3). Chapter 3 defines information about the AU that is context specific to the course (e.g., the default sequence of AU's, and prerequisites or completion requirements that might alter the delivery path.)

7.5.2 AU Responsibilities

The AU is responsible for discovering (locating) the API object.

The AU shall be able to call JavaScript functions in a "foreign window". The A U does not have to be developed in JavaScript but shall be able to call it. This capability enables the clean separation between the function calls used in the AU and the implementation of those function calls provided by a learning management system.

For conforming Assignable Units, the AU 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.

The AU may support the required set of "communication" data elements defined in this specification.

The table below summarizes the requirements for conforming AU's.

Conformance Requirements for The AU
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

Binding Mechanism

AU shall communicate with a CMI system through a JavaScript API. This API will be part of a JavaScript object attached to either a parent window or the "opener" window for the HTML page. The AU object shall look for an instance of the API implementation in the following locations, in order of precedence, and stop as soon as an instance is found:

a) The chain of parents of the current window, if any exist, until the top window of the parent chain is reached.
b) The opener window, if any.
c) The chain of parents of the opener window, if any exist, until the top window of the parent chain is reached.

An AU object may follow a simple algorithm to find an instance of an API implementation.

- Follow the algorithm until an instance is found.
- When found, return the instance and exit the "find adapter" routine.
- If not found, return a null and exit the routine.

A sample JavaScript implementation of this algorithm tested with several Web browsers is provided below.

Sample JavaScript to Locate API object

Sample JavaScript to Locate API object

```
var findAPITries = 0;

// returns the CMI API object (may be null if not found)
function findAPI(win)
{
   while ( (win.API == null) &&
           (win.parent != null) &&
           (win.parent != win) )
   {
      findAPITries++;
      if (findAPITries > 7)
      {
         alert("Error finding API.");
         return null;
      }
      win = win.parent;
   }
   return win.API;
}

// obtain the CMI API
function getAPI()
{
   var theAPI = findAPI(window);
   if ( (theAPI == null) &&
        (window.opener != null) &&
        (typeof(window.opener) != "undefined") )
   {
      theAPI = findAPI(window.opener);
   }
   if (theAPI == null)
   {
      alert("Unable to find an API adapter");
   }
   return theAPI;
}
```

Summary Points: the AU assignable unit may only be launched by a CMI. An assignable unit may not itself launch other assignable units. An assignable unit must, at a minimum, contain an *initialize()* and a *finish()* API call to conform with this guideline.

7.6 Communication Data Model Mapping

The following table indicates the data elements that may be used by the AU in communicating with a CMI using the API. Definitions and examples for the data elements are in Chapter 2.

In the following table, "***n***" represents the array index (zero based). It is optional when there is only one member in the array. The "Data Model Name" reflects the name of the data element that appears in Chapter 2. The "API Name" is the name that shall be used in the LMSSetValue and LMSGetValue methods to identify the element. Some elements, namely the _count and _children, do not appear in the data model, and may only be used in the API. The "Get/Set" column indicates which methods may be used with the data element. The "Section" column references the section in this document where the data model element is defined. The "Ob" column indicates the whether an element is Mandatory for a CMI or not ("M" indicates mandatory, "O" indicates optional).

API Name	Data Model Name	Section	Ob	Get/Set
	core	2.1	M	none
cmi.core._children			M	Get
cmi.core.student_id	. .student id	2.1.1	M	Get
cmi.core.student_name	. student name	2.1.2	M	Get
cmi.core.lesson_location	. lesson location	2.1.4	M	Get & Set
cmi.core.credit	. credit	2.1.5	M	Get
cmi.core.lesson_status	. lesson status	2.1.6	M	Get & Set
cmi.core.exit	. exit	2.1.7	M	Set
cmi.core.entry	. entry	2.1.8	M	Get
	. score	2.1.10	M	none
cmi.core.score._children			M	Get
cmi.core.score.raw	. . raw	2.1.10	M	Get & Set
cmi.core.score.max	. . max	2.1.10	M	Get & Set
cmi.core.score.min	. . min	2.1.10	M	Get & Set
cmi.core.session_time	. session time		M	Set
cmi.core.total_time	. total time	2.1.12	M	Get
cmi.core.lesson_mode	. lesson mode	2.1.13	O	Get
cmi. suspend_data	suspend data	2.1	M	Get & Set
cmi.launch_data	launch data	2.3	M	Get
cmi.comments	Comments from learner	2.4	O	Get & Set
	Itemized Comments from Learner		O	
cmi.evaluation.comments._children			O	Get
cmi.evaluation.comments._count			O	Get
cmi.evaluation.comments.***n***.date	. Date	2.5.3	O	Set
cmi.evaluation.comments.***n***.time	. Time	2.5.7	O	Set
cmi.evaluation.comments.***n***.location	. Location	2.5.6	O	Set
cmi.evaluation.comments.***n***.content	. Content	2.5.1	O	Set
cmi.evaluation.comments.***n***.lesson_id	Lesson_ID	2.14	O	Set
cmi.comments_from_lms	**Comments from lms**	2.6	O	Get
	objectives	2.8	O	
cmi.objectives._children			O	Get
cmi.objectives._count			O	Get
cmi.objectives.n.id	. id	2.8.1	O	Get & Set
cmi.objectives.n.score	. score	2.8.2	O	
cmi.objectives.score._children			O	Get
cmi.objectives.score._count			O	Get
cmi.objectives.n.score.raw	. . raw	2.8.2	O	Get & Set
cmi.objectives.n.score.max	. . max	2.8.2	O	Get & Set
cmi.objectives.n.score.min	. . min	2.8.2	O	Get & Set
cmi.objectives.n.status	. status	2.8.3	O	Get & Set
cmi.objectives.n.date	. date	2.8.4	O	Set
cmi.objectives.n.time	. time	2.8.5	O	Set
cmi.objectives.n.mastery_time	. mastery time	2.8.6	O	Set
	Student data	2.9	O	
cmi.student_data._children			O	Get
cmi.student_data.attempt_number	. Attempt number	2.9.1	O	Get
cmi.student_data.tries	. Tries	2.9.2	O	Get & Set

API Name	Data Model Name	Section	Ob	Get/Set
cmi.student_data.tries._children			O	Get
cmi.student_data.tries._count			O	Get
cmi.student_data.tries.n.status	. . Status	2.9.2.2	O	Get & Set
cmi.student_data.tries.n.score	. . Score	2.9.2.1	O	Get & Set
cmi.student_data.tries.score._children			O	Get
cmi.student_data.tries.***n***.score.raw	. . . raw	2.9.2.1	O	Get & Set
cmi.student_data.tries.***n***.score.max	. . . max	2.9.2.1	O	Get & Set
cmi.student_data.tries.***n***.score.min	. . . min	2.9.2.1	O	Get & Set
cmi.student_data.tries.***n***.time	. . time	2.9.2.3	O	Set
cmi.student_data.mastery_score	. Mastery score	2.9.2	O	Set
cmi.student_data.max_time_allowed	. Max Time Allowed	2.9.3	O	Get
cmi.student_data.time_limit_action	. Time Limit Action	2.9.4	O	Get
cmi.student_data.tries_during_lesson	. Tries During Lesson	2.9.5	O	Set
cmi.student_data.attempt_records._children	. Sessions Journal	2.9.7	O	Get
cmi.student_data.attempt_records.***n***.score	. . Score	2.9.7.1	O	Get
cmi.student_data.attempt_records.***n***.score.children			O	Get
cmi.student_data.attempt_records.***n***.score.raw	. . . raw	2.9.7.1	O	Get
cmi.student_data.attempt_records.***n***.score.max	. . . max	2.9.7.1	O	Get
cmi.student_data.attempt_records.***n***.score.min	. . . min	2.9.7.1	O	Get
cmi.student_data.attempt_records.***n***.lesson_status	. . Lesson Status	2.9.7.2	O	Get
	Student preference	2.1	O	
cmi.student_preference._children			O	Get
cmi.student_preference.audio	. Audio	2.10.1	O	Get & Set
cmi.student_preference.language	. Language	2.10.2	O	Get & Set
cmi.student_preference.lesson_type	. Lesson type	2.10.3	O	Get & Set
cmi.student_preference.speed	. Speed	2.10.4	O	Get & Set
cmi.student_preference.text	. Text	2.10.5	O	Get & Set
cmi.student_preference.text_color	. Text color	2.10.6	O	Get & Set
cmi.student_preference.text_location	. Text location	2.10.7	O	Get & Set
cmi.student_preference.text_size	. Text size	2.10.8	O	Get & Set
cmi.student_preference.video	. Video	2.10.9	O	Get & Set
cmi.student_preference.windows._count			O	Get
cmi.student_preference.windows.***n***	. Windows	2.10.10	O	Get & Set
	Interactions	2.11	O	
cmi.interactions._children			O	Get
cmi.interactions._count			O	Get
cmi.interactions.***n***.id	. ID	2.11.1	O	Set
	. Objectives	2.11.2	O	
cmi.interactions.objectives._count			O	Get
cmi.interactions.***n***.objectives.***n***.id	. . ID	2.8.1	O	Set
cmi.interactions.***n***.date	. Date	2.11.3	O	Set
cmi.interactions.***n***.time	. Time	2.11.4	O	Set
cmi.interactions.***n***.type	. Type	2.11.5	O	Set
	. Correct Responses	2.11.6	O	
cmi.interactions.***n***.correct_responses._count			O	Get
cmi.interactions.***n***.correct_responses.***n***.pattern		2.11.6	O	Set
cmi.interactions.***n***.weighting	. Weighting	2.11.7	O	Set
cmi.interactions.***n***.student_response	. Student Response	2.11.8	O	Set
cmi.interactions.***n***.result	. Result	2.11.9	O	Set
cmi.interactions.***n***.latency	. Latency	2.11.10	O	Set
	paths	2.12	O	
cmi.paths._children			O	Get
cmi.paths._count			O	Get
cmi.paths.***n***.location_id	. Location ID	2.12.1	O	Set
cmi.paths.***n***.date	. Date	2.12.2	O	Set
cmi.paths.***n***.time	. Time	2.12.3	O	Set
cmi.paths.***n***.status	. Status	2.12.4	O	Set
cmi.paths.***n***.why_left	. Why Left	2.12.5	O	Set
cmi.paths.***n***.time_in_element	. Time in Element	2.12.6	O	Set
	Student demographics	2.13	O	
cmi.student_demographics._children			O	Get
cmi.student_demographics.city	. City	2.13.1	O	Get
cmi.student_demographics.class	. Class	2.13.2	O	Get
cmi.student_demographics.company	. Company	2.13.3	O	Get
cmi.student_demographics.country	. Country	2.13.4	O	Get

API Name	Data Model Name	Section	Ob	Get/Set
cmi.student_demographics.experience	. Experience	2.13.5	O	Get
cmi.student_demographics.familiar_name	. Familiar Name	2.13.6	O	Get
cmi.student_demographics.instructor_name	. Instructor Name	2.13.7	O	Get
cmi.student_demographics.title	. Title	2.13.12	O	Get
cmi.student_demographics.native_language	. Native Language	2.13.8	O	Get
cmi.student_demographics.state	. State	2.13.9	O	Get
cmi.student_demographics.street_address	. Street Address	2.13.10	O	Get
cmi.student_demographics.telephone	. Telephone	2.13.11	O	Get
cmi.student_demographics.years_experience	. Years Experience	2.13.13	O	Get

8.0 Course Structure Definition (File Binding)

This chapter defines the File binding to the course structure data model (in chapter 3.0). This is the only binding to course structure data model.

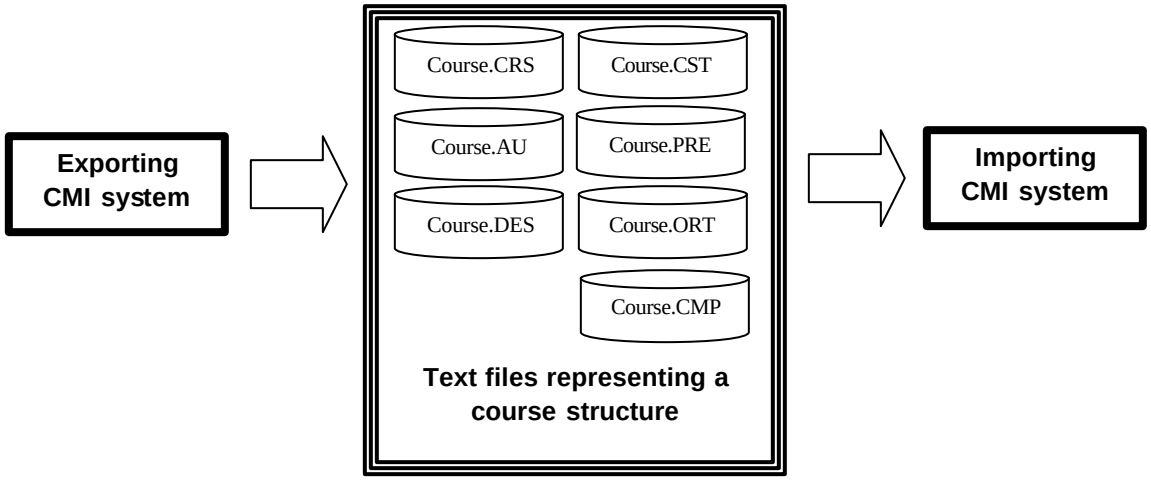
The following items are covered in this section:

- How the CMI uses the files in this binding for interchange (import/export)
- Conformance requirements for this binding
- Which elements from the data model described in chapter 3.0 may be used by the File binding (Including which files specific elements are located in and the format of those files).

Although many of the data elements in the course structure data model have different names in the interchange files, there are no new data elements appearing in this chapter.

8.1 Conceptual Model

In the File binding, the CMI imports and exports course structures using text files (see figure below). To export a course structure, the CMI system writes a series of related text files. The text files represent a general course description (a single file) and series of relational data "tables" (one per file) that define all data needed to recreate a course structure in an importing (target) CMI system. A CMI importing a course structure reads the text files and recreates the course structure (or a subset of the original course structure) for its internal use.



8.2 Course Interchange

A set of 4 to 7 text files is used to describe a course's content and structure (a Course Interchange File set). A CMI system must be able to create and interpret course interchange file sets for import and export operations (i.e. course interchange). The table below depicts the files used in a course interchange file set.

Files used in a Course Interchange File set

File Type	Data Elements & Description (See section below)	File Set Extension	Data Table	Obligation
Course Description (CRS) File	8.4.1	{filename}.CRS	No	Mandatory
Assignable Unit (AU) File	8.4.2	{filename}.AU	Yes	Mandatory
Descriptor (DES) File	8.4.3	{filename}.DES	Yes	Mandatory
Course Structure (CST) File	8.4.4	{filename}.CST	Yes	Mandatory
Objectives Relationships (ORE) File	8.4.5	{filename}.ORE	Yes	Optional
Prerequisites (PRE) File	8.4.6	{filename}.PRE	Yes	Optional
Completion Requirements(CMP) File	8.4.7	{filename}.CMP	Yes	Optional

There is one course interchange file set per course. Files in course interchange file set must be named with the corresponding file extensions (shown in the table above). In order to be considered a valid course interchange file set, all of the following rules must be met:

Rule #1 - All files in the set must have the same base filename (depicted in the table above)
Rule #2 - All files in the set must be located in the same directory.
Rule #3 - All of the mandatory file types must be included with all required course data elements (see Chapter 3.0) and in the proper format (see section 8.4)
Rule #4 - The structure represented must follow the correct usage requirements for course data elements (see chapter 3.0).

There are three kinds of course elements that compose a course structure:

- Assignable Units
- Blocks
- Objectives

Course structures are logically organized around these three kinds of elements and the interpretation/creation of course definition files sets depends on this organization. Certain files in a set are tabular representations of data (i.e. "tables) – see table above. The files that represent data tables have a CMI system generated identifiers (see *Course Elements. System ID*) that identify records that are specific to individual course elements. The identifiers serve as an index to find data specific to a course element.

8.2.1 Course Structure Export

To export a course structure, the CMI system must create (export) a valid course interchange file set that accurately reflects the data stored internally (in the CMI system database) for the given course.

The CMI must do the following in order to create a Course Definition File Set:

- Create all of the required files (as described in section 8.2).
- For each Assignable unit in the course, the CMI must:
  - Generate a corresponding record in the Assignable Unit File (see section 8.4.2)
  - Generate a corresponding record in the Descriptor File (see section 8.4.3)
- For each Block in the course, the CMI must:
  - Generate a corresponding record in the Course Structure File (see section 8.4.4)
  - Generate a corresponding record in the Descriptor File (see section 8.4.3)

The CMI may do the following in order to create optional features in a Course Definition File Set:

- Create optional files (described in section 8.2) as needed.
- Add optional data elements to the Course Description File (see section 8.4.1)
- For each course element, the CMI may:
    - Generate a corresponding record in the Prerequisites File (see section 8.4.6)
    - Generate a corresponding record in the Completion Requirements File (see section 8.4.7)
- For each Objective in the course, the CMI may generate a corresponding record in the Objectives Relationships File.
- For each Objective in the course, the CMI must generate a corresponding record in the Descriptor File (see section 8.4.3)

8.2.2 Course Structure Import

To import a course structure, the CMI system must read a valid course interchange file set and build an internal representation (in the CMI system database) accurately reflects the logical structure and data for the course definition.

Since exported course structure file sets contain explicit references to Assignable Unit locations, it may be necessary to edit the following course data elements prior to import:

Course Elements.File Name
Course.Elements.Command Line

This editing may be done manually prior to the import process or in an automated fashion. Some possible scenarios for automated (AU location) updating are as follows:

- An installation process provided by the course developer
- Special import functionality in to the CMI system.

8.3 Conformance Requirements

There are three levels of obligation described in this binding specification:

- Mandatory
- Optional
- Extension

Mandatory means that the CMI must be able to import and export (create) a set of required course structure files (as described in sections 8.2) and support all mandatory course data elements in those files.

Optional means that a conforming CMI may be able to import or export (create) optional course structure files and support indicated course data elements. A conforming CMI may support many options. Course structure options are grouped in levels of complexity (see section 3.5). A CMI may support individual optional elements without supporting all elements defined in a course "level"
.
An *extension* is a course data element that is not described in this specification. Extensions may be supported by a CMI for course structure data import or export. However, extension course data elements may not perform an identical function as data elements defined in this specification; 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 CMI is non-conforming.

8.4 Course Structure Data Model Mapping

This section contains the mapping of the course structure data model elements (defined in section 3.0) to the file binding. The files are as follows:

- Course Description (.CRS) File
- Descriptor (.DES) File
- Assignable Unit (.AU) File
- Course Structure (.CST) File
- Objectives Relationships (.ORT) File
- Prerequisites (.PRE) File
- Completion Requirements (.CMP) File

The following is defined for each of the above files:

- A description of the file's purpose
- A list of course structure data model elements used
- The file's data format
- An example

8.4.1 Course Description (.CRS) File

Purpose

This file contains information about the course as a whole. It offers information that relates to more than just a single element in the course.

Course Structure Data Model Elements

The following table identifies the Fields, Data Model Names, Data Model Section reference, obligation, course level for the Course Description file.

Group Names & Keywords	Data Model Element	Section	Obligation	Course Level
[Course]	Course	3.1	Mandatory	1
Course_Creator	Course.Creator	3.1.1	Mandatory	1
Course_ID	Course.ID	3.1.2	Mandatory	1
Course_System	Course.System	3.1.3	Mandatory	1
Course_Title	Course.Title	3.1.4	Mandatory	1
Level	Course.Level	3.1.5	Mandatory	1
Max_Fields_CST	Course.Max Fields CST	3.1.6	Mandatory	1
Max_Fields_ORT	Course.Max Fields ORT	3.1.7	Optional	3b
Total_Aus	Course.Total AUs	3.1.8	Mandatory	1
Total_Blocks	Course.Total Blocks	3.1.9	Mandatory	1
Total_Objectives	Course.Total Objectives	3.1.10	Optional	3b
Total_Complex_Obj	Course.Total Complex Objectives	3.1.11	Optional	3b
Version	Course.Version	3.1.12	Mandatory	1
[Course_Behavior]	Course.Behavior	3.2	Mandatory	1
Max_Normal	Course.Behavior.Max Normal	3.2.1	Mandatory	1
[Course_Description]	Course.Description	3.3	Mandatory	1

File Format

The Course Description file is text formatted as datatype *CMIFormatINI.* (see section 9.0 - Datatypes)

Example
An example of a typical Course Description file is show below

Course Description (.CRS) File Example

```
[Course]
course_creator=ABC Airplanes, "Jason Doit, CIO", Taylor Belt, Criss Cross
course_id = A16.82.2003
course_system = C++ for most units, Delphi for management system
course_title = Principles of Airplane Design and Flight
level=3b
max_fields_cst=7
max_fields_ort = 5
total_aus = 36
total_blocks = 8
total_objectives = 46
total_complex_objectives = 5
version = 4.0
[Course_Behavior]
max_normal = 99
[Course_Description]
This course is designed to instill in the student a sense of wonder and amazement.  It
covers the principles of flight, putting the principles in historical context.  It
includes interactivity and multimedia.

When the student completes this course he will be able to complete a 100 question,
multiple choice test, with over 80% correct answers.  The test is included as lesson 36:
"Final Quiz."
```

8.4.2 Descriptor (.DES) File

Purpose
This file contains a complete list of every course element in the course. It is used as the basic cross-reference file showing the correspondence of system generated IDs with user defined IDs for every element.

Course Structure Data Model Elements
The following table identifies the Fields, Data Model Names, Data Model Section reference, Obligation, and Course Level for the Descriptor file.

Field Name	Data Model Element	Section	Obligation	Course Level
System_ID	Course Elements.System ID	3.4.1	Mandatory	1
Developer_ID	Course Elements.Developer ID	3.4.2	Mandatory	1
Title	Course Elements.Title	3.4.3	Mandatory	1
Description	Course Elements.Description	3.4.4	Optional	2

File Format
The Descriptor file is text formatted as datatype *CMIFormatCSV*. (see section 9.0 - Datatypes). All field name identifiers must be included in the header row. Note that the order of field name identifiers specify field position (i.e. "columns") in a record (i.e. a "row") and can be in any order. Unsupported data elements are represented as empty strings. Custom fields can be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields' functionality.

Example
An example of a typical Descriptor file is show below

Descriptor (.DES) File Example

Descriptor (.DES) File Example

```
"system_id","developer_id","title","description"
“Root”, “AP-PP-2003”, Modern Power Plants – Description and Operation, “This course covers
Pratt & Whitney jet engines.  It provides information on both how they are designed, and how
they may be operated.”

"A1","PP1-2","Power Plant Introduction","An overview of the operation of the primary systems in
the Pratt & Whitney PW2037 engine."

B1, “PP20-1”, “Power Plant Description”,,

"A2","PP2-1","Power Plant Fuel System","Fuel movement from the tank to the combustors."

"A3","PP3-1","Power Plant Oil System","Oil circulation system in the PW2037 engine."

“A4”, PP4-1, “Designing for the Future”, “A historical perspective on how these engines came to
be.”

“B2”, “PP20-2”, “Power Plant Operation”,,

“A5”,”PP5-2”, “Starting an Engine”, “A generic tutorial on what must be done in any airplane to
start one of these jet engines.”

A5, PP6-2, “From the Ground to Flight”, “How to operate, and what performance to expect, when
engines are in the ground and in flight.”
```

8.4.3 Assignable Unit (.AU) File

Purpose
Information relating to the assignable units (AU) in the course.

Course Structure Data Model Elements
The following table identifies the Fields, Data Model Names, Data Model Section reference, Obligation, and Course Level for the Assignable Unit file.

Field Name	Data Model Element	Section	Obligation	Course Level
System_ID	Course Elements.System ID	3.4.1	Mandatory	1
Type	Course Elements.Type	3.4.5	Optional	2
Command_Line	Course Elements.Command Line	3.4.6	Mandatory	1
File_Name	Course Elements.File Name	3.4.7	Mandatory	1
Max_Score	Course Elements.Max Score	3.4.8	Optional	2
Mastery_Score	Course Elements.Mastery Score	3.4.9	Optional	2
Max_Time_Allowed	Course Elements.Max Time Allowed	3.4.10	Optional	2
Time_Limit_Action	Course Elements.Time Limit Action	3.4.11	Optional	2
System_Vendor	Course Elements.Development System	3.4.12	Optional	2
Core_Vendor	Course Elements.Launch Data	3.4.13	Mandatory	1
Web_Launch	Course Elements. Web Launch Parameters	3.4.14	Mandatory	1
AU_Password	Course Elements.AU Password	3.4.15	Mandatory	1

File Format
The Assignable Unit file is text formatted as datatype *CMIFormatCSV*. (see section 9.0 - Datatypes). All field name identifiers must be included in the header row. Note that the order of field name identifiers specify field position (i.e. “columns”) in a record (i.e. a “row”) and can be in any order. Unsupported data elements are represented as empty strings. Custom fields can be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields’ functionality.

Example
An example of an Assignable Unit file is show below

Assignable Unit (.AU) File Example

Assignable Unit (.AU) File Example

```
"system_id", "type", "command_line", "Max_Time_Allowed", ”time_limit_action”, "file_name",
"max_score", "mastery_score", "system_vendor", "core_vendor”, “web_launch”, “AU_password”

"A11","B16-lesson","APU1 -nuv", "00:16:00", “Exit”, "APU1.EXE", 80, 80, "APW", , ,
”invasion1944”

"A12","test", "APU2 -nuv", "00:26:00", ”E,Message”, "APU2.EXE", 100, 90, "APW", "test = on",
”vendorparam = plato”, “strangelove”

"A13", "lesson", "ELEC -nuv", "00:28:00", ”E,N”, "ELEC1.EXE", 50, 50, "APW",
```

8.4.4 Course Structure (.CST) File

Purpose

This file contains the basic data describing the order and grouping of AU's in a course. It includes the definition of course elements contained in blocks. The order in which these appear in the file implies (but does not force) an order for presentation to the student.

Course Structure Data Model Elements

The following table identifies the Fields, Data Model Names, Data Model Section reference, Obligation, and Course Level for the Course Structure file. Note that there may be multiple instances of *Course Elements.Members.System ID* associated with a single instance of *Course Elements.System ID.*

Field Name	Data Model Element	Section	Obligation	Course Level
Block	Course Elements.System ID	3.4.1	Mandatory	1
Member	Course Elements.Members.System ID	3.4.16.1	Mandatory	1

File Format

The Assignable Unit file is text formatted as datatype *CMIFormatCSV*. (see section 9.0 - Datatypes). Note that each record may have a variable number of columns for *Course Elements.Members.System ID* (Including corresponding field name header). The maximum number of columns is determined by the header row.

Example

Course Structure File Example

```
"block","member","member","member","member"
"root", "B1",   "B2",   "B3",,
"B1",  "A1",   "A2",   "A3",,
"B2",  "A4",   "A5",   "A6",   "A7"
"B3",  "A8",   "A9", ,,
```

8.4.5 Objectives Relationships (.ORT) File

Purpose

The Objectives Relationship file defines the relationships of simple and complex objectives to assignable units and blocks.

Course Structure Data Model Elements

The following table identifies the Fields, Data Model Names, Data Model Section reference, Obligation, and Course Level for the Objectives Relationship file. Note that there may be multiple instances of *Course Elements.Members.System ID* associated with a single instance of *Course Elements.System ID*

Field Name	Data Model Element	Section	Obligation	Course Level
Course_Element	Course Elements.System ID	3.4.1	Optional	3b
Member	Course Elements.Members.System ID	3.4.15	Optional	3b

File Format
The Objectives Relationship file is text formatted as datatype *CMIFormatCSV*. (see section 9.0 - Datatypes). Note that each record may have a variable number of columns for *Course Elements.Members.System ID* (Including corresponding field name header). The maximum number of columns is determined by the header row.

Example
An example of an Objectives Relationship file is show below

Objectives Relationship File example
"course_element","member","member","member","member","member" "B13","J23","J24","J25",, "A48","J27","J28",,, "J16","J93","J94","J95",, "B14","J16","J26","J29","J30","J31" “J31”,”A15”,,,,

8.4.6 Prerequisites (.PRE) File

Purpose
Sometimes it may be desirable to prevent a student from entering a lesson or assignable unit until he has met certain prerequisites. This file allows that sort of constraint to be placed on each block or assignable unit (AU) in a course.

Course Structure Data Model Elements
The following table identifies the Fields, Data Model Names, Data Model Section reference, Obligation, and Course Level for the Prerequisite file.

Field Name	Data Model Element	Section	Obligation	Level
Structure_Element	Course Elements.System ID	3.4.1	Optional	2
Prerequisite	Course Elements.Prerequisite	3.4.16	Optional	2, 3b **

(** = See section 3.5.1 notes for additional information about this data element levels)

File Format
The Prerequisite file is text formatted as datatype *CMIFormatCSV*. (see section 9.0 - Datatypes). All field name identifiers must be included in the header row. Note that the order of field name identifiers specify field position (i.e. “columns”) in a record (i.e. a “row”) and can be in any order. Unsupported data elements are represented as empty strings.

Example

Prerequisite File Example
structure_element, prerequisite a2, a1 a3, a2 b1, a3 a6, b1 b2, a6

8.4.7 Completion Requirements (.CMP) File

Purpose
The Completion Requirements file is designed to allow the explicit specification of when an assignable unit, block or objective should be assigned a specific status when that status does not conform to the defaults. It is essentially an exception file. All field name identifiers must be included in the header row. Unsupported data elements are represented as empty strings.

Course Structure Data Model Elements

The following table identifies the Fields, Data Model Names, Data Model Section reference, Obligation, and Course Level for the Completion Requirements file.

Field Name	Data Model Element	Section	Obligation	Level
Structure_Element	Course Elements.System ID	3.4.1	Optional	2
Requirement	Course Elements.Completions.Requirement	3.4.18.1	Optional	2, 3a, 3b **
Result	Course Elements.Completions.Status if True	3.4.18.2	Optional	2
Next	Course Elements.Completions.Next AU if True	3.4.18.3	Optional	2
Return	Course Elements.Completions.Goto after Next	3.4.18.4	Optional	2

(** = See section 3.5.1 notes for additional information about this data element levels)

File Format

The Prerequisite file is text formatted as datatype *CMIFormatCSV*. (see section 9.0 - Datatypes). All field name indentifiers must be included in the header row. Note that the order of field name identifiers specify field position (i.e. “columns”) in a record (i.e. a “row”) and can be in any order. Unsupported data elements are represented as empty strings. Custom fields can be added to support vendor specific extensions but these must have corresponding field identifiers in the header row and must not duplicate or conflict with existing fields’ functionality.

Example

An example of an Completion Requirements file is show below

Completion Requirements file example
`Structure_Element, Requirement, Result, Next, Return` `A4, A4=F, Passed, A5, A4`

9.0 Data Types

All data types used in this specification are defined in the following section. All data types are character strings encoded per ISO-8859. Any ISO-8859 defined character set can be used. (ISO-8859 characters sets include US-ASCII as a subset)

Each data type has the following items to describe it:

Data type
Name of the data type defined

Description
A verbal description of the size and data formatting rules for a data type.

BNF Notation
This is a structured notation representing the format of the data in BNF (Backus-Naur Form). How to interpret BNF is described in section 10.0 *BNF Notation*. Data Types defined in this section may be also used is BNF statements as constructs (all other BNF constructs are described in section 10.0). The BNF notation takes precedence should it be in conflict with the verbal description of a datatype.

Size
Size limit for this data type

Examples.
Examples included in this section are surrounded by double-quotes (")s to indicate literal values. Unless otherwise specified. the double-quotes are not part of the values depicted. Comments describing the examples are indicated *italics* and are not part of the data values depicted

Data Types	
Data type	CMIBlank
Description	An empty string.
BNF Notation	""
Size	0 Characters
Examples	""
Data type	CMIBoolean
Description	A vocabulary of two words. ("true" or "false").
BNF Notation	"true" | "false"
Size	4 Characters
Examples	"true"
Data type	CMIComment4096INI
Description	A string composed of zero or more consecutive "comment statements". Comment statements are composed of the following items: • **Start tag** - The comment statement starts with integer number enclosed in angle brackets (i.e. "<1>"). This number is serialized for the next comment statement (i.e. the next comment would begin with a "<2>")

Data Types	
	Location Tag (optional) – The location tag indicates the location in the AU where the comment was made by the user. This optional tag is located immediately after the start tag. It is comprised of a the letter "L" followed by a period and an AU defined location enclosed in angle brackets (e.g."<L.some lesson location>") • **Body** - The body of the comment is included after the start tag. A comment may include any printable character except "<> [ ]". Embedded carriages, spaces, and tabs are also allowed. **End Tag** - The comment statement ends with a ".e" added to an integer number with enclosed in angle brackets (i.e. "<e.1>") The integer number in the end tag matched the start tag.
BNF Notation	*(("<" *1(DIGIT) ">") ["<" ("L" | "l") "." 1*(INI_CMT_OK) ">"] *(INI_CMT_OK | WHITESPACE) ("<e." *1(DIGIT) ">"))
Size	4096 Characters
Examples	"<1>The background color is too blue!<1.e><2>The CDU panel has the incorrect 'way points' displayed for this route. <2.e><3><l.slide #36>The CDU panel has the incorrect 'way points' displayed for this route. <3.e><4>The CDU panel has the incorrect 'way points' displayed for this route. <4.e>"
	"<1>The background color is too blue!<1.e> <2>The CDU panel has the incorrect 'way points' displayed for this route. <2.e> <3>The CDU panel has the incorrect 'way points' displayed for this route. <3.e> <4>The CDU panel has the incorrect 'way points' displayed for this route. <4.e>"
Data type	CMIDate
Description	A period in time of one day, defined by year, month, and day in the following numerical format YYYY/MM/DD.
BNF Notation	4DIGIT "/" 2DIGIT "/" 2 DIGIT
Size	10 Characters
Examples	"2002/05/01" *May 5th, 2002*
Data type	CMIDecimal
Description	A number that may have a decimal point. If not preceded by a minus sign, the number is presumed to be positive. Examples are "2","2.2" and "-2.2).
BNF Notation	["-"] *DIGIT ["." *(DIGIT)]
Size	1 to 255 characters
Examples	
Data type	CMIDirectoryNameFull
Description	Fully qualified Windows directory path specification with drive letter(s), directory path.

Data Types	
	<Drive Letter>**:**<directories>**** Embedded spaces in directory names are allowed. Non printable characters and **< > ? * " / \ :** are not allowed in directory names. Directory names are separated by \'s (back slashes). Leading and trailing spaces are not allowed around the back slashes. This data type may be up to 255 characters in size.
BNF Notation	2*1(ALPHA) ":\" ; Drive volume and root dir *(*1(ALPHA | DIGIT | FILE_SAFE) "\") ; zero or more directory names
Size	255 characters
Examples	
Data type	CMIFeedback
Description	A structured description of a student response in an interaction. The structure and contents of the feedback depends upon the type of interaction. CMIFeedBack sub datatype(s) are as follows (each one matching the various interaction types): Choice Fill-in Likert Matching Numeric Performance Sequencing Single character True/False
Data type	CMIFeedback:Choice
Description	Feedback is one or more single characters separated by a comma. Legal characters are "0" to "9" and "a" to "z". If all the characters must be chosen to assume the feedback is correct, then the comma-separated list must be surrounded by curly brackets: { }. If there are multiple possible correct responses, they are separated by semi-colons (";")s.
BNF Notation	ENUM | ("{" SEQ "}" *(";" "{" SEQ "}"))
Size	255 characters
Examples	"2;3;4;a;c" *2,3,4,a, or c are all valid choices*
	"{3,4,5};{2,4,b}" *3,4, 5 all selected or 2,4,b all selected are the possible correct answers.*
	"3;4;5" *3,4, or 5 selected are the possible correct answers.*
Data type	CMIFeedback:Fill-in
Description	A character string of up to 255 characters in length. After the first letter spaces are significant.
BNF Notation	*255(LCHAR)
Size	255 characters
Examples	"The procedure is not correct !"
	"The sequence should be 4-3-2-1 instead of 1-2-3-4"
Data type	CMIFeedback:Likert

Data Types		
Description	Single character. Legal characters are “0” to “9” and “a” to “z”.	
BNF Notation	DIGIT | LOWERCASE	
Size	1 character	
Examples	“1”	
	“a”	
Data type	CMIFeedback:Matching	
Description	One or more pairs of identifiers. Each identifier is a single letter or number (0 to 9 and a to z). The identifiers in a pair are separated by a period. Commas separate the pairs. If multiple pairs must be matched correctly to consider the interaction correct, then the comma separated list of pairs are surrounded by curly brackets “{ }”.	
BNF Notation	MSEQ | (“{“ MSEQ “}”	
Size	255 characters	
Examples	“2.a;3.b;4.c”	*2.a,3.b,4.c are all valid matches*
	“{3.c,4.d,5.e}”	*The match pairs 3.c,4.d, 5.e (as a group) .*
	“3.a”	*The match pair 3.a is the only correct answer.*
	“1.b, 2.e, 3.d”	*1.6, 2.e, or 3.d are all possible answers*
Data type	CMIFeedback:Numeric	
Description	A valid CMIDecimal value. This element may be up to 255 characters in length.	
BNF Notation	See CMIDecimal	
Size	255 characters	
Examples	“2.5”	
Data type	CMIFeedback:Performance	
Description	This is a very flexible format. Essentially an alphanumeric string of 255 characters or less.	
BNF Notation	*255(LCHAR)	
Size	255 characters	
Examples		
Data type	CMIFeedback:Sequencing	
Description	A series of single characters separated by commas. Legal characters are “0” to “9” and “a” to “z”. The order of the characters determines the correctness of the feedback.	
BNF Notation	(DIGIT | LOWERCASE) 1*(“,” (DIGIT | LOWERCASE))	
Size	255 characters	
Examples	“0,1”	
	“a,b,c,1,2”	
Data type	CMIFeedback:True-False	
Description	A true/false value of type CMIBoolean.	
BNF Notation	See CMIBoolean	
Size	4 characters	
Examples	See CMIBoolean	
Data type	CMIFeedbackCSV	
Description	A structured description of a response in an interaction. The structure and contents of the feedback depends upon the type of interaction.	

Data Types	
	CMICSVFeedBack sub datatype(s) are as follows (each one matching the various interaction types): Choice Fill-in Likert Matching Numeric Performance Sequencing Single character True/False
Data type	CMIFeedbackCSV:Choice
Description	Feedback is one or more single characters separated by a comma. Legal characters are "0" to "9" and "a" to "z". If all the characters must be chosen to assume the feedback is correct, then the comma-separated list must be surrounded by curly brackets: { }. If there are multiple possible correct responses, they are separated by semi-colons (";")s.
BNF Notation	ENUM | ("{" SEQ "}" *(";" "{" SEQ "}"))
Size	255 characters
Examples	"2;3;4;a;c" *2,3,4,a, or c are all valid choices*
	"{3,4,5};{2,4,b}" *3,4, 5 all selected or 2,4,b all selected are the possible correct answers.*
	"3;4;5" *3,4, or 5 selected are the possible correct answers.*
Data type	CMIFeedbackCSV:Fill-in
Description	A string up to 255 characters in length. After the first letter spaces are significant. Double quotes are not allowed.
BNF Notation	*255(CSV_OK | ",")
Size	255 characters
Examples	"The procedure is not correct !"
	"The sequence should be 4-3-2-1 instead of 1-2-3-4"
Data type	CMIFeedbackCSV:Likert
Description	Single character. Legal characters are "0" to "9" and "a" to "z".
BNF Notation	DIGIT | LOWERCASE
Size	1 character
Examples	"1"
	"a"
Data type	CMIFeedbackCSV:Matching
Description	One or more pairs of identifiers. Each identifier is a single letter or number (0 to 9 and a to z). The identifiers in a pair are separated by a period. Commas separate the pairs. If multiple pairs must be matched correctly to consider the interaction correct, then the comma separated list of pairs are surrounded by curly brackets "{ }". If there are multiple pair combinations that are possible correct responses then those combinations are separated by semi-colons ";".
BNF Notation	MSEQ | ("{" MSEQ "}" *(";" "{" MSEQ "}"))

Data Types		
Size	255 characters	
Examples	"2.a;3.b;4.c"	*2.a,3.b,4.c are all valid matches*
	"{3.c,4.d,5.e};{2a,6.b}"	*The match pairs 3.c,4.d, 5.e (as a group) or matched pairs 2.a,6.b (as a group) are the possible correct answers.*
	"3.a"	*The match pair 3.a is the only correct answer.*
Data type	CMIFeedbackCSV:Numeric	
Description	A valid CMIDecimal value. This element may be up to 255 characters in length.	
BNF Notation	CMIDecimal	
Size	255 characters	
Examples	"2.5"	
	3;4;5	
Data type	CMIFeedbackCSV:Performance	
Description	This is a very flexible format. Essentially an alphanumeric string of 255 characters or less. Double quotes not allowed.	
BNF Notation	*255(CSV_OK | ",")	
Size	255 characters	
Examples		
Data type	CMIFeedbackCSV:Sequencing	
Description	A series of single characters separated by commas. Legal characters are "0" to "9" and "a" to "z". The order of the characters determines the correctness of the feedback.	
BNF Notation	(DIGIT | LOWERCASE) 1*("," (DIGIT | LOWERCASE))	
Size	255 characters	
Examples	"0,1"	
	"a,b,c,1,2"	
Data type	CMIFeedbackCSV:True-False	
Description	A vocabulary limited to on of the following values: "true" or "false". The values are case insensitive and only the first character is significant. (But it is recommend to use the CMIBoolean values for greater compatibility)	
BNF Notation	("t" | "T" | "f" | "F") *3(CSV_OK)	
Size	4 characters	
Examples	"T"	
	"False"	
Data type	CMIFileNameFull	
Description	A fully qualified Windows file specification with drive letter(s), directory path, filename, and file extension (if any). <Drive Letter>:\<directories>\<filename> Embedded spaces in filenames and directory names are allowed. Non printable characters and **< > ? * " / \ :** are not allowed in filenames or directory names. Filename and Directory names are separated by \'s (back slashes). Leading and trailing spaces are not allowed for file name.	

Data Types	
BNF Notation	2*1(ALPHA) ":\" ; Drive volume and root dir *(*1(ALPHA | DIGIT | FILE_SAFE) "\") ; zero or more directories *1(ALPHA | DIGIT | FILE_SAFE) ; filename
Size	255 characters
Examples	BB:\some dir1\some dir 2\file.ext
	C:\
Data type	CMIFormatCSV
Description	A tabular representation of data in a text string. (Or the Contents of a CSV (Comma-Separated Value) formatted text file) This datatype is divided into records and those records into fields (i.e. "rows and columns"). A record is the data found on a single line (using a carriage-return/line feed as a end-of-line marker). A field is the data that is found between commas ","s (comma delimited) on the line. Field data may or may not be enclosed in double-quotes (""). Field data must be enclosed in double quotes if it contains leading/trailing spaces or commas (","s). Leading/trailing space on unquote field data is ignored. Field data may not contain double-quotes. The first line is called the "header" and contains a comma-separated list of the field identifiers. Field identifiers are not data but specify the name and position of each field in the following records (lines). Note the first two examples below. Both examples represent the same data even though the order is different. Note that a system interpreting CSV data must be able to parse the data in both cases and yield the same result. Refer to the BNF notation below (and section 10.0) for more detail.
BNF Notation	CSV_HEADER *CSV_RECORD
Size	Undefined
Examples	Field#3Name,Field#2Name,Field#1Name, Field#4Name Field#3-Rec1-Data, Field#2-Rec1-Data, Field#1-Rec1-Data, Field#4-Rec1-Data "Field#3-Rec2-Data", Field#2-Rec2-Data, Field#1-Rec2-Data, Field#4-Rec2-Data Field#3-Rec3-Data, Field#2-Rec3-Data, Field#1-Rec3-Data, Field#4-Rec3-Data
	"Field#1Name","Field#2Name","Field#3Name","Field#4Name" Field#1-Rec1-Data, Field#2-Rec1-Data, Field#3-Rec1-Data, Field#4-Rec1-Data Field#1-Rec2-Data, Field#2-Rec2-Data, Field#3-Rec2-Data, Field#4-Rec2-Data Field#1-Rec3-Data, Field#2-Rec3-Data, Field#3-Rec3-Data, Field#4-Rec3-Data
Data type	CMIFormatINI
Description	Contents of an "AICC style" INI formatted text file (or text string). The format used in this specification is a variation of the Microsoft Windows™ *.INI file format. It is organized as follows: • Groups • Keywords • Comments • "Free Form" Groups **Groups** are names enclosed in square brackets "[" "]". Groups contain *keywords.* Groups are essentially records and keywords are essentially fields. Groups must be unique. Should a Group name be duplicated, only the first instance is used. Each keyword within a single group must be unique. If keywords are duplicated within a group, only the first instance is used. (See datatype *CMIGroupINI*.)

Data Types	
	Keywords are assigned values. (i.e. "keyword = keyword value"). Leading and trailing "linear whitespace" (tabs and spaces) are not included in the value of keyword. **Comments** are any line within a group (or any line positioned before all groups) that has a semi-colon ";" as its first non-whitespace character. Comments are text that is of use to a human viewing a file. Programs processing the data in the file ignore them. **"Free-Form" Groups** represent the variation from Microsoft Windows™ *.INI file format. They are delimited in the same manner as Groups (with a name enclosed in square brackets), but the contents of this kind of group can contain free formatted text and it not restricted to "keyword=keyword value" format. Another distinction is that all the data contained in a "Free- Form" Group is treated as a single data element. The data begins at the first non-whitespace character after the group name and ends with the last non-whitespace character before the next group name (or end of buffer/file). Leading and trailing whitespace are not included in the value of a "Free-Form" group. Square brackets ("[ ]") are not allowed. (See datatype *CMIGroupFreeFormINI*.) See BNF notation below (and in section 10.0) for more details on formatting
BNF Notation	*(WHITESPACE | INI_COMMENT) *(CMIGroupINI | CMIGroupFreeFormINI)
Size	Undefined
Examples	; Comments can appear before [Core] ; and after group names. ; Comments can also appear before SCORE = 87 ; and after keywords. TIME = 00:25:30 ; Their existence is ignored LESSON_STATUS= I ; CORE_VEDNOR is a "Free-form" group ; [CORE_VENDOR] xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxx [CORE_LESSON]

Data type	CMIGroupINI
Description	An INI "group". This element is patterned after a "section" contained a Microsoft Windows INI format file. An INI group consists of the following elements: 1. One group name enclosed in square brackets ("[ ]") 2. Zero or more Keyword/value pairs (i.e. "keyword = value") 3. Zero or more comments. (a comment consists of a line with the first character being a semi-colon ";") Comments are not processed. Each of the above elements (Keyword/value pair, comment, section name) exists on a single line with leading trailing whitespace. Blank lines may existing

Data Types	
	between element. Please refer to BNF notation below (and in Section 10.0) for a more detailed definition of this type.
BNF Notation	INI_SECTION *(INI_NAME_VALUE | INI_COMMENT | WHITESPACE)
Size	Undefined
Examples	[CORE] Student_ID = jones-123 Student_NAME = Jones, Jackie J. LESSON_STATUS = NA ,A SCORE= TIME = 00:00:00 Credit = Credit
	[Some section] keyword1 = value 1
Data type	CMIGroupFreeFormINI
Description	A Freeform INI "group". This data type has a "group name" (like *CMIGroupINI*) but does not require the contents of the group to have name/value pairs. An Freeform INI group consists of the following elements: 1. One group name enclosed in square brackets ("[ ]") followed by a carrige return/linefeed. 2. Zero or more characters of INI_FREEDATA Please refer to BNF notation below (and in Section 10.0) for a more detailed definition of this type.
BNF Notation	INI_SECTION *(INI_FREEDATA)
Size	Undefined
Examples	[CORE_VENDOR] xxxxxxxxxxxx 01010101010101
	[CORE_VENDOR] keyword1 = value 1
Data type	CMIIdentifier
Description	A string with no white space or unprintable characters in it. Maximum of 255 characters.
BNF Notation	*255(VIEWABLE)
Size Limit	255 characters
Examples	"Student#*(&%^*(#^*(&Q"
	"Student#23423"
Data type	CMIIdentifierDevID

Data Types	
Description	Up to 255 alphabetic, numeric, or “”_ - { } ” characters with no spaces.
BNF Notation	*255(DIGIT | ALPHA | “_” | “-“ | “}” | “{“)
Size Limit	255 characters
Examples	“{E8128C30-6BF8-11cf-96FC-0020AFED9A65}”
Data type	CMIIdentifierGUID
Description	A 128-bit value that is universally unique. This 128-bit value can be generated using algorithms described in any of the following documents: • ISO-11578 • Draft RFC *UUIDs and GUIDs* by Paul J. Leach and Rich Salz. The value is represented by 5 hexadecimal numbers separated by dashes. The value may or may optionally be enclosed in curly braces “{}”. See BNF notation below for detailed formatting.
BNF Notation	[“{“] 8HEX “-“ 4HEX ”-“ 4HEX “-“ 4HEX “-“ 12HEX [“}”]
Size	36 characters
Examples	“{E8128C30-6BF8-11cf-96FC-0020AFED9A65}”
Data type	CMIIdentifierINI
Description	A string of up to 255 characters with no whitespace. Double quote (“)s are not allowed.
BNF Notation	*255(DIGIT | ALPHA | EXTENDED | CSV_SAFE)
Size	255 characters
Examples	“Jstudent_1234”
	“Student-12”
	“STUD_1”
Data type	CMIInteger
Description	An integer number from 0 to 65536.
BNF Notation	5*1(DIGIT)
Size	5 characters
Examples	65000
	“1”
Data type	CMILevel
Description	A string indicating the level of features in a course structure. Limited to the following vocabulary of values: “1” - *Level 1 course structure* “2” - *Level 2 course structure* “3a - *Level 3a course structure* “3b” - *Level 3b course structure*
BNF Notation	“1” | “2” | “3a” | “3b”
Size	2 characters
Examples	“3a” *Level 3a course structure*
	“1” *Level 1 course structure*
Data type	CMILogic
Description	A logical statement following the rules described in section 4.2.3.
BNF Notation	`*( Term *(Operator Term) )`
Size	255
Examples	“(A5=passed)&A8”
	“{A4,A3,A6}&(B2|B3)”

Data Types	
Data type	CMIScoreINI
Description	Empty string ("") or the following: A score made up of up to three decimal numbers, separated by commas. The order is significant. The first number represents the "raw" score, the second number represents the maximum possible score, and the third number represents the lowest possible score. Commas may have leading and trailing spaces.
BNF Notation	(CMIDecimal *2(*LWS "," *LWS CMIDecimal)) | ""
Size	255 Characters
Examples	"75,100,0" *- Raw score of 75, maximum of 100, minimum of 0*
	"75" *- Raw score of 75*
	"75,100" *- Raw score of 75, maximum of 100*
	"" *- No score*
Data type	CMISIdentifier
Description	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.
BNF Notation	("A" | "B" | "J" | "a" | "b" | "j") 1*5(DIGIT)
Size	6 characters
Examples	"a01"
	"B00005"
	"J1"
Data type	CMISInteger
Description	A signed integer number from –32768 to +32768.
BNF Notation	("-" | "+") 1*5(DIGIT)
Size	7 characters
Examples	"-16412"
	"+5"
~~**Data type**~~	~~CMIString255~~
~~**Description**~~	~~A set of ASCII characters with a maximum length of 255 characters.~~
~~**Bindings Used**~~	~~*255(LCHAR)~~
~~**Size**~~	**255 CHARACTERS**
~~**Examples**~~	
Data type	CMIString255CSV
Description	A set of characters with a maximum length of 255 characters. Carriage return, line feed, and double-quotes (")s are not allowed.
Bindings Used	*255(CSV_OK | ",")
Size	255 characters
Examples	
Data type	CMIString255INI
Description	A set of characters with a maximum length of 255.. Carriage returns and linefeeds are not allowed. All leading and trailing linear whitespace (tabs or spaces) are discarded if present.

Data Types	
Bindings Used	(*1(VIEWABLE) *(* LWS *1(VIEWABLE | LWS))) | ""
Size	255 characters
Examples	
~~**Data type**~~	~~CMIString4096~~
~~**Description**~~	~~A set of characters with a maximum length of 4096 characters.~~
~~**BNF Notation**~~	~~*4096(CHAR)~~
~~**Size**~~	~~4096 characters~~
~~**Examples**~~	
Data type	CMIString4096CSV
Description	A set of characters with a maximum length of 4096 characters. Carriage return, line feed, and double-quotes (")s are not allowed.
BNF Notation	*4096(CSV_OK | ",")
Size	4096 characters
Examples	
Data type	CMIString4096INI
Description	A set of characters with a maximum length of 4096 characters. Square brackets (" [ ] ")s) are not allowed. All embedded whitespace is included. All leading and trailing whitespace is discarded if present.
BNF Notation	(*1(INI_OK | "=") *(*WHITESPACE *1(INI_OK | "="))) | ""
Size	4096 characters
Examples	
Data type	CMIStudentName
Description	Last name, first name and middle initial. Last name and first name are separated by a comma. Alphabetic, space, period, dash, and upper-ASCII (per ISO-8859) characters are allowed. Embedded spaces are also allowed.
BNF Notation	(ALPHA | EXTENDED) *(ALPHA | EXTENDED | "." | "-"| LWS) "," *(LWS) (ALPHA | EXTENDED) *(ALPHA | EXTENDED | "." | "-"| LWS) (ALPHA | EXTENDED)
Size	255 characters
Examples	"Hyde, Jack Q."
	"Two-names, Kelly"
	"Wu , "
	"Schmidt, JF"
Data type	CMITime
Description	A chronological point in a 24 hour clock. Identified in hours, minutes and seconds in the format: HH:MM:SS.SS Hours and seconds shall contain two digits. Seconds shall contain 2 digits with an optional decimal point and up to two additional digits.
BNF Notation	2(DIGIT) ":" 2(DIGIT) ":" 2(DIGIT) ["." 1*2(DIGIT)]
Size	11 characters
Examples	"12:02:45.56"
	"12:03:45"

Data Types	
Data type	CMITimespan
Description	A length of time in hours, minutes, and seconds shown in the following numerical format: HHHH:MM:SS.SS. Where: **HHHH =** **Hours**. Hours shall contain a minimum of 2 digits and maximum of 4 digits. The range of allowable values for hours is 00 – 9999. Values for hours may have leading zeros. **MM =** **Minutes.** Minutes shall consist of 2 digits. The range of allowable values for minutes is 00 – 59. **SS =** **Seconds**. Seconds shall consist of 2 digits. The range of allowable values for seconds is 00 – 59. **.SS =** **Tenth/Hundredths of Seconds**. This is the only optional element for this data type. This element shall consist of 1 to 2 digits. The range of allowable values is 01 - 99. Note that single digit values have an implied trailing zero (e.g. ".1" and ".10" represent the same value)
BNF Notation	2*4(DIGIT) ":" 2(DIGIT) ":" 2(DIGIT) ["." 1*2(DIGIT)]
Size	13 characters
Examples	"12:02:45.56"
	"0012:02:45.56"
Data type	CMIurl
Description	A fully qualified URL (Uniform resource locator)
BNF Notation	PROTOCOL "://" (IP | DOMAIN_NAME) [":" PORT] URL_PATH
Size	255 characters
Examples	"http://somedomain.org/dir1/index.html"
	"https://somedomain.org/dir1/index.html"
Data type	CMIurlEncNVPairList
Description	A list of name/value (i.e. "name=value") pairs separated by ampersands ("&" s). The "name" represents data element (or variable) name and the "value" is the value held by the "name" variable. Both the "name" and the "value" are URL-encoded (see section 6.4.1.1) representations of the actual values.
BNF Notation	*(NVCHAR)"="1*(NVCHAR) *("&" 1*(NVCHAR)"="1*(NVCHAR))
Size	255
Examples	Name1=value1&Name2=value2
Data type	CMIVersionNumber
Description	A string indicating which version of this specification. *(CMI001 – CMI Guidelines for Interoperability)* is implemented. Values are limited to the current and previously released version numbers (see BNF notation below).

Data Types	
BNF Notation	“1.0” | “1.1” | “1.2” | “1.3” | “1.4” | “1.5” | “1.7” | “1.8” | “1.9” | “2.0” | “2.2” | “3.0” | “3.0.1” | “3.4” | “3.5” | “4.0”
Size	255 characters
Examples	“4.0”
	“3.5”
Data type	CMIVocabulary
Description	Used to attach specific vocabularies within contexts in a schema. Vocabulary words must be complete and exact matches to those below. See the each sub data type below for the valid list of vocabularies. Mode Status Exit Credit Entry Interaction Result Time Limit Action
Data type	CMIVocabulary:Credit
Description	A specific vocabulary limited to on of the following values: “credit” or “no-credit”. Case sensitive
BNF Notation	“credit” | “no-credit”
Size	9 characters
Examples	“credit”
	“no-credit”
Data type	CMIVocabulary:Credit-INI
Description	A vocabulary limited to on of the following values: “credit” or “no-credit”. The values are Case insensitive and only the first character is significant. (But it is recommend to use the complete values for greater compatibility)
BNF Notation	(“C” | “c” | “n” | “N”) *9(INI_OK)
Size	10 characters
Examples	“c”
	“Credit”
	“No-”
Data type	CMIVocabulary:Entry
Description	A specific vocabulary limited to on of the following values: “ab-initio”, “resume”, or "" - (empty string)
BNF Notation	“ab-initio” | “resume” | ""
Size	9 characters
Examples	“ab-initio”
	“resume”
Data type	CMIVocabulary:Exit
Description	A specific vocabulary limited to on of the following values: “time-out”, “suspend”, “logout”, or "" - (empty string)
BNF Notation	“time-out” | “suspend” | “logout” | ""
Size	8 characters

Data Types	
Examples	“time-out”
	“logout”
Data type	CMIVocabulary:Interaction
Description	A specific vocabulary limited to on of the following values: “true-false” “choice”, “fill-in”, “matching”, "performance", "likert", "sequencing", or "numeric".
BNF Notation	“true-false”	“choice”	“fill-in”	“matching”	"performance"	"likert"	"sequencing"| "numeric"
Size	11 characters
Examples	“matching”
	"numeric"
Data type	CMIVocabulary:Mode
Description	A specific vocabulary limited to on of the following values: “normal”, “review”, or “browse”. All values are case sensitive.
BNF Notation	“normal”	“review”	“browse”
Size	6 characters
Examples	“normal”
	“browse”
Data type	CMIVocabulary:Result
Description	A specific vocabulary limited to on of the following values: “correct” “wrong”, “unanticipated”, “neutral”, or a valid CMIDecimal value.
BNF Notation	“correct”	“wrong”	“unanticipated”	“neutral”	CMIDecimal
Size	255 characters
Examples	“correct”
	“3.5”
Data type	CMIVocabulary:Status
Description	A specific vocabulary limited to on of the following values: “passed”, “completed”, “failed” , “incomplete”, “browsed”, or “not attempted”
BNF Notation	“passed”	“completed”	“failed”	“incomplete”	“browsed”	“not attempted”
Size	13 characters
Examples	“passed”
Data type	CMIVocabulary:Time Limit Action
Description	A specific vocabulary limited to one of the following values: “exit,message”, “exit,no message”, “continue,message”, or “continue,no message”
BNF Notation	“exit,message”	“exit,no message”	“continue,message”	“continue,no message”
Size	16 characters
Examples	“exit,message”
	“continue,no message”
Data type	CMIVocabulary:Why Left
Description	A specific vocabulary limited to following values: “student selected”, “lesson directed”, ”exit”, or “directed departure”.
BNF Notation	“student selected”	“lesson directed”	”exit”	“directed departure”.
Size	18 characters
Examples	“student selected”
	”exit”

Data Types	
	"directed departure".
Data type	CMIVocabularyINI
Description	Used to attach specific vocabularies within contexts in a schema. Vocabulary words must be complete and exact matches to those below. See the each sub data type below for the valid list of vocabularies. Mode Status Exit Credit Entry Interaction Result Time Limit Action
Data type	CMIVocabularyINI:Credit
Description	A vocabulary limited to on of the following values: "credit" or "no-credit". The values are Case insensitive and only the first character is significant. (But it is recommend to use the complete values for greater compatibility)
BNF Notation	("C" | "c" | "n" | "N") *9(INI_OK)
Bindings Used	File, HACP
Size	10 characters
Examples	"c"
	"Credit"
	"No-"
Data type	CMIVocabularyINI:Entry
Description	A vocabulary limited to on of the following values: "ab-initio", "resume", or "" - (empty string). The values are Case insensitive and only the first character is significant. (But it is recommend to use the *CMIVocabulary:Entry* values for greater compatibility)
BNF Notation	("A" | "a " | "R" | "r") *9(INI_OK)
Size	10 characters
Examples	"A"
	"resume"
Data type	CMIVocabularyINI:Exit
Description	A specific vocabulary limited to on of the following values: "time-out", "suspend", "logout", or "" - (empty string). The values are Case insensitive and only the first character is significant. (But it is recommend to use the *CMIVocabulary:Exit* values for greater compatibility)
BNF Notation	("T" | "t" | "L" | "l" |"S" | "s") *8(INI_OK)
Size	8 characters
Examples	"Time-oUT"
	"L"
	"suspend"
Data type	CMIVocabularyINI:Interaction
Description	A specific vocabulary limited to on of the following values: "true-false" "choice", "fill-in", "matching", "performance", "likert", "sequencing", or "numeric".

Data Types	
	The values are case insensitive and only the first character is significant. (But it is recommend to use the *CMIVocabulary:Interaction* values for greater compatibility)
BNF Notation	(“t” | “c” | “f” | “m” | "p" | "l" | "s"| "n" | “T” | “C” | “F” | “M” | "P" | "L" | "S"| "N") *11(CSV_OK)
Examples	“MATCHING”
	“c”
	“performance”
Data type	CMIVocabularyINI:Mode
Description	A specific vocabulary limited to on of the following values: “normal”, “review”, or “browse”. The values are case insensitive and only the first character is significant. (But it is recommend to use the “complete” values for greater compatibility)
BNF Notation	(“n” | “r” | “b” | “N” | “R” | “B”) *7(INI_OK)
Size	8 characters
Examples	“normal”
	“B”
Data type	CMIVocabularyINI:Result
Description	A specific vocabulary limited to on of the following values: “correct” “wrong”, “unanticipated”, “neutral”, or a valid CMIDecimal value. The values are case insensitive and only the first character is significant. (But it is recommend to use the “complete” values for greater compatibility)
BNF Notation	((“c” | “w” | “u” | “n” |“C” | “W” | “U” | “N”) *22(CSV_OK))	CMIDecimal
Size	255 characters
Examples	“correct”
	“W”
Data type	CMIVocabularyINI:Status
Description	A specific vocabulary limited to on of the following values: “passed”, “completed”, “failed” , “incomplete”, “browsed”, or “not attempted”. The values are case insensitive and only the first character is significant. (But it is recommend to use CMIVocabulary:Status values for greater compatibility)
BNF Notation	(“P” | “p” | “N” | “n” | “F” | “f” | “C” | “c” | “I” | “i” | “B” | “b”) *12(INI_OK)
Size	13 characters
Examples	“pass”
	“p”
	“Not Attempted”
	“browsed”
Data type	CMIVocabularyINI:Time Limit Action
Description	A specific vocabulary limited to following values: “exit,message”, “exit,no message”, “continue,message”, or “continue,no message”. More explicitly : “exit” or “continue” followed by a comma (with leading trailing spaces) , further followed by “message” or “no message”. Each of the 4 values are case insensitive and only the first character is significant. (Note: It is recommend to use *CMIVocabulary:Time Limit Action* values for greater compatibility).
BNF Notation	(“e” | “E” | “c” | “C”) *10(CSV_OK) *(LWS) “,” *(LWS)

Data Types	
	(“M” | “m” | “N” | “n”) *10(CSV_OK)
Size	255 characters
Examples	“E,n”
	“exit,no message”
	“continue , no message”
Data type	CMIVocabularyINI:Why Left
Description	A specific vocabulary limited to following values: “student selected”, “lesson directed”, ”exit”, or “directed departure”. Each of the 4 values is case insensitive and only the first character is significant. (Note: It is recommend to use *CMIVocabulary:Why Left* values for greater compatibility).
BNF Notation	(“e” | “E” | “s” | “L” | “l” | “D” | “d”) *17(CSV_OK)
Size	18 characters
Examples	“student selected”
	“E”
	“Directed”
Data type	HacpCommand
Description	Message type included in a HACP request message. See section 6.4 for a description of each HACP message type. This datatype has a vocabulary of the following (case insensitive) values: GetParam PutParam ExitAU PutInteractions PutComments PutPath PutPerformance
BNF Notation	(“GetParam| “PutParam” | “ExitAU” | “PutInteractions” | “PutComments” | “PutPath” | “PutPerformance” | “getparam” | “putparam” | “exitau” | “putinteractions” | “putcomments” | “putpath” | “putperformance”)
Size	15 characters
Examples	“1”
	“0”
Data type	HacpErrorNumber
Description	Numbers corresponding to error conditions (see datatype *HacpErrorNumber*) in a HACP response message. See section 6.4.8 for a description of HACP error conditions. The *HacpErrorNumber*(s) corresponding to (datatype) *HacpErrorText* are as follows: 0 - Successful 1 - Invalid Command 2 - Invalid AU-Password 3 - Invalid Session ID

Data Types	
BNF Notation	(“0” | “1” | “2” | “3”)
Size	undefined
Examples	“1”
	“0”
Data type	HacpErrorText
Description	Text describing error conditions corresponding to error numbers (see datatype *HacpErrorNumber*) in a HACP response message. See section 6.4.8 for a description of HACP error conditions.
BNF Notation	(“Successful” | “Invalid Command” | “Invalid AU-Password” | “Invalid Session ID”)
Size	1
Examples	“1”
	“0”
Data type	HacpRequestMessage
Description	
BNF Notation	
Size	undefined
Examples	“1”
	“0”
Data type	HacpResponseMessage
Description	
BNF Notation	
Size	undefined
Examples	“1”
	“0”

10.0 Augmented Backus-Naur Form (BNF) Notation

Backus-Naur Form (BNF) is a structured notation for describing data formats. BNF has many variations. The BNF described in this section is an augmented form of BNF partially derived from *RFC1945 - Hypertext Transfer Protocol -- HTTP/1.0.* (Please note that this section will be the authoritative source for interpreting BNF notation in this document).

The BNF notation described in this section is used to define the formatting rules for all data types included this specification.

10.1 Augmented Backus-Naur Form (BNF) Constructs

This augmented BNF used in this specification includes the following constructs:

NAME = DEFINITION

The name of a rule is simply the name itself (without any enclosing "<" and ">") and is separated from its definition by the equal character "=". Whitespace is only significant in that indentation of continuation lines is used to indicate a rule definition that spans more than one line. Certain basic rules are in uppercase, such as SP, LWS, CRLF, DIGIT, ALPHA, etc. Angle brackets ("<" and ">") are used within definitions whenever their presence will facilitate discerning the use of rule names.

"literal"

Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

rule1 | rule2

Elements separated by a bar ("I") are alternatives, e.g. “Yes | no" will accept yes or no.

(rule1 rule2)

Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo | bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

***rule**

The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two.

[rule]

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

N rule

Specific repetition: "<n>(element)" is equivalent to "<n>*<n>(element)"; that is, exactly <n> occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

; comment

A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications

10.2 Basic BNF Rules

The following BNF rules are used to describe the more common data types (in this document) are also as the basic "building blocks" used to construct more complex rules in the following sections. All ASCII character code values shown are in decimal numbers. All extended range ASCII character codes (128 -256) must conform to IS0-8859 character sets.

```
CR          = < ASCII Character (13) -- carriage return >
LF          = < ASCII Character (10) -- linefeed >
SP          = < ASCII Character (32) -- space >
TAB         = < ASCII Character (9) -- horizontal-tab >
<">         = < ASCII Character (34) -- double-quote mark >
CRLF        = CR LF
UPPERCASE   = < any ASCII uppercase letter "A".."Z" >
LOWERCASE   = < any ASCII lowercase letter "a".."z" >
CTL         = < Control ASCII characters (0 – 31) and DEL (127) >
CTLEXT      = < Extended ASCII control characters 128 – 159 >
EXTENDED    = < Extended ASCII characters (160 – 255).
              Viewable per ISO-8859 defined character sets>

ALPHA       = UPPERCASE | LOWERCASE
DIGIT       = < any ASCII digit "0".."9" >
HEX         = DIGIT | “A” | “B” | “C” | “D” | “E” | “F”
ESCAPE      = “%” HEX HEX
LCHAR       = < All ASCII characters except CTL>
INTEGER     = 1*DIGIT
DECIMAL     = [“-“]*DIGIT ["."] 1*DIGIT
NUMERIC     = INTEGER | DECIMAL
LWS         = SP|TAB
VWS         = CR|LF
WHITESPACE     = SP|TAB|CR|LF
ID             = 1*255( DIGIT | ALPHA | “_” | “-“ )
DATE           = 4DIGIT “/” 2DIGIT “/” 2DIGIT
TIME           = (2DIGIT | 4 DIGIT) “:” 2DIGIT “:” 2DIGIT [“.” 1*2(DIGIT) ]
STIME          = 2DIGIT “:” 2DIGIT “:" 2DIGIT

FILE_SAFE   =  “.”|”;” | “{“ | “}” | “+” | “~” | “`” | “!” | “@” | “#” |
               “$” | “%” | “^” | “&” | “(“ | “)” | “_” | “-“ | “[“ | “]” |
               “=”

INI_UNSAFE  = “[“ | “]” | “=”

INI_SAFE    = <”> | “\” | “/” | “?” | “,” | “.” | “<” | “>” |
      ”:” | ”;” | “{“ | “}” | “+” | “~” | “`” | “!” | “@” | “#” | “$” | “%” |
      “^” | “&” | “*” | “(“ | “)” | “_” | “-“ | “|”

CSV_SAFE    = | “\” | “/” | “?” | “.” | “<” | “>” | ”:” | ”;” |
      “{“ | “}” | “+” | “~” | “`” | “!” | “@” | “#” | “$” | “%” | “^” | “&” |
      “*” | “(“ | “)” | “_” | “-“ | “|” | “[“ | “]” | “=”

SPECIAL        = INI_SAFE | INI_UNSAFE

VIEWABLE       = ALPHA|DIGIT|EXTENDED|SPECIAL
```

```
;
;   URL & HTTP Specific BNF
;

SAFE          = “$” | “-“ | “_” | “.”
UNSAFE        =
EXTRA         = “!” | “*” | “'” | “(” | “)” | “,”

SAFE_URL      = “$” | “-“ | “_” | “@” | “.” | “&” | “+” | “-“
EXTRA_URL     = “!” | “*” | “"” | “'” | “|” | “,”

NVCHAR        = ESCAPE | ALPHA | DIGIT | EXTENDED | SAFE

PROTOCOL      = < Case insensitive “http” or “https” >
IP            = 1*3(DIGIT) 3(“.” 1*3(DIGIT))
DOMAIN_NAME   = 1*(ALPHA|“-“) 1*(“.” 1*(ALPHA|“-“) )
PORT          = *DIGIT
SEGMENT       = *(ALPHA | DIGIT | SAFE_URL | EXTRA_URL | ESCAPE)
URL_PATH      = “/” *(SEGMENT)  *(“/” *(SEGMENT)  )
URL           = PROTOCOL “://” (IP | DOMAIN_NAME) [“:” PORT] URL_PATH

; Name/Value Pair list
NVPRLIST      = 1*(NVCHAR)“=”1*(NVCHAR) *(“&” 1*(NVCHAR)“=”1*(NVCHAR))
```

10.3 AICC Style INI Related BNF Rules

```
; non-whitespace characters allowed in INI format
INI_OK         = ALPHA|DIGIT|EXTENDED|INI_SAFE

INI_CMT_SAFE      = <”> | “\” | “/” | “?” | “,” | “.” |
                    ”:” | ”;” | “{“ | “}” | “+” | “~” | “`” |
                     “!” | “@” | “#” | “$” | “%” | “^” | “&” |
                     “*” | “(“ | “)” | “_” | “-“ | “|” | “=”

INI_CMT_OK  = ALPHA|DIGIT|EXTENDED|INI_CMT_SAFE


; text string with embedded spaces
INI_NV         = *1(VIEWABLE) *( *LWS *1(VIEWABLE) )

; a keyword/value pair i.e. “x = y”
INI_NAME_VALUE = *LWS INI_NV *LWS “=” *LWS INI_NV *LWS CRLF

; an INI comment
INI_COMMENT    = *LWS “;” *(VIEWABLE|LWS) CRLF

; AICC style INI Group Name
INI_SECTION    = *LWS “[“ 1*(INI_OK) “]” *LWS CRLF



; AICC style INI “free form” data
INI_FREEDATA   = *WHITESPACE *( INI_OK | “=” | WHITESPACE ) *WHITESPACE CRLF

; Normal Group
CMIGroupINI        = INI_SECTION *(INI_NAME_VALUE | INI_COMMENT | WHITESPACE)

; Free-Form Group
CMIGroupFreeFormINI = INI_SECTION *(INI_FREEDATA)

; Definition of AICC style INI file format
CMIFormatINI =    *( WHITESPACE | INI_COMMENT )
                  *(CMIGroupINI| CMIGroupFreeFormINI)

; Definition of AICC style INI file format
AICC_INI_FORMAT = *( WHITESPACE | INI_COMMENT )
                  *( INI_SECTION ( INI_FREEDATA |
                                   *(INI_NAME_VALUE |
                                         INI_COMMENT |
                                             WHITESPACE)
                                 )
                   )
                  *( WHITESPACE | INI_COMMENT )
```

10.4 HACP Related BNF Rules

```
; List of valid version names
;
CMIVER         = “2.0” | “2.1” | “2.2” | “3.0”| “3.0.1”|
                 “3.0.2”| “3.4”| “3.5” | “4.0”

; HACP Request Message related constructs
vCMIVER        = < url-encoded, CMIVER >
NmCOMMAND      = < url-encoded, case insensitive string, “command” >
NmVERSION      = < url-encoded, case insensitive string, “version” >
NmSESSION_ID   = < url-encoded, case insensitive string, “session_id” >
NmAU_PASSWORD  = < url-encoded, case insensitive string, “AU_PASSWORD”  >
NmAICC_DATA    = < url-encoded, case insensitive string, “AICC_DATA” >
VPASSWORD      = < Url-encoded, *255(LCHAR) >
vSESSION_ID    = < Url-encoded, *255(LCHAR) >
vAICC_DATA     = < Url-encoded, *AICC_INI_FORMAT >
vHACP_COMMAND  =   “GetParam| “PutParam” | “ExitAU” | “PutInteractions” |
                   “PutComments” | “PutPath” | “PutPerformance”

; == HACP Response Message related constructs ==
NmrAICC_DATA   = <case insensitive string “AICC_DATA” >
NmrError_Text  = <case insensitive string “error_text” >
NmrError       = <case insensitive string “error” >
NmrVersion     = <case insensitive string “version” >


Vendor_Error_Text = *255(INI_OK)

vERROR_TEXT     = “Successful” | “Invalid Command” | “Invalid AU-Password” |
                  “Invalid Session ID”

vERROR_CODE    = “0” | “1” | “2” | “3”

NVPAIR1        = NmCOMMAND “=” vHACP_COMMAND
NVPAIR2        = NmVERSION “=” vCMIVER
NVPAIR3        = NmSESSION_ID “=” vSESSION_ID
NVPAIR4        = NmAU_PASSWORD “=” vAU_PASSWORD
NVPAIR5        = NmAICC_DATA “=” vNmAICC_DATA

; Definition of HACP request Message
HACP_REQUEST = NVPAIR1 “&” NVPAIR2 “&” NVPAIR3 [“&” NVPAIR4] “&” NVPAIR5
               <All NVPAIR’s are “&” separated and can be in any order>
               <NVPAIR5 is not required for GetParam Messages>

; HACP response Message
HACP_RESPONSE =   NmrError “=” vERROR_CODE CRLF
                  [ NmrError_Text “=” vERROR_TEXT CRLF ]
                  [ NmrVersion “=” CMIVER CRLF ]
                  [ NmrAICC_DATA “=” [AICC_INI_FORMAT]  ]
                  < AICC_DATA name/value pair is required only
                  for GetParam response messages >
```

10.5 CSV Related BNF Rules

```
;
FIELD_NAME = (*255(ALPHA|DIGIT|CSV_SAFE)) |
             < Reserved AICC Header Name >

; All chars except CR LF, DEL, and <”>
CSV_OK = ALPHA | DIGIT | EXTENDED | LWS | CSV_SAFE
; Quoted or not quoted – embedded commas are allowed inside quoted
CSV_ELEMENT = ( *CSV_OK | ( <”> *(CSV_OK | “,”) <”> ) )
; Quoted or not quoted
HEADER_NAME = ( FIELD_NAME | ( <”> FIELD_NAME <”> ) )
; Comma separated list
CSV_HEADER = ( LWS HEADER_NAME LWS *( "," LWS HEADER_NAME LWS ) ) CRLF
; Comma separated list with leading/training linear whitespace
CSV_RECORD= ( LWS CSV_ELEMENT LWS *( "," LWS CSV_ELEMENT LWS ) ) CRLF


; AICC Comma Separated Values (CSV) Format definition
CSV_FORMAT = CSV_HEADER *CSV_RECORD
```

10.6 “AICC Script” BNF Rules

```
;  Format rules for an “AICC script” – statement for logical expressions
;  Used in completion requirements and/or Perquisites


Expr               = *( Term *(Operator Term) )

SystemID           = ( “A” | “B” | “J” | “a” | “b” | “j” ) 1*5(DIGIT)

Status             = “passed” | “completed” | “failed” | “incomplete” |
                     “browsed” | “not attempted” |  “P” | “p” | “C” | “c” |
                     “F” | “f” | “I” | “i” | “B” | “b” | “N” | “n”

Operator            = “&” | “|”

Factor              = SystemID |
                      ( “(“ Expr *( “,” Expr) “)” ) |
                      ( “{“ Expr *( “,” Expr) ”}” ) |
                      ( DIGIT “*” “{“ Expr *( “,” Expr) ”}” )

n-Term              = Factor *(Operator Factor)
unaryTerm           = “~” Factor
equTerm             = SystemID “=” Status

Term                = n-Term | unaryTerm | equTerm
```

10.7 Interactions related BNF Rules

```
PERF_SAFE     = “-“ | “_”
PERF_OK       = DIGIT | ALPHA | LWS | PERF_SAFE
PERF_VAL      = 1*PERF_OK
ENUM          = LOWERCASE | DIGIT
SEQ           = ENUM 1*(“,” ENUM)
MSEQ          = (ENUM “.” ENUM) *(“,” (ENUM “.” ENUM))
PSEQ          = (  [ ID “.” ] PERF_VAL ) *(“,” ( [ ID “.” ] PERF_VAL)  )

;True-False type
T_TYPE        = “0”|”1”|”t”|”f”|”T”|”F”
;Choice type
C_TYPE        = ENUM | ( “{“ SEQ “}” *(“;” “{“ SEQ “}”) )
;Fill-in type
F_TYPE        = [“<case>”] 1*CSV_OK
;Matching Type
M_TYPE        = MSEQ | (“{“ MSEQ “}” *(“;” “{“ MSEQ “}”) )
;Performance Type
P_TYPE        = PSEQ | ( “{“ PSEQ “}” *(“;” “{“ PSEQ “}”) )
;Likert Type
L_TYPE        = ENUM

RESPONSE            =  T_TYPE | C_TYPE | F_TYPE | M_TYPE | P_TYPE | L_TYPE

INTERACTION_TYPE    = (“T” | “t” | “F” | “f” | “M” | “m” | “P” |
                       “p” | “S” | “s”  | “L” | “l” | “C” | “c”) *11(INI_OK)


; Interactions fields Data types
InDATE                  = DATE
InTIME                  = TIME
InSTUDENT_ID            = ID
InCOURSE_ID             = ID
InLESSON_ID             = ID
InTYPE_INTERACTION      = INTERACTION_TYPE | NUMERIC
InINTERACTION_ID        = ID
InOBJECTIVE_ID          = ID
InCORRECT_RESPONSE      = <”> RESPONSE <”>
InSTUDENT_RESPONSE      = <”> RESPONSE <”>
InRESULT                = NUMERIC | “C” | “c” | “U” | “u” | “W” | “w” | “N” |
“n”
InWEIGHTING             = NUMERIC
InLATENCY               = STIME
```

```
AuSYSTEM_ID             = ALPHA 1*DIGIT
AuTYPE                  = *255(CSV_OK)
AuCOMMAND_LINE          = *255(CSV_OK)
AuFile_Name             = <URL or file spec – need BNF for each>
AuMastery_Score         = *DIGIT
AuMAX_SCORE             = NUMERIC
AuMAX_TIME_ALLOWED      = STIME
AuTime_Limit_Action     =
AuSystem_Vendor         = *255(CSV_OK )
AuCORE_VENDOR           = *255(CSV_OK | “’”)
```

11.0 Glossary

ASCII	American Standard Code for Information Interchange. The de facto standard for the code numbers used by computers to represent all the upper and lower-case Latin letters, numbers, punctuation, and certain device control codes. The original version of ASCII (US-ASCII) has only 128 codes defined. "Extended" (or internationalized) versions of ASCII contain the original 128 codes plus an additional 128 for a total of 256.
AU	Assignable Unit. A module of computer based learning content (or CBT) that can be launched and tracked by a CMI system . The smallest logical unit of learning content in a course.
CBT	Computer-Based Training. Learning material wholly (or partially) in computer media form. Commonly known as "learning content". Assignable units (AU's) are considered a type of CBT.
CMI	Computer Managed Instruction. A system for launching and tracking learning content. Commonly known as a Learning Management System (LMS)
Course	A logical collection of AU's with metadata describing organization, launch data, and sequencing rules.
ECMAScript	ECMAScript is the ISO standard version of JavaScript. In this document the use of the term "JavaScript" is actually a reference to ECMAScript.
HACP	HTTP/S-Based AICC/CMI Protocol.
HTTP	Hypertext Transfer Protocol.
HTTPS	Secure HTTP. HTTP protocol encrypted using secure sockets layer (SSL).
HTTP/S	HTTP or HTTPS
LMS	Learning Management System.
URL	Uniform resource locator.
URL-encoding	A method of encoding text for HTTP messages. *See section 6.4.1.1 URL-Encoding/Decoding*
US-ASCII	The original version of ASCII with only 128 defined codes. See ASCII.

12.0 References

ISO-8859 Information Processing -- 8-bit Single-Byte Coded Graphic Character Sets – Parts 1 thru 10.

ISO/IEC 11578 - Remote Procedure Call (RPC)
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=2229

Leach and Salz, Draft RFC, “UUIDs and GUIDs”, Feb 04 1998
http://www.webdav.org/specs/draft-leach-uuids-guids-01.txt

T. Berners-Lee et al, RFC 1945, " Hypertext Transfer Protocol -- HTTP/1.0", May 1996.
http://ietf.org/rfc/rfc1945.txt?number=1945

T. Berners-Lee et al, RFC 1738, "Uniform Resource Locators (URL)", Dec 1994.
http://ietf.org/rfc/rfc1945.txt?number=1945

US-ASCII Coded Character Set--7-Bit American Standard Code for Information Interchange, ANSI X3.4-1986.

INDEX