DOCUMENT NO. CMI001

CMI Guidelines for Interoperability
AICC

ORIGINAL RELEASE DATE 25-Oct-93
Revision 3.5 release 2 Apr 2001

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) 356-1136
Internet address: bergstroms@ricks.edu

PREPARED ON PC   FILED UNDER CMI18.DOC

Prepared by:

                  _____   _____

                  Jack Hyde   Date

                  Chairman CMI Subcommittee

                  FlightSafetyBoeing

                  (206) 662-8484

Approved by:

                  _____   _____

                  Chairman AICC   Date

ABSTRACT

This guideline first outlines and defines a number of Computer Managed Instruction (CMI) principles and terms. It then addresses three aspects of interoperability of Computer Managed Instruction systems. Guidelines appear for:

- Communication between a CMI system and a lesson

- Moving a course between different CMI systems

- Storing lesson evaluation data

Each of these aspects of interoperability depends upon files. Guidelines for the format and content of the files are also described.

KEY WORDS

CMI
Computer Managed
   Instruction
Course structure
Evaluation

File format
Guidelines
Lesson evaluation
Performance data
Student performance

REVISION HISTORY

NEW **(Version 1)**	October 1993
	The contents of this document represent the first official release, which was approved in September, 1993 at the General Meeting of the AICC.

REV 1.1	April 1994

Sec 5.0	Reformats and amplifies description of contents of the chapter.
Sec 5.0.3	Changes DOS environment variable approach to passing file name to lesson. Now requires PARAM$CMI as environment variable name.
Sec 5.1.1	Adds names of lesson status flags to the reference.
Sec 5.1.2	Reformats information, no change in content.
Sec 5.1.5	Amplifies definition and argument description.
Sec 5.2	Adds Student_Data group.
Sec 5.2.6	New group with new keywords.
Sec 6.1	Adds four keywords to track file version.
Sec 6.1.1	Adds explanation of why four new keywords are needed.
Sec 6.1.2	Adds the new version keywords.
Sec 6.2, 6.3	Redefines first record in the file. Reformats table display.
Sec 6.4, 6.5	Adds version number to title and table. Updates example.
Sec 6.6	Adds version number to title and table.
Sec 6.7	Adds version number to title.
Sec 7.1, 7.2, 7.3, 7.4	Adds STUDENT_ID field. Reformats table. Adds first record with field identifiers. Defines field identifiers.
Sec 8.1	Adds [STUDENT_DATA] group.
Sec 8.2, 8.3	Adds descriptive material. Adds version keywords. Reformats some tables. Removes line number

REV 1.2	July 1994

Pg i	Adds E-mail address for "Point of contact."
Pg iii	Adds request for E-mail address to registration form.

Sec 5.0.2	Removes incorrect statement that this document does not contain standards for student records generated by the CBT lesson.
Sec 5.0.3	Changes third method for passing data from CMI to CBT. Indicates PARAM.CMI file should be placed in the \WINDOWS directory instead of the working directory. Necessary because some Windows-based CBT lessons change the working directory when launched.
Sec 5.1.1	Amplifies the description of the usage rules to clarify and correspond to the definition.
Sec 5.2.1	Adds description of the "Score" keyword. Inadvertently left out on previous revisions.

REV 1.3   March 1995

Sec 3.1	Added a description and rationale for preserving course behavior when using course interchange.
Sec 5.1	Add credit keyword to [Core] group. Moved the Path keyword from [Private_Area] to [Core]. Removed the [Private_Area] group. Moved language keyword from [Student_Demographics] to [Student_Preferences] Change arguements of Lesson_Mode to relate to lesson behaviors. Separate lesson behavior from CMI mode. Two separate concepts. Lesson behaviors can be set independently from CMI mode. Appropriate lesson behaviors are preset for each CMI mode. The behavior for each mode is found as an argument (for keywords Normal_Mode_Default, Browse_Mode_Default, and Review_Mode_Default) in the course interchange Course file.
Sec 5.1.1	Define 5 lesson behaviors for Lesson_Mode keyword.

- Sequential
- Test
- Choice
- X-test
- Fast

	Lesson_Status now includes only one flag -- the resume flag. Added a new status of BROWSED
Sec 5.1.1.1	New section to define CMI modes, and the defaults for when they are to be used.
Sec 5.1.1.2	New section to define special cases where CMI modes may not be the same as the defaults defined in 5.1.1.1.
Sec 5.1.1.3	New section to revise the status vs. modes discussion. Revised table with the addition of Credit and Lesson Mode columns.

Sec 5.1.6	This section formerly covered the [Private_Area] group.
Sec **5.1.6**	Local_ID was removed and J_ID definition changed.
Sec **5.1.7**	Cumulative_time keyword was removed.
	Time_limit_action keyword arguments modified to include NO_MESSAGE
Sec 5.2	Local_ID removed from [Objectives_Status]
	Moved language keyword from [Student_Demographics] to [Student_Preferences]
Sec 5.2.1	Lesson_Status has 3 flags

- Time-out: same as before
- Suspend: new
- Logout: same as before

Sec 6.1	New group created: Course_Behavior. Reorder group names to place in alphabetical order.
Sec 6.1.3	Course_Behavior group has six new keywords defined.

• Browse_Mode_Default	• Normal_Mode_Default
• Browse_Mode_Type	• Review_Mode_Default
• Max_Normal	• Review_Mode_Type

Define arguments for browse and review mode types.
- Forced
- Not_allowed
- Allowed
- Suspend_only

REV 1.4   January 1996

Rev 1.4	This is a maintenance update to the document with mostly

- typographic error and formatting corrections,
- order changes (rearrangements), and
- minor amplifications and corrections.
  Pages 46 to 49
  Page 138
  Page 199

Sec 6.2	The only changes to the standard are the

- Addition of a field (Time Limit Action) to the assignable unit file
   Page 159
- Renaming of "Duration" to "Max Time Allowed" in the assignable unit file.
   Pages 155 to 156, and page 159

REV 1.5   January 1996

Several keyword format descriptions were modified to identify that their extensions should not be zero padded.

Sec 4.3.3	Clarify format of groups. Explicitly identify no spaces between brackets.
Sec 5.1.6	Objectives_Status keywords J_ID, J_Score, and J_Status extension definitions changed to eliminate zero padding. J_ID definition clarified.
Sec 5.1.7	Student_Data keywords Lesson_Status and Score extension definitions changed to eliminate zero padding.
Sec 5.1.9	Student_Preferences Window extension definition changed to eliminate zero padding.
Sec 5.2.4	Objectives_Status keywords J_ID, J_Score, and J_Status extension definitions changed to eliminate zero padding.
Sec 5.2.5	Student_Data keywords Lesson_Status and Score extension definitions changed to eliminate zero padding.
Sec 5.2.6	Student_Preferences Window extension definition changed to eliminate zero padding.
Sec 6.1.2	Course Keywords. Total_Aus – removed zero padding in AU identifiers.
Sec 6.1.3	Clarify note regarding CMI compliance on page **Error! Bookmark not defined.**.
Sec 6.3	System ID – removed the example with zero padding. Developer ID – Data format definition changed to exclude spaces.

REV 1.6	25 April 1996

Sec 6.6.3	Logic statement descriptions amplified and new examples added. New logic constructs include the use of "not" and "sets".
Sec 6.7	Added reference to logical notation in Section 6.6.3.

REV 1.7	2 January 1997

	Typographical errors corrected. Note, many typographical corrections are not marked with a revision bar.
Sec 5.1	Significant rewrite to this section to clarify concepts.
	CMI compliant features now fall into two categories: Required and Optional. Core is no longer a compliance category.
	Lesson_Mode keyword made optional.
	Score.1 keyword added to [Student_Data] summary list (corrected oversight).
	Age, Birth_Date, Race, Religion, and Sex removed from [Student_Demographics] group (change in guidelines).
Sec 5.1.1	Lesson_Mode made optional. Description of Lesson_Mode clarified.
	Lesson_Status clarified. Status Flag now required, no longer optional.
Sec 5.1.1.1	Section retitled and rewritten to eliminate ambiguity.
Sec 5.1.1.2	Section retitled and rewritten to eliminate ambiguity.
Sec 5.1.1.3	Section retitled and rewritten to eliminate ambiguity.
Sec 5.1.8	Age, Birth_Date, Race, Religion, and Sex removed from [Student_Demographics] group.

REV 1.8   3 March 1997

Sec 4.4	Emphasized limitation to comma-delimited ASCI file format – separator may be comma only. Some current software allows other separators.
Sec 5.1.1	Added an "A" flag to lesson_status. Amplified and added to examples.
Sec 5.1.1.2	Added note that lesson status may also be changed by using rules defined in the Completion Requirements file.
Sec 5.1.7	Mastery_Score: Identify where system keeps and finds mastery_score. Max_Time_Allowed: Identify where system keeps and finds. Time_Limit_Action: Identify where system keeps and finds.
Sec 5.1.9	Clarification of how system handles Student_Preferences (storing both keyword and argument as given by the lesson.) Removed 255 byte limit on size of student preferences. Added clarification of lesson and CMI responsibilities in handling student preferences. Added SPEED preference. Bookmark preference removed. Feature is more appropriate for Core_Lesson than Student_Preferences.
Sec 5.2.1	Lesson_Status: clarify that use of flags is required.
Sec 6	Removed version number from all file names. Files may be referenced to Specification Document version – they do not need separate version numbers.
Sec 6.0.3	Reduced levels of complexity for structure passing from 5 to 4. When the Completion Requirements moved from level 4 to level 2, there was no longer a need for a separate level 4. Redefined and described each of the 4 levels now supported.
Sec 6.2	Assignable Unit File: Clarify that fields may be in any order (as determined by first record).
Sec 6.3	Descriptor File: Clarify that fields may be in any order (as determined by first record).
Sec 6.4	Clarify that order is critical in Course Structure File and fields may not be in any order.
Sec 6.6	Prerequisites File: Clarify that fields may be in any order (as determined by first record). Added first record to example file.
Sec 6.6.1	Added clarification to defining AU and objective status.
Sec 6.6.3	Expanded description of logic statements to allow explicit identification of status of any structure element.
Sec 6.7	Completion Requirements File: Significant changes to allow definition of completion requirements for all structure elements; and to add a third field in each record. Third field allows specific statuses to be assigned based on completion logic.

Sec 8.1	CMI/Lesson Communication Files:Revised to reflect Student Preferences changes.
Sec 8.2	Course Structure Files: Revised to reflect 4 levels and changes in Completion Requirements file.

REV 1.9   22 May 1997

Sec 5.1.1.2	Added paragraph summarizing the principles resulting in the Lesson Status table. Moved Note to end of table.
Sec 5.1.1.3	Added note to end of Status and Behavior Relationships table.
Sec 6.0.1	Added paragraph emphasizing the importance of the order of records in the structure interchange tables.
Sec 6.1	Identified keywords that are optional for level 1 complaince.
Sec 6.1.2	Identified keywords that are optional for level 1 complaince.
Sec 6.1.3	Identified keywords that are optional for level 1 complaince.
Sec 6.2	Identified which field values are required for level 1 complaince. References to "non-existent files" changed to "hypothetical files" throughout the document.
Sec 6.3	Identified which field values are required for level 1 complaince.
Sec 6.4	Emphasized importance of maintaining order of records in file.
Sec 6.4.2	Amplified the example by adding a Course Descriptor file.
Sec 6.6	Removed references to 5 levels of course interchange.
Sec 6.7	Added rules on the number of completion logic statements allowed per course element, and the order of these statements. Added additional examples.

REV 2.0   1 February 1998

	This revision added information necessary to enable the operation of CMI over intranets and the Internet.
Appendix A	This is all new, and covers information to enable CMI integration with Web technology.
Glossary	New terms have been inserted. These new terms all relate to use of CMI with Web technology.

REV 2.1   1998 June 18

Sec 4.5	Created new section to emphasize existing size limits. Clarified and added two size limits for [comments] and [course_description] (4096 bytes.)
	Change keyword length and field size to 255.
Sec 5.1.1	Integer number. Clarify that range is –32,768 to +32,767.
	Lesson location: May be blank as well as 0. Change length (in Format) of student ID from 11 to 255 characters.
Sec 5.1.2	Change length of [Core_Lesson] from 254 to 4096 bytes.
Sec 5.1.3	Change length of [Core_Vendor] from 254 to 4096.
Sec 5.1.4	Change length of [Comments] group to 4096.
Sec 5.1.7	Clarify that **mastery score** must be supported completely. CMI must change status if necessary as a result of mastery score computations.
Sec 5.2.3	Change length of [Comments] to 4096.
Sec 5.2.6	Removed bookmark from student preferences. Overlooked when done in revision 1.8.
Sec 6.1.4	Course Description group: Changed size limit for line and added group size limit of 4096. . Remove description of line length limits and embedded carriage returns <cr>.
Sec 6.2	Core vendor: Clarify that is level 1 requirement though field may be blank. Change field length from 254 to 4096.
Sec 6.3	Developer ID. Clarify definition. Change Description field length from 254 to 255.
Sec 6.7	Completion Requirements File: Added a table representing the contents of the file to the beginning of the section. This makes it consistent with other sections.
Sec 7.1	Date field: Make year 2000 compliant. New format 1998/05/01.
	Change Date field in example file to make year 2000 compliant.
	Time: Make time unambiguous.
	Comment: Change the length of each field from 254 to 255.
Sec 7.2	Time: Make time unambiguous.
	Change Date field in example file to make year 2000 compliant.
	Objective_ID: Clarify relationship to Developer_ID.
Sec 7.3	Time: Make time unambiguous.
Sec 7.4	Time: Make time unambiguous.
	Change Date field in example file to make year 2000 compliant.
Appendix A.3.2	Change Web_Launch and AU_Password fields from 254 to 255
Appendix A.4	Carify how to construct the URL command line.
Appendix A.5	Correct content type to "application/x-www-form-urlencoded".

REV 2.2   1998 October 12

Sec 4.3.1	Remove limitation of 80 characters for an INI file line.

Sec 5.1.2	Change recommendation to requirement. CMI must hold [Core_Lesson] data for as long as the student is in the course.
Sec 6.1	Replace reference to 8-character filename. Remove implied limit of 8 characters. The filename is not limited except by operating system limits.
Sec 6.2	Replace reference to 8-character filename.
Sec 6.3	Eliminate requirement for line_number.
	Replace reference to 8-character filename.
Sec 6.4	Replace reference to 8-character filename.
Sec 6.5	Replace reference to 8-character filename.
Sec 6.6	Replace reference to 8-character filename.
Sec 6.7	Replace reference to 8-character filename.
Sec 7.1	Eliminate requirement for Line_Number.
Appendix A.3.2	Clarified definition of Web Launch Parameters to avoid confusion with AICC-required parameters.
Appendix A.4	Amplified description of URL command line to avoid confusion of required parameters and web launch parameters.
Appendix A.5.1	Corrected example to include %0A (line feed) as well as %0D (carriage return).
Appendix A.5.2	Remove option of using single character -- ASCII 13 -- for carriage return. Must use line feed with each carriage return.
	Remove reference to "request method."

REV 3.0   1999 September 1 & 21

Sec 1.0	Added a paragraph on the organization of the document.
Sec 5.0.3	Define a mechanism for finding PARAM.CMI that works for Windows NT as well as 95/98.
Sec 5.1.1	Redefine and significantly simplify Lesson Mode.
	Moved information on Lesson Status into this section.
	Added maximum and minimum to Score. Allow use of decimal in score.
	Changed format of time to allow up to 9999 hours. Changed seconds to allow inclusion of decimal point.
Sec 5.1.1.1	Removed. Section on complex modes and lesson behavior.
Sec 5.1.1.2	Removed. Determining Lesson Status.
Sec 5.1.1.3	Removed. Determining Lesson Behavior.
Sec 5.1.6	Corrected keyword format typos.
	Added minimum and maximum to objective score. Allow use of decimal in score.
Sec 5.1.7	Added minimum and maximum to objective score. Allow use of decimal in score.
Sec 6.0.1	Removed references to complex modes.

Sec 6.0.3	Change levels of complexity to 3 from 4.
Sec 6.1	Removed 5 course behavior keywords.
Sec 6.1.1	Removed. Version Sensitivity.
Sec 6.1.1	Redefined Level to incorporate 3 not 4.
Sec 6.1.2	Removed 5 course behavior keywords.
Sec 6.3	Corrected Description to 4096 characters instead of 255.
Sec 6.4	Corrected Course structure table.
Sec 6.5	Changed Objectives Relationship table to begin with Course_Element instead of Block.
Sec 6.6	Correct levels reference to 4.
	Remove Mode column from prerequisites file.
Sec 6.6.1	Added Browse to Incomplete statuses.
Sec 6.6.2	Removed. Entry Modes.
Sec 6.6.2	Renumbered 6.6.3.
	Removed never and added equals to logic operators.
	Removed references to level 4. Corrected to level 3A and 3B.
Sec 6.7	Added Next field to completion requirements file.
	Added forced Return field to completion requirements file.
	Corrected level references from 4 to 3A or 3B.
Sec 6.8	Removed mode from examples.
	Demonstrated how Next may be used for remediation.
Sec 7.2	Added Numeric as another type of interaction.
	Result field number may now have a decimal point in it. No longer limited to an integer from 1 to 100.
Sec 8.2	Removed 5 course behaviors.
	Removed line number in Descriptor file. Field now has up to 4096 characters.
	Changed title of Structure_Element column to Course Element to accurately reflect possible entries in the column.
	Added next column to Completion Requirements File.
	Changed references to level 4 to 3a or 3b.
Appendix A.4	Corrected examples to URL encoded.
Appendix B	Added. Specification for API usage with CMI.
Glossary	Amplified definition of URL encoded.

REV 3.0.1   1999 November 24

Sec 5.1.1	Clarify meaning of time – total time in assignable unit.
Sec 5.2.1	Clarify meaning of time –time for a single use: session time.
Sec 7.2	Added Response_Value field to avoid ambiguity of Weighting.
Sec 8.3	Added Response_Value to table.
Appendix B.3.8	Add 3 new error codes.

Appendix B.4   Rename time to total_time. Avoid API get and set time ambiguity.
                      Rename status to statuses. Consistent use of plurals for arrays.
Appendix B.5   Rename time to session_time. Avoid API get and set time ambiguity.
Appendix B.6   Change correct_response to responses with sub-elements of description
                      and value. Fix weighting ambiguity.
                      Change path to paths – array convention.
Appendix B.7   Add clarification that vocabulary is very literal.
Appendix B.8   Rename time to total_time. Avoid API get and set time ambiguity.
                      Objectives.status to objectives.statuses. Plural for array.
                      Rename time to session_time. Avoid API get and set time ambiguity.
                      Add responses, description, and value to Evaluation Data Table.
                      Change path to paths – array convention.

REV 3.0.2   2000 May 8

Sec 5.1.1   Added more examples to Scores.
Sec 6.7   Corrected first record of Completion Requirements File in tables.
                      Added graphics to clarify examples.
Sec 7.2   Amplify description of Type Interaction. Remove Unique type
                      (unanticpated may be a response, cannot be an interaction.)
                      Removed Response_Value.
Sec 7.4   Added definition for 4th reason for leaving.
Appendix A.6.2   Remove references to multiple PutParam's affecting subsequent
                      GetParams.
                      Emphasize that multiple PutPerformance's result in complete
                      overwrites.
Appendix B.3.5   cmi.element._count description amplified to make clear that count is
                          total elements, not the last number used in indexing the array.
                      Return value for a call for _children clarified to indicate an empty string
                          may mean support with no children or no support. Another call for
                          last error is required to determine which.
Appendix B.4   Corrected table: heading fix and matching call to name column.
                      Corrected data model in the table. Scores and statuses changed to single
                          values for both objectives and student_data.attempt_records. Tries
                          added to indicate which score or status is being reported.
                      Changed evaluation.path to evaluation.paths to be consistent with B.6.
                      Removed evaluation.performance because no defined way of passing
                          complex performance information with API.
Appendix B.5   Corrected table: heading fix and matching call to name column.

Appendix B.6	Corrected table: matching call to name column.
	Changed objectives to objectives_status to be consistent with B.4 and file-based usage.
	Changed interactions.responses.description to interactions.correct_responses.pattern to be more consistent with file-based terminology.
	Removed interactions.responses.value.
	Changed objective_ids to objectives.n.id to be consistent with B4, B5, and data modeling conventions.
	Changed objectives.id value type from CMIString256 to CMIIdentifier.
Appendix B.7	Corrected 3 Interaction data types to reflect file-based usage: multiple choice -> choice; fill in the blank -> fill-in; simple performance -> performance.
Appendix B.8	Changed evaluation.path to evaluation.paths to be consistent with other tables.
	Removed evaluation.performance because complex performance information is not supported by any defined API's.
	Changed name from interactions.responses.description to interactions.correct_responses.pattern to make more obvious relationship to file-based concepts.
	Removed interactions.responses.value.

REV 3.4   2000 June 27

Sec 5.1.1	Lesson Location: First entry may be blank or empty string. Not zero.
	Lesson_Status: Meaning of the absence of a flag explained.
	Score. Changed wording from "last attempt" to "last session." Changed "lesson" to "assignable unit." or AU
Sec 5.1.7	Attempt_Number: Differences between the meaning of attempts and sessions explained.
	Lesson_Status: Meaning of the absence of a flag explained.
Sec 5.2.1	Time: Clarification of how the AU and CMI treat the time data element.
Appendix A.5.2	In Usage rules, parenthetical description of white space corrected to exclude CR and LF.
Appendix B	In all API calls the word "null" is replaced by an empty string. This avoids ambiguity and is consistent with the original intent of using "null".
	Replaced multiple occurrences of "lesson" with "assignable unit" or "AU."
	Replaced multiple occurrences of "attempt" with "session."

Appendix B.2.2	Amplify and clarify the CMI responsibilities. Divide responsibilities into Launch, Communication, and Sequencing. Add Sequencing information.
Appendix B.2.3.1	Revise code to find the API. New code works with more browsers.
Appendix B.3	Added rule that array elements shall be added sequentially. Skipping index numbers and empty array elements are not allowed.
Appendix B.3.4	LMSFinish() shall return a value of "true" or "false." (Same as LMSInitialize().)
Appendix B.3.5	Corrected case of return value for lesson_status.
Appendix B3.8	Added error code 102. Server is busy.
	Added error code 402. Invalid SetValue, element is a CMI keyword.
	Added error code 403. Element is read only.
	Added error code 404. Element is write only.
	Added error code 405. Incorrect Data Type
Appendix B.4	Amplified introductin. Explained the different uses of the data element "comments." Changed the word lesson to assignable unit or AU.
	Corrected example calls for total_time.
	Amplified definition of total_time to better define behavior of LMS.
	Changed CMILocale to CMIString255
Appendix B.5	Corrected example for session_time.
	Changed CMILocale to CMIString255.
	Corrected comments. It is not an array, (but retains the plural form for consistency.)
Appendix B.6	Intent of this table and relation to Lesson Evaluation Data is explaned.
	Corrected mastery_time example call.
	Changed name of Student Data Collection table to Lesson Evaluation Data table to be consistent with naming used in other chapters of this document.
Appendix B.7	CMIFeedback description amplified.
	CMIString256 replaced by CMIString255 throughout document to reflect the actual number of bytes in the data type.
	CMITime definition clarified.
	Time Limit Action vocabulary more clearly defined. Ambiguity of how to order and separate elements eliminated.
Appendix B.8	Corrected reference to objectives.id in Lesson Evaluation Data table.
	Corrected data model name to objectives_status.
Appendix B.9	Create new table to consolidate AU to LMS communication and Lesson Evaluation data models.
Glossary	Added definition for API.
	Added to definition of CMI.
	Corrected definition of "core item."
	Added definition for LMS.
	Added definition for "session."
	Added definition for "session."

REV 3.5   2001 Apr 2

	The main purpose of this revision to to make the AICC API match the ADL SCORM version 1.1. The two changes necessary are to make the function calls all return a true false if not returning data, and to add an new data element called comments_from_lms. The other changes are corrections or amplifications.
Sec 5.1.1	Correct to say lesson_status must have a flag when appropriate. (No flag has meaning too. No flag means normal re-entry.)
Sec 6.4.2	Developer ID added to example table. Developer ID is required. Removed line_number from same table.
Sec 7.0	Removed line # from contents table.
Appendix A.4	Corrected table and example to show "http://" is required in URL.
Appendix B.3.2	Added return value to data elements that previously did not have one.
Appendix B.3.4	Corected parameter and added return value.
Appendix B.3.6	Added return value of CMIBoolean.
Appendix B.3.7	Added return value of CMIBoolean.
Appendix B.3.8	Corected parameter (from none to empty string).
Appendix B.4	Added comment_from_lms data element. Changed objectives.n.scores to objectives.n.score (single value to match SCORM.) Changed objectives.n.statuses to objectives.n.status (single value)
Appendix B.5	Changed objectives.n.scores to objectives.n.score (single value to match SCORM) Changed objectives.n.statuses to objectives.n.status (single value).
Appendix B.6	Corrected interactions.n.id value data type to CMIIdentifier (from CMIString255) Corrected paths.n.location_id data type to CMIIdentifier.
Appendix B.8	Changed objectives.n.scores and statuses to score and status.
Appendix B.9	Changed interactions.id data type to CMIIdentifier. Changes objectives.scores and statuses to score and status. Changed paths.location_id data type to CMIIdentifier.

TABLE of CONTENTS

Blank

1.0   INTRODUCTION

Purpose

CMI systems manage both courseware and students in a training environment. This document provides a description of how CMI systems can do this without requiring specific CBT courseware. This document formulates guidelines that when followed by the CMI and CBT system builders, enable any CMI system to manage any CBT course. In addition, there is a tutorial description of CMI functionality.

Scope

This document describes

- How CMI systems in general function

- How CMI systems can pass course structure and content to other CMI systems

- How CMI systems can work with different CBT systems

- How CBT systems can work with different data analysis tools

Organization

This document includes three main specifications or guidelines. The file-based specification and the CMI data model are defined in the first 8 chapters. Appendix A defines the HTTP guidelines for Web-based CMI systems. Appendix B defines guidelines for API-based (application programming interface) CMI systems.

2.0   **CMI OVERVIEW**

Purpose This chapter provides a description of the functions of Computer-Managed Instruction (CMI). CMI systems manage both courseware and students in a training environment.

This chapter describes both basic and advanced features of CMI. It does not attempt to describe any current commercially available CMI system. The chapter is intended to provide introductory material and an overview of what CMI systems can do.

It first outlines
- General Components of CMI

and then covers in greater detail, each of the following components.
- Course Structure Development
- Testing
- Roster Operations
- Student Assignment Management
- Data Collection and Management

2.1   General Components of CMI

Five CMI components

A CMI system is more than a scheduler of CBT materials. CMI systems are capable of managing both on-line (CBT) and off-line instructional activities and tests. In general a CMI system can have these five components:

1. A component used for the **development of course structures**. (Section 2.2 Development of Course Structures)

2. A **testing** component used for the development and administration of off-line and on-line tests. Testing can be handled via

    • the CMI system

    • a separate test system (off line)

    • traditional CBT.

    Each of these must be able to report test results to the CMI system. (Section 2.3 Testing)

3. A student **rostering** component enables entering student names and demographic data. Student classes and class assignments are also made with this component. (Section 2.4 Rostering Operations)

4. A component which provides **student assignment management** including:

-   Administrator/instructor functions to oversee the day-to-day training operations and intervene when necessary

-   Assignment manager functions to control student assignments based on sets of rules (both predetermined and user-defined)

-   Standard approach to lesson initiation to provide a method for the CMI system to start-up lessons from different CBT vendors

-   Student logon functions to control and manage student access, maintain student-accessible data records, and display the student's current assignment

(Section 2.5 Student Assignment Management)

5. A component which provides student **data collection and management** including:

-   Functions to collect and maintain performance data on students at all levels of courseware presentation

-   Functions to provide standard analyses on performance data collected

(Section 2.6 Data Collection and Management)

CMI terms   Because there are many different interpretations of terms used to discuss training activities, definitions of some of them appear in the Glossary near the end of this document.

Terms that are used in this section include:

> **assignable unit**
> **block**
> **course**
> **curriculum**
> **hierarchy**
> **lesson**

2.2   Development of Course Structures

The heart of the CMI system is its course development component. In order for CMI to manage student data and make student assignments it must have a well defined structure for, and hierarchy of training materials
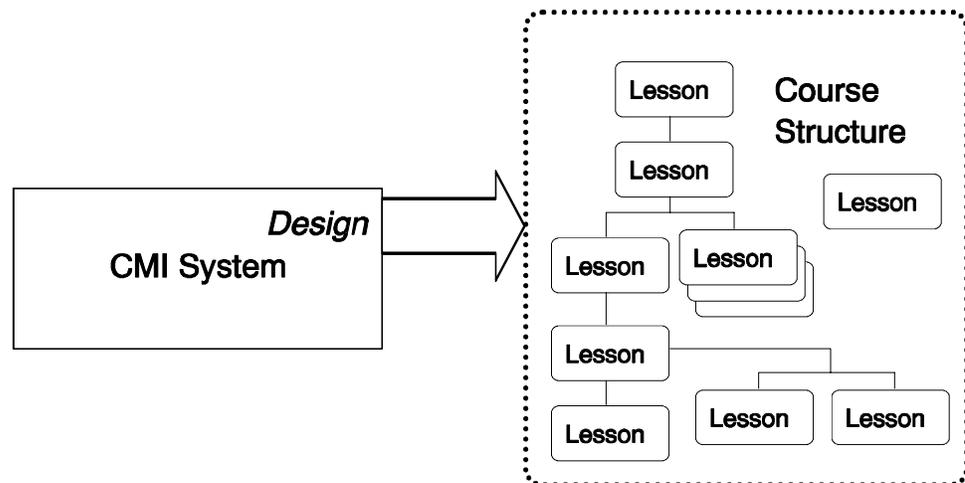
Lesson definition   Course structure tools include a method to define a list of lessons and their attributes. These tools work at the level of the smallest assignable and trackable unit (i.e. lesson).

Lesson attributes can include the following:

- associated training objective(s) and classification
- test id(s)
- number of attempts allowed
- lesson type (on-line/off-line)
- lesson assignment rules
- need for instructor intervention if all attempts failed
- resources required
- equipment, classrooms, instructors, consumables (e.g. workbooks)
- teaming requirements
- number of members of a team
- team resources
- remediation strategies and remedial lessons

Training objectives

Course structure tools include a method to define training objectives. This includes the objective identifier and a statement or description of the objective.

Alternate treatments

Lessons could have alternate treatments which could be assigned by the CMI system. That is, lessons may behave differently depending on information passed to them from the CMI system. For instance, a lesson's alternate behaviors can be based on attempt number, student attributes, past performance, or course developer criteria.

Lesson assignment and availability

Flexible CMI systems are able to assign lessons from any CBT development system, or lessons from a combination of several different CBT development systems. Lesson completion data is passed back to the CMI system for tracking and reporting.

Lesson availability indicators can be set on specific lessons so that once completed, the student can select to view it again as reference material.

Hierarchies

Course structure provides a method to group lessons into sequences for assignment. This entails support for lesson hierarchies which allow the course developer to define predecessor and successor relationships. An example follows:

```
                        ┌──────────┐                              ┌──────────┐
                        │ Lesson 1 │                              │ Lesson 6 │
                        └────┬─────┘                              └──────────┘
         ┌───────────────────┼───────────────────┐
    ┌────┴─────┐        ┌─────┴────┐        ┌─────┴────┐
    │ Lesson 2 │        │ Lesson 3 │        │ Lesson 4 │
    └────┬─────┘        └──────────┘        └────┬─────┘
         └───────────────────┬───────────────────┘
    ┌──────────┐             │
    │ Lesson 7 │        ┌────┴─────┐
    └──────────┘        │ Lesson 5 │
                        └──────────┘
```
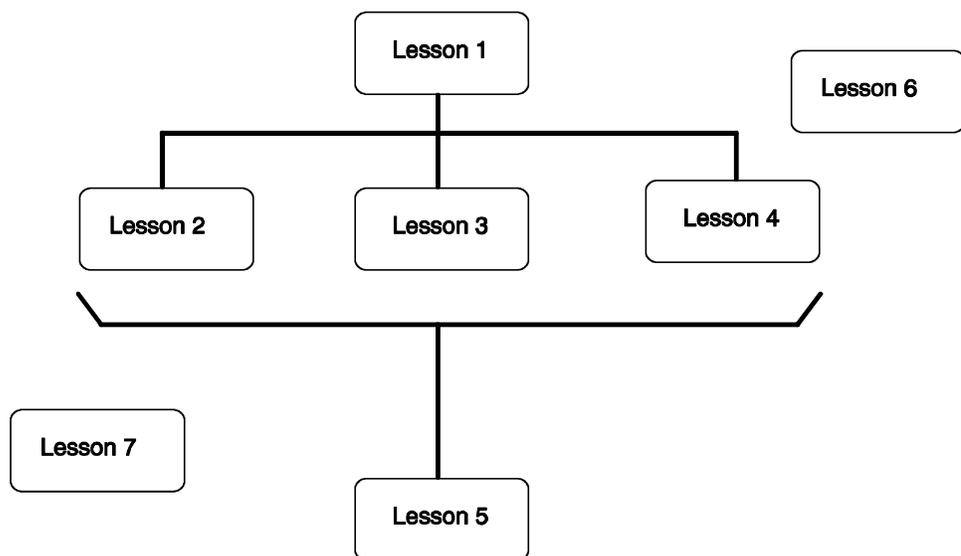
In this example, there are 7 lessons. Lesson 1 must be taken first. Lessons 2,3,4 may be taken in any order but must be taken before lesson 5 and after lesson 1. Lessons 6 and 7 are called floating lessons and may be taken at any time.

Block level The ability to define complex lesson hierarchies is important when managing resources. It is important to define multiple paths through the material to insure that bottlenecks are minimized.   In addition to lesson hierarchies, it is convenient to group lessons into a high level (blocks) and apply hierarchies at both the lesson and block level. Using the above hierarchy as block 1, an example would be:

```
          ┌─────────────────────────────────┐
          │  Block 1        ┌──────────────┐ │
          │                 │  Lesson 1    │ │
          │              ┌──┴───────────┐  │ │
          │           ┌──┴───────────┐  │  │ │
          │        ┌──┴───────────┐  │  │  │ │
          │        │              │  │  │  │ │
          │        └──────────────┘  │  │  │ │
          └─────────────────────────────────┘
                           │
              ┌────────────┴────────────┐
     ┌────────────┐              ┌────────────┐
     │            │              │            │
     │  Lesson 9  │              │  Lesson 8  │
     │            │              │            │
     └────────────┘              └────────────┘
```
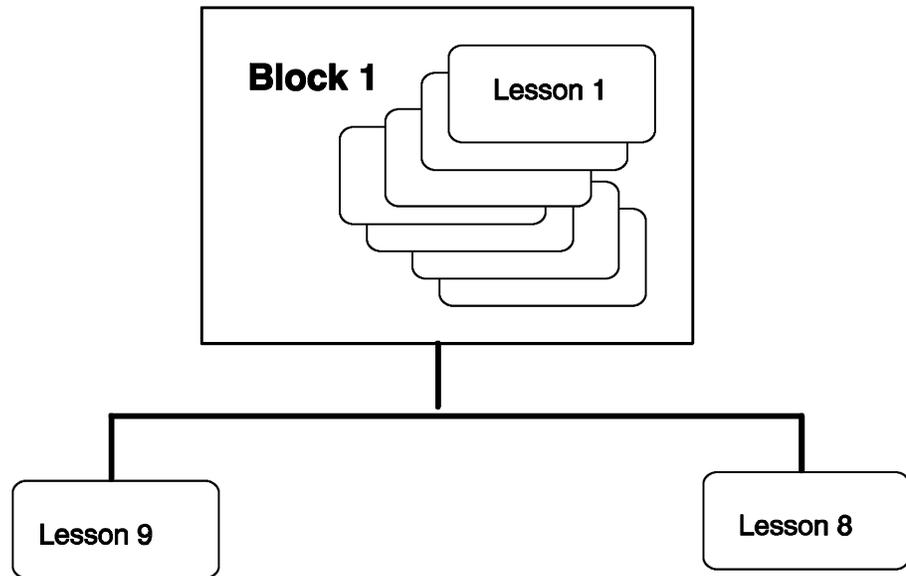
Resources Resources can be defined and managed by the CMI system. Resources are reusable (e.g. equipment) or consumable (e.g. paper handouts) with reorder points defined. Classrooms are part of the resource pool and scheduled along with the lessons using them. Resources such as instructors have qualifications associated with them. The CMI system may have a facility for maintaining qualification currency requirements and status.

Scheduling outside the CMI system

CMI systems are able to adapt to having some lessons in the course hierarchy (e.g. prerequisite lessons) which can be scheduled by an outside agency (e.g. a separate look-ahead scheduling system used to layout simulator usage). This can be done by defining another lesson attribute: on-demand and scheduled.

On-demand lessons are assigned by the CMI system if resources are available.

Scheduled lessons are assigned by an outside agency.

The CMI system can be informed that the scheduled lesson is complete by the student taking a CBT lesson or an instructor submitting an evaluation/completion form.

Other functions

Other functions can be supplied by a course structure component. The CMI system can provide support for shifts which is useful in maximizing utilization of resources.   Support for a calendar provides a method to define vacations, meetings, and course milestones.   Support for a student flow leveling algorithm provides smoother running lessons and courses.

The concept of testing points in the course hierarchy, as opposed to associating tests with the end of a lesson, provides a great deal of flexibility. This allows testing to take place after a group of lessons is completed and enables remediation before the student can continue.

Support for loading the course structure data-base from a data interchange file provides a means to transfer course data between different CMI systems and between task analysis systems and CMI systems.

2.3   Testing

Testing is often done as part the lesson and is usually handled separately as a function of the CBT system. Making test definition part of the CMI system provides more flexibility in test administration, tracking, and data analysis and does not preclude the use of the CBT system in development of the test if it is to be on-line.

Test types   Types of tests include mastery, performance checklists, and attitude questionnaires. By making an attitude questionnaire a type of test, the questionnaire can be evaluated using the item analysis facilities of the testing system and not require a separate set of evaluation tools. Tests are administered either on-line (from the CBT system) or off-line (e.g. paper or performance). Off-line tests may be scored by an optical scanner.

Test items   Tests are composed of test items. Test items relate to objectives. CMI systems support both norm-referenced and criterion-referenced tests. For mastery tests, use of both item pools and test forms is useful. Flexible test scoring strategies provide the following in addition to the standard percent correct:

- Number of items within an objective to pass the objective

- Number of objectives within the test to pass the test

- Item weighting (especially useful for attitude questionnaires)

- Critical items (pass item to pass test)

- Critical objectives (pass objective to pass test)

Test assignment   Tests are assigned as pre-tests or post-tests. Results then can assign credit to lessons not requiring the student to take those lessons (e.g. pretest to credit lessons containing material already mastered). Results can also be used to de-credit lessons and enforce remediation of completed material.

Test data   The testing component can support data collection for item analysis. If tests are administered via traditional CBT lessons, there are mechanisms to dump item results into the CMI system for item analysis.

2.4   Roster Operations

**Student
registration**

CMI systems support registration or enrollment of a student in a course or courses. This "roster" component provides for the definition of basic student data (student id and student name) and the course(s) in which he is enrolled. This component can also provide for definition of additional student demographic data.

The CMI system may additionally provide mass-registration and/or self-rostering. Mass-registration (or batch registration) provides a method to register a whole class or learning center with a single transaction. Self-rostering (or self-registration) is a method for the student to enroll himself into a course without instructor/administrator intervention.

**Student disenroll-
ment**

Support is provided for disenrollment of a student from a course. Disenrollment allows removal of a student from a course either with retention of performance data for later analysis, or complete deletion of data associated with the student.

**Administrative
reports**

Standard reports provide day-to-day administrative information in addition to the ad-hoc kinds of reports provided by all data-base management systems. Basic reports include:

- Course rosters (e.g. list of students in a specific course)
- Current assignment lists (current assignment of all students in a course)
- Resource utilization (resources in use and resources available)
- Student performance history (basic performance data for each lesson the student has completed)
- Course maps (graphical and descriptive)

Records transfer

The administrator or instructor can transfer student data records into and out of the CMI system from/to other sites if the training is geographically distributed.

2.5   Student Assignment Management

**Assignment
component**

The assignment management component can perform the day-to-day functions of the lesson assignment and performance recording. It can be the primary interface to the various CBT systems providing presentation of CBT lesson material.

Assignable unit

This is where the concept of assignable unit (lesson) is important. The assignable unit is the smallest unit the CMI system assigns and tracks. A CBT system may allow smaller units for lesson material but results must be reported back to the CMI system to correspond to these assignable units.

Section contents

The assignment component supports both assignment by the instructor and by the system. It also provides access to the system by the student. These topics are discussed under the following headings:

- Instructor/Administrator Functions
- System Assignment Management
- Student Logon

2.5.1   **Instructor/Administrator Functions**

Day-to-day functions

In addition to lesson assignment and results recording, certain features assist instructors/administrators in their day-to-day duties. These include the following:

- Assignment certification/override.

- Instructor evaluations.

Assignment certification/ override

When a student has taken the maximum attempts on a lesson and still not passed the lesson, the student must still be able to move along in the course. This can be done by certifying that the student has finished the lesson. This can also be done by overriding the student to another lesson.

Support should be provided to allow an administrator or instructor to certify completion of a lesson and provide a status (i.e. pass/fail/ incomplete). In addition to a status a grade could also be certified if appropriate.

Assignment override would allow an administrator or instructor to change the system designated assignment to another lesson within a course or to change the student's assigned course.

Instructor evaluation

Since instructors play a major role in aviation training, CMI must support easy, flexible instructor evaluation input (e.g. simulator time, check ride). Instructors have the ability to easily define/modify instructor evaluation forms. Types of data include the following:

- Evaluator ID

- Task evaluated

- Date/time of evaluation

- Task-level pass/fail decision

- Pass/fail decision of each objective

- Rating given each item in the objective

- Test of any instructor comments

- ID, data, and time of any instructor who overrode the evaluation

2.5.2	System Assignment of Student

Routing

All CMI systems provide methods to rout students from one lesson to another. This functionality in a CMI system is called the "router."

Basic functions

The most basic assignment management functions are

- To record student progress on the current lesson.
- Determine the student's next assignment.
- Initiate that assignment.

More sophisticated

More sophisticated assignment managers determine student assignments based on a student's individual requirements. This provides a means to tailor the course to the student's needs. At a minimum the system provides the ability to skip lessons and go directly to tests. Additionally, based on a pretest it can provide for customized assignment strategies.

Most sophisticated

Another level of sophistication allows the assignment manager to select lessons based on certain criteria such as

- Student's past performance.
- Demographic data (e.g. language, experience).
- Airplane configuration (e.g. Pratt & Whitney vs. GE engines).

Resource allocation

Another component of an advanced assignment manager is the resource allocation model. This model maximizes the use of resources based on current utilization. The resource allocator has the ability to bottleneck (stop progress in the course) a student if no lessons are available for assignment due to lack of resources. Resource allocation algorithms use resource weighting (e.g. assign most critical resources last or first) and past resource availability data for determining the lesson assignment rules.

student perfor-
mance data
The assignment manager maintains the student's performance data records. These data records include the following information:

- lesson completions

- lesson transaction data
  - attempt number
  - time on lesson
  - objective pass/fail status by objective id
  - lesson pass/fail status
  - lesson score (if applicable)
  - lesson status (e.g. bottlenecked, in progress, complete, etc.)
  - date started and date completed

- current resources if lesson currently in progress

- administrative information
  - biographical/demographic data
  - shift
  - current classroom/instructor
  - course completion data (e.g. pass/fail, dates)

2.5.3   Student Logon

Student access to training

The CMI system provides a single point of entry for student access to training materials. The main function of student logon is to display the student's assignment and then to initiate that assignment (if it is a CBT lesson or test.)

Security

This component provides a level of security for the lesson materials by displaying to the student only data to which he has access. This data includes:

- Past performance history.

- Current assignment.

- Current position in course/curriculum.

- Graphical course map designating what is complete and what remains.

- Possible other assignments.

Mail

Student logon provides a central place for the student to send and receive mail from instructors and other students. It also provide a note facility.

Self rostering

A student logon facility can also provide a central place for self rostering (or self registration) of the student into a course if the instructor has not already registered the student.

2.6   Data Collection and Management

Collection and reporting

The data collection component provides automated collection and management of data. This component also provides for both standard and ad-hoc reports on the data collected.

Types of data

The types of data collected can include the following:
- lesson and course summary data
- test item response
- student performance data
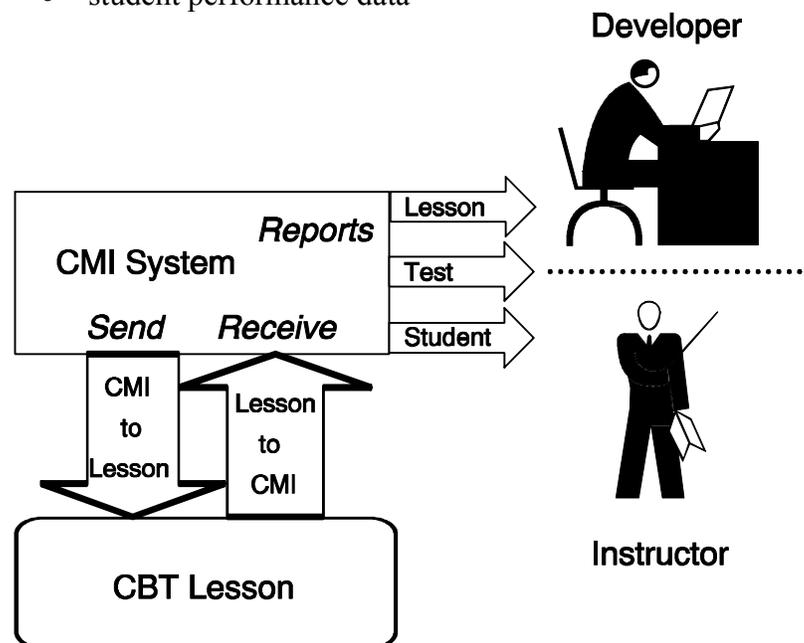
For developer

The lesson and course summary data is used by the instruction developer to evaluate the course, and determine what changes can improve the lessons.

For instructor

The student performance information is used by the instructor to help him provide his students the information they need to master the training material.

For both   The test data is used by both the instruction developer and the instructor.[1]

Levels of data   There are really two levels of data in the CMI system.

- The first is the relatively concise lesson-level and objective-level data required for assignment management and lesson routing.

- The second level of data is the more extensive test item and CBT frame and path data required for course and curriculum analysis.

---
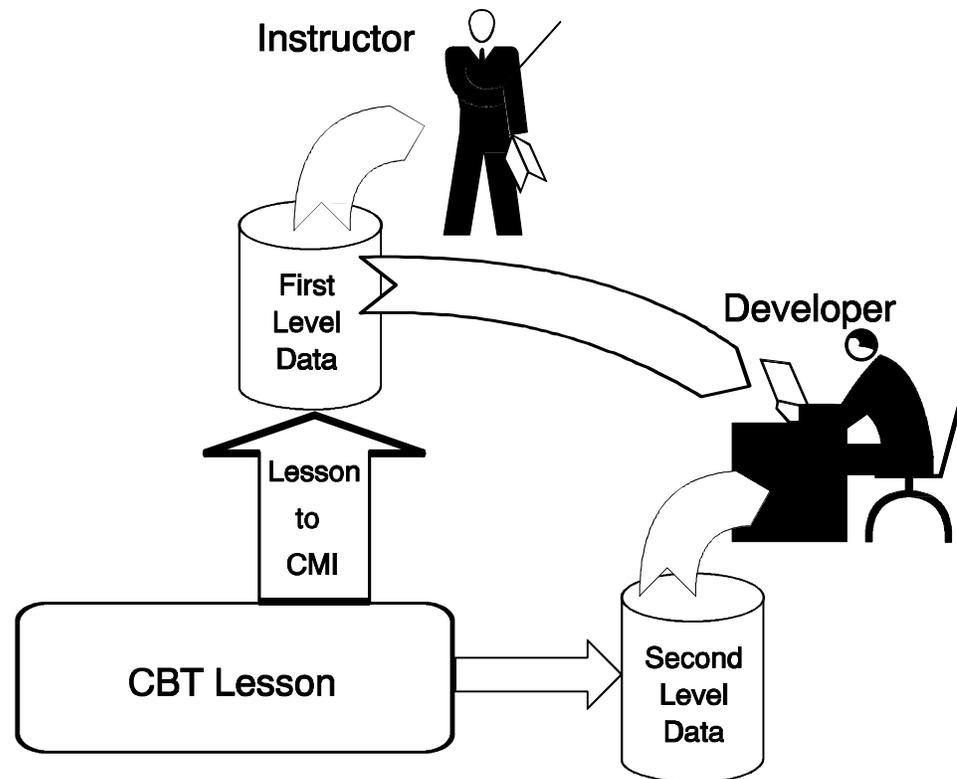
[1] The term "instructor" is used here rather loosely. In this context, the instructor is the person responsible for manageing the student. He may for instance, be called a site administrator instead of an instructor.

However the CMI system is implemented, there should be two data collection systems so that more extensive data (usually needed for initial courseware validation) does not interfere with the relatively smaller amount of data required for assignment management. By using separate data collection systems, data can selectively be turned off and on, based on evaluation and installation requirements.

Level-one data   Level-one data would probably always be turned on. However, there may be circumstances where it is desirable to turn off level one data, for instance:

- Legal reasons for not keeping some performance data.

- Administrative reasons for not wanting data on individual students.

- Course to be used for review (no need for lesson routing or performance data).

Level-two data   Level-two data would probably be turned on for all lessons during course small-group tryouts. Data would be evaluated and revisions made to the indicated lessons in the course. Then the level-two data collection would be turned off. When a new lesson revision was implemented, data collection would be turned on for a small sample size, data would be evaluated, and any indicated lesson revisions would be made.

Data feed   The data collected should be capable of being used with (or fed into) any of the standard statistical packages such as SPSS or SAS. The data collected should also be capable of being used by a data base management system (DBMS). Preferably the CMI system should use a relational data base management system (RDBMS) as the foundation for its file management.

PTT data   The evaluation component of CMI should be able to handle the types of data generated by CBT-driven part-task trainers(PTTs). This is handled relatively easily if data collection is structured so that evaluation data from on-line tests, off-line tests, PTTs, and simulators all have the same basic format. This significantly simplifies data handling and reporting.

Standard reports   The data collected should be used to provide standard reports for both courseware and student evaluation. More elaborate types of reporting could be accomplished using the data and the ad-hoc reporting capabilities of the DBMS or RDBMS. Standard reports should include the following:

- **First level:** Student performance history reports
- **Second level:** Lesson and course level analysis reports
- **Both levels:** Test analysis reports

Student perfor-mance reports

A tool for course instructors is a standard report on how well his students are performing. This would contains individual student data on each lesson by attempt.

The types of data that are useful for reporting include:

- Student ID or name
- Lesson identifier
- Lesson test identifier (if applicable)
- Attempt number
- Date completed
- Total time on attempt
- Lesson attempt pass/fail indicator
- Lesson attempt score (if applicable)
- Objectives failed
- Data collected.

Lesson/course analysis reports

A good tool for the courseware evaluator is a report providing a high-level overview of how each lesson in the course is progressing. This high-level data is useful to flag lessons which should be looked at in more detail (e.g. lessons with high failure rates, lessons containing an objective with a high failure rate, or very long lessons).

The types of data that are useful for reporting includes:

- Lesson times and scores including means and standard deviations
- Lesson failure rates
  - Failure rates by lesson attempt
  - Objective failure rates
- Lesson sample sizes

Test analysis reports

Another good tool for the courseware evaluator is a report on how the mastery tests are performing. Test item analysis reports can help identify unreliable test questions. It can also show how well success or failure of a test item correlates to success or failure on either the objective or the entire test. However, item-objective and item-test correlations are more conclusive when using standard forms of a test rather than testing from an item pool.

Types of data that are useful for reporting include:

- Sample size

- Sample time period (date range)

- Cause of test failures
    - Score
    - Objectives failed
    - Critical items or objectives

- Objective summary data (statistics by objective)
    - Number of failures
    - Mean score
    - Standard deviation
    - Number items in objectives/number need to pass objective
    - Reliability coefficient
    - Correlation of how performance on item relates to performance on test

- Item summary data (statistics by item)
    - Difficulty index (mean number students answering correctly)
    - Number of failures
    - Correlation of how performance on item relates to performance on objective and performance on test
    - Item choice distributions (how many times individual alternatives were chosen by students)
    - Unanticipated response statistics

3.0   INTEROPERABILITY OVERVIEW

The problem

In the past, authoring systems made the customer (an airline or manufacturer) a captive of his own CMI system. If the customer wanted to take advantage of CMI features in his courses, he had two choices.

1) Design his own CMI system with his authoring system tools, or
2) Purchase a CMI system from the same vendor who supplies the authoring system.

In either case, the system works only for a single vendor's CBT lessons. This is fine, until the customer aquires courseware designed with a different authoring system, from a different vendor.

Why incompatible courseware?

Several circumstances can motivate a customer to use CBT courseware incompatible with his CMI system.

- A manufacturer delivers incompatible courseware with a new airplane purchase.

- An airline purchases courseware from a vendor that used a different authoring system.

- A customer decides to design new CBT with a different authoring system.

Why a single CMI system?

There are many reasons a customer may wish to continue use a single CMI system instead of multiple systems to match his different CBT lessons.

- Instructors are familiar with the current CMI system, and training on a new system would take time. This impacts the speed with which new courseware can be used, and the cost of training how to use it.

- It is desirable to maintain the student's overall "look and feel" in the airline's courseware. (The CMI/student interface provides a significant part of the look and feel.)

- Maintenance of two different CMI systems is more complex than maintaining a single system.

- The current CMI system has features and functions not available with the CMI associated with the new courseware.

- There is a desire to add some new lessons designed with a different authoring system to an existing course. A single CMI system is desirable for the entire course.

Three aspects of interoperability

This chapter describes the three aspects of CMI interoperability covered in this guideline; and suggests reasons why these aspects of interoperability are desirable.

The three aspects or types of interoperability discussed are:

- Moving course structure, behavior, and content between systems

- Communication between a CMI system and a lesson

- Storing student performance data

3.1   Moving Courses

Definition of course

A course may be as simple as a few lessons to be viewed sequentially, or it may be as complex as hundreds of lessons, some of which are prerequisites to others and some of which may be experienced in any order. Basically, courses have three components: instructional elements, structure, and behavioral elements.

Course content

The instructional elements include all the lessons, tests, and other assignable units in the course -- often referred to as content. Frequently, a description of the content also includes all of the objectives to be mastered in the course.

Course structure

The structure determines the order in which these are to be experienced by each student. The order may as simple (as the implied order of a list of lessons) or quite complex, depending on prerequisites, or even student performance. The part of the CMI system that sequences the course content, is referred to as the *router*.

Course behavior

Behavior can be expressed as the progression logic philosophy for the whole instructional material of a course. It is determined by relations between CMI modes and lesson modes, or by a specific behavior description. It is the progression logic within a lesson, and also the progression logic from one lesson to another.

Structure   Content
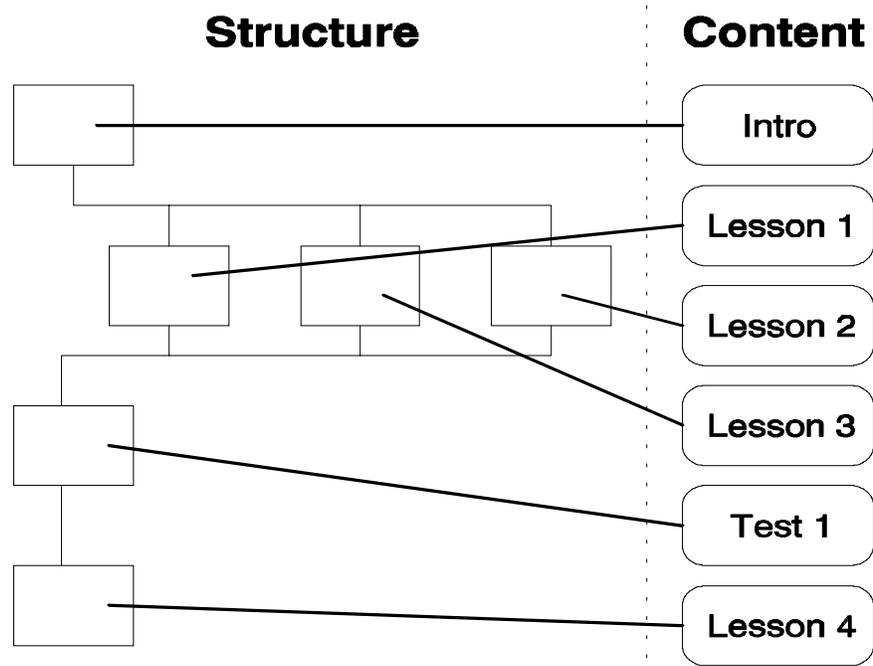
Reasons for guidelines	There are at least two circumstances in which guidelines for moving courses from one environment to another are useful.

- The first assumes a course is complete and is being transferred from a vendor or manufacturer to an airline -- moving from one CMI system to another.

- The second assumes a course is being designed in a tool other than a CMI system -- moving course design into CMI.

From one CMI system to another	Transferring the new course into the existing CMI system manually, requires typing hundreds of lesson names, and duplicating all of the sequencing information. This requires a significant number of man hours. Having a standardized mechanism for describing course content and structure, enables CMI systems to "ingest" a new course with minimal manual effort.
From course design into CMI	There are many tools, other than a CMI system, which may be used to design a new course. One of the most common is a Task Analysis tool. If a course design tool can output a standardized description of a course, the CMI system can pull in the new course from that description. This can save hundreds of man hours of retyping and inputting data.

Why keep structure and behavior?

Defining behavior has been part of the course design. Not keeping behavior is a change of the instructional material, of its philosophy and likely of its efficiency. This is, of course, should be avoided.

One example is that of a course designed to be taken sequentially, with completion assumed when student has been exposed to all the material. If changing CMI system makes all the material accessible freely it is likely that student will skip some chapters, because they think they know them, and maybe loose some instructional material. In this example reliability of the course is impaired by not keeping the intended behavior.

Reverse example is that of a course designed to be accessed freely, with completion assumed when student has passed a set of tests. If changing CMI system makes the progression sequential it is likely that students will spend additional time on the course, compelled to follow sequences that they would normally have skipped to go directly to the test. In this example student acceptance of the course is affected by not keeping the intended behavior
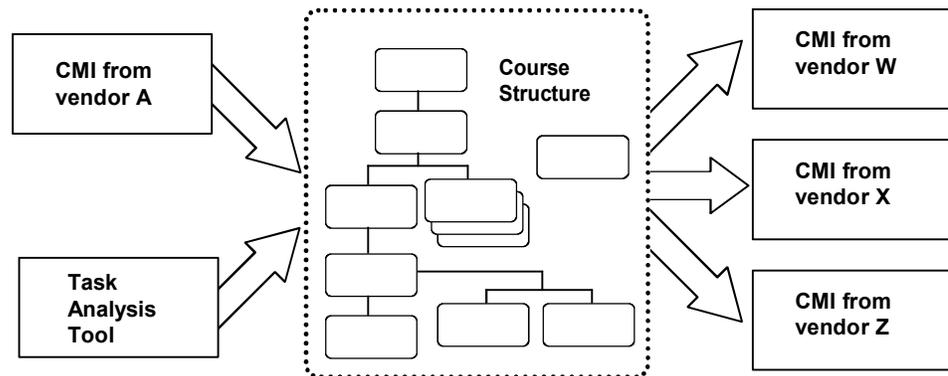
3.2   CMI Communication

CBT system   The CMI should provide a standard approach to lesson initiation to
interoperability   provide a method for interoperability of CMI and CBT systems.   This
would allow a single CMI system to initiate lessons from different CBT
vendors. To accomplish this function the CMI system and CBT system
must communicate passing standard types of data. The following data are
need:

- data needed to be passed into a CBT system from CMI to start
  the CBT lesson

- data needed to be passed from a CBT system to CMI to record
  student performance and perform the next lesson routing or
  assignment

- data needed for evaluation of a lesson such as item response
  data, CBT simulation performance data, and lesson path data

This data flow can be pictured in the following manner:
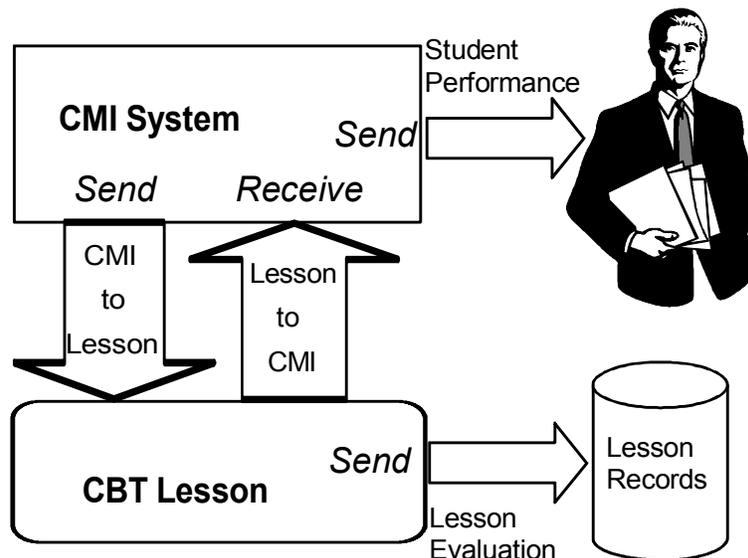
Student performance data may be considered level-one data as described
on page 18. The lesson evaluation data is the level-two data described
there.

3.3   Storing Student Data

Reasons for data   Student performance data includes information that a CBT lesson or test generates for a CMI system, or other analysis tool. Because the file can be used by more than just the CMI system, lesson and student data that is redundant for the CMI system needs to be included.
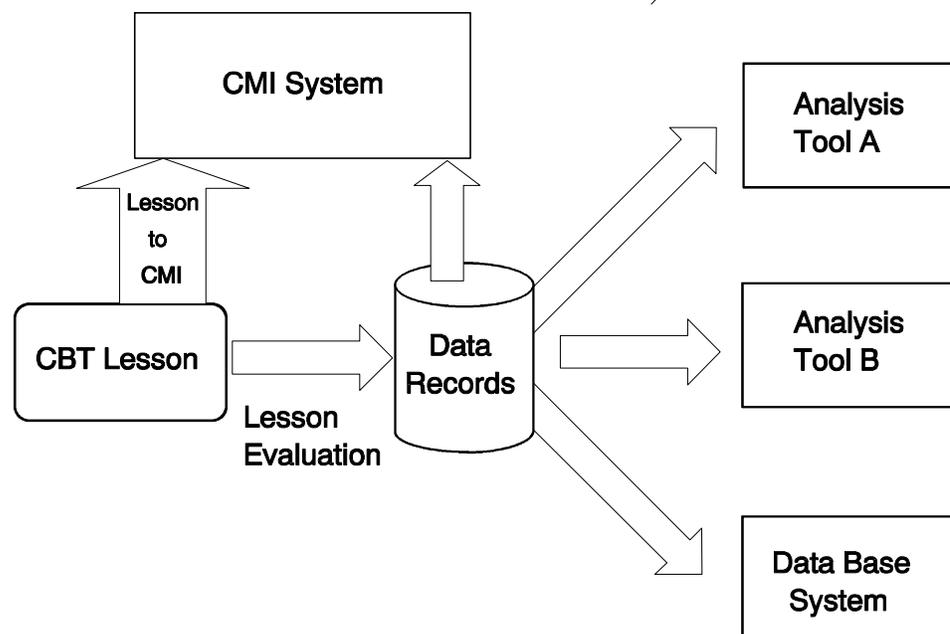
This information can be used for

- Student performance analysis. Data collection of the student's interaction with the lesson. This helps to determine what the student knows, and what he learns. Comparing individual student progress with his peers gives a measurement of individual rate of learning.

- Item analysis. This can indicate how well an element of instruction trains; or how well a test question measures student performance. This enables quality control of the testing and instruction.

- Courseware analysis. The determination of how well the course meets its training objectives. The determination of exactly where, and why the courseware is not performing. This is related to the item analysis.

- Attitude survey. The determination of how well the student likes the courseware. How well the student feels the courseware is working. This aids in measuring customer satisfaction.

- Path optimization. The determination of the best sequencing of lessons and tests for a specific student. The determination of what material may be skipped by a student. The determination of what supplementary material or remediation is required by a student. This is linked to the student performance analysis. This may also support the concept of "progressive sign-off."

Why interoperability

Standardizing the format of the student records permits multiple tools to use the information.

- CMI. Some CMI systems are able to analyze or use more information than is available in the standard Lesson-to-CMI file. By having a standardized format for storing student performance data, these CMI systems can all have access to the additional data,

- CMI. There are a number of reasons for using a single CMI system for lessons made with different authoring systems. By having a standardized format for storing student performance data, lessons from different vendors can be used effectively under the same CMI system.

- Analysis tools. There are a number of different tools that can be used for analysis of student and lesson performance data. If the data for analysis is stored in a standardized format, the different tools can be used to analyze the information.

- Competition. By having standard interchange formats, the market for analysis tools becomes much larger than just a single vendor's customers. Vendors are therefore encouraged to create sophisticated, easy-to-use analysis tools because of the payback of a larger customer base.

4.0   INTEROPERABILITY KEY: THE FILE

Background   The key to interoperability is communication. CMI and CBT systems
must be able to communicate with each other in order to work together.
Communication is essentially a flow of data from one program to
another, or from one system to another.

The three data flows required for interoperability discussed in this
document are:

- CMI $\Leftrightarrow$ CBT

- CMI $\Rightarrow$ CMI

- CMI $\Rightarrow$ Lesson-evaluation

4.1   Data Flow

CMI⇔CBT data flow

The data flow required for CMI and courseware systems can be pictured in the following manner:

```
┌─────────────────────────────────┐
│                                 │
│   CMI System                    │
│                                 │
│     Send         Receive        │
└─────────────────────────────────┘
      │  CMI      ▲  Lesson
      │  to       │  to
      ▼ Lesson    │  CMI
┌─────────────────────────────────┐
│                                 │
│       CBT Lesson                │
│                                 │
└─────────────────────────────────┘
```
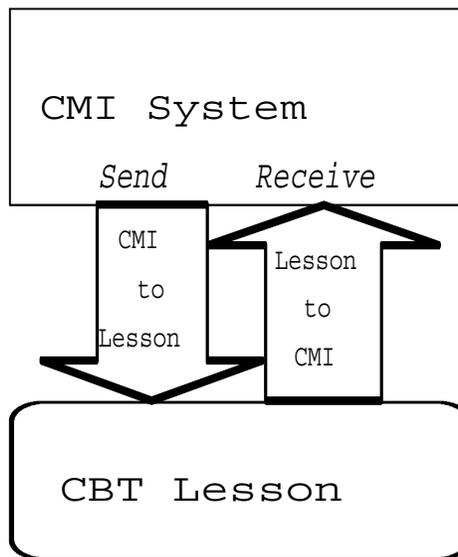
There are two types of data in this flow.

- Data needed to be passed into a CBT system from CMI to start the CBT lesson

- Data needed to be passed from a CBT system to CMI -- data to enable the CMI system to record student performance and perform the next lesson routing or assignment

To enable the data to be used by both the sender and receiver, it needs to have a standard format and content. The format of the data is a file. Details of this file format are discussed in this chapter. Subsequent chapters in this document describe the contents of the different types of files.

CMI to CMI data flow

The following diagram shows the data flow for passing information on course structure and student routing from one CMI system to another.

Course
Structure

```
          ┌─────────────────────┐          ┌─────────────────────┐
          │                     │          │                     │
          │     CMI System      │ ═══════▷ │     CMI System      │
          │                     │          │                     │
          └─────────────────────┘          └─────────────────────┘
```

CMI
to
Lesson

Lesson
to
CMI

CBT Lesson
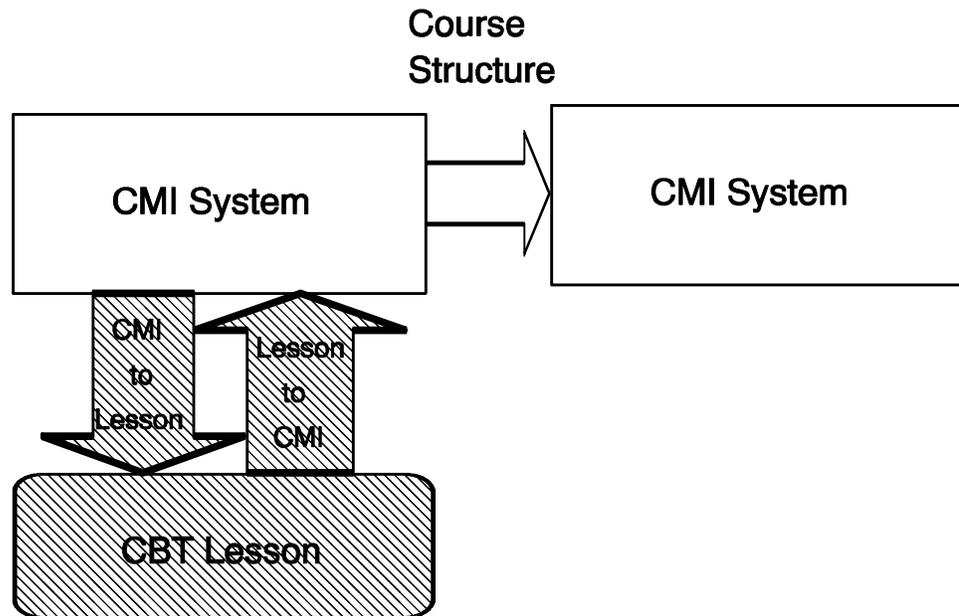
For this information, several files have been developed. The number of
files required depends on the level of detail that is going to be passed
from one system to the next. The files contain ASCII data, but two
different formats are used. The details of the two different file formats
are found in this chapter.

CBT to Lesson-
evaluation data
flow

The following diagram shows data from a lesson being made available
to a number of tools that can be used for item analysis or lesson
evaluation.
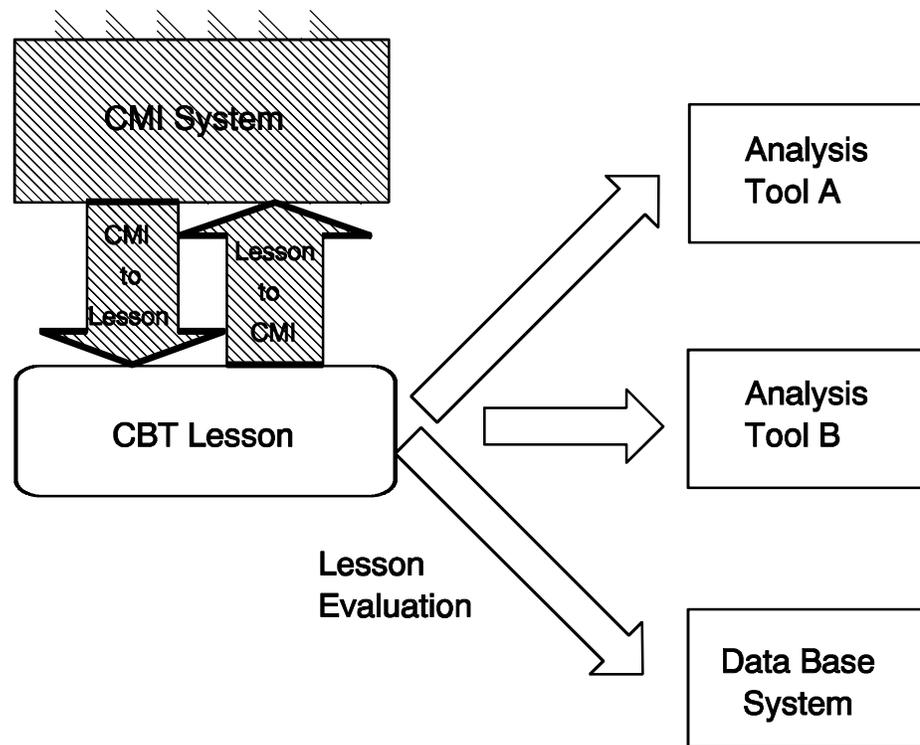
If data is to be accumulated over a number of lesson uses, there are three ways the data can be accumulated:

1) A separate file for every time the lesson is used by a different student.

2) A single file for a lesson, containing records of every student that has used it. Each student as he finishes a lesson has his experiences in the lesson appended.

3) A single file for a student, appending data for each lesson as the student leaves it.

These guidelines use method three. The first approach would result in hundreds of files to manage. The second could result in contention on a network where more than a single student can be using the same lesson.

4.2 File Summary

Files overview   The following is an overview of the files used for CMI interoperability:

- Data is passed as an ASCII file of one of two types
  - Microsoft Windows INI also referred to as a group/keyword file.
  - Comma delimited ASCII also referred to as a table.
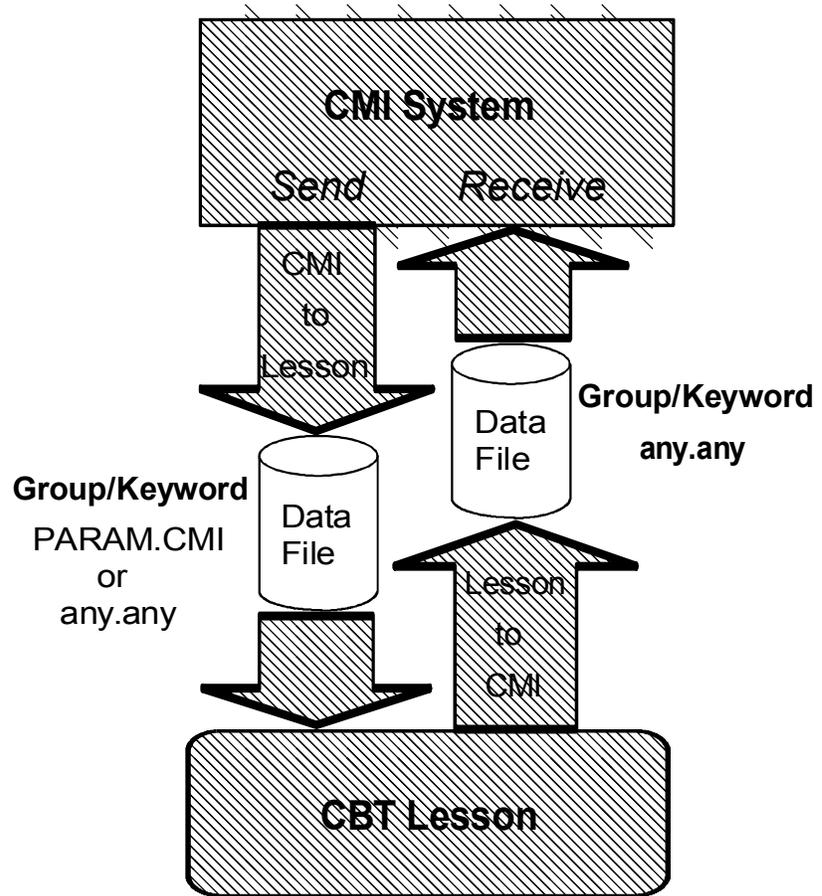
- ♦   CMI/CBT Lesson communication files

CMI to CBT file
• Name: PARAM.CMI or any other name as described in Section 5.0.3.
• Contents: Information needed by a lesson to function optimally.
•   Type: Group/Keyword (Windows INI)
CBT to CMI file
• Name: any.any (name is defined by CMI)
• Contents: Information needed by a CMI system to track student performance and make new decision assignments.
•   Type: Group/Keyword (Windows INI)
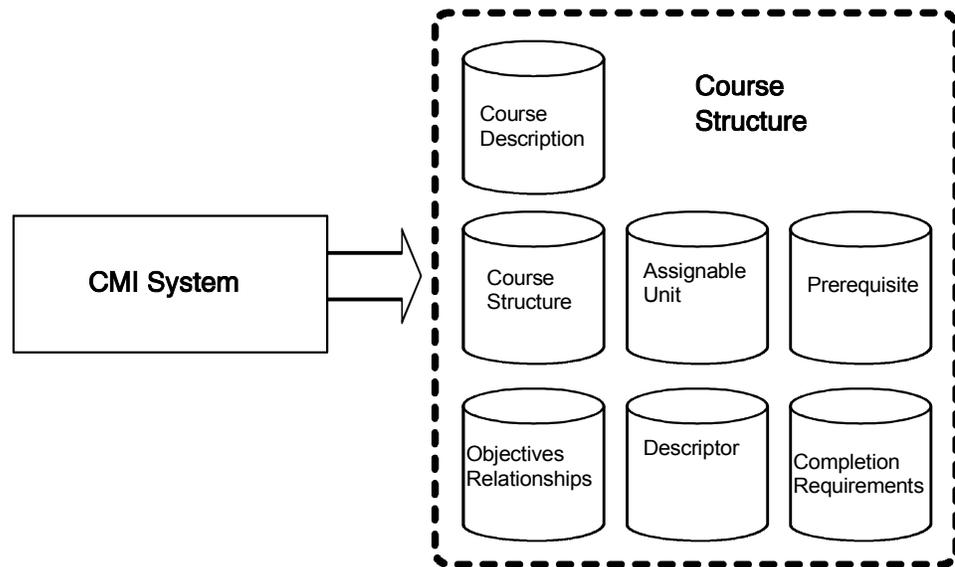
♦   Course description from one CMI system to another

Course Description
•   Name: any.CRS
•   Contents: Basic information about a course, including a textual description.
•   Type: Group/Keyword (Windows INI)
Descriptor
•   Name: any.DES
•   Contents: System generated IDs, titles, and descriptions of each element in the course: Elements include Assignable Units, Blocks, Objectives, and Complex Objectives.
•   Type: Table (Comma Delimited)
Assignable Unit
•   Name: any.AU
•   Contents: Information about each assignable unit, including data needed to launch the unit.
•   Type: Table (Comma Delimited)
Course Structure
•   Name: any.CST
•   Contents: Course structure table (CST).
•   Type: Table (Comma Delimited)
Objectives Relationships
•   Name: any.REL
•   Contents: Shows the relationship (if any) of each objective in the course to 1) other objectives, 2) blocks, and 3) assignable units.
•   Type: Table (Comma Delimited)

Prerequisite •   Name: any.PRE •   Contents: Prerequisite table. Indicate prerequisites for entering each assignable unit. •   Type: Table (Comma Delimited)
Completion Requirements •   Name: any.CMP •   Contents: Completion table. Indicates the requirements for completion of each block or complex objective whose completion cannot be determined by the defaults. •   Type: Table (Comma Delimited)
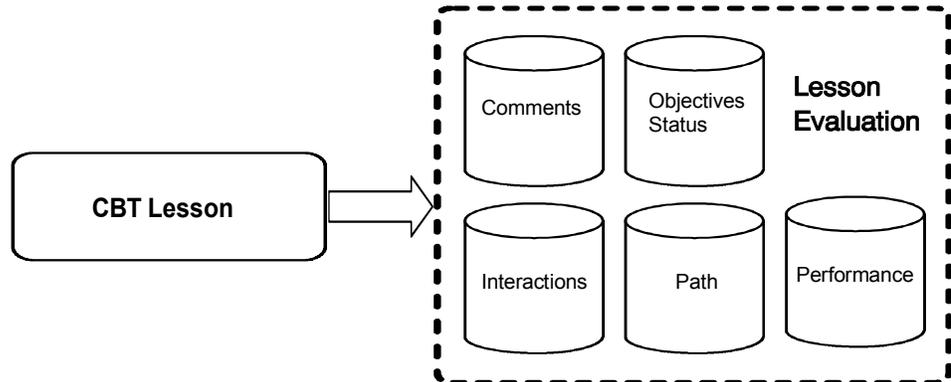
♦   CBT Lesson Evaluation files

Comments
•   Name: any.any (name defined by CMI)
• Contents: Comments made by student while taking a lesson.
•   Type: Table (Comma Delimited)

Objectives Status
•   Name: any.any (name defined by CMI)
• Contents: Repeat of information that is in the CBT to CMI file under the group [Objectives_Status].
•   Type: Table (Comma Delimited)

Interactions
•   Name: any.any (name defined by CMI)
• Contents: Detailed information on each interaction measured as the student takes a lesson.
•   Type: Table (Comma Delimited)

Path
•   Name: any.any (name defined by CMI)
• Contents: A description of the path the student took through the lesson. (What he experienced first, second, third, and so forth.)
•   Type: Table (Comma Delimited)

Performance
•   Name: any.any (name defined by CMI)
• Contents: Information on student's performance in complex simulations. Format yet to be determined.
•   Type: Table (Comma Delimited)

4.3   MS Windows INI Files

This file structure is based on the Microsoft WINDOWS *.INI files.
The INI file contains three types of data -- **group, keyword**, and
comment. The structure of the file and these data types are discussed in
the following sections.

4.3.1   **File Structure**

3 types   Each item in the file is one of three types -- GROUP, KEYWORD, or COMMENT. These are defined in Section 4.3.2.

Naming rules   Group and keyword names are case insensitive. The names may include alphanumeric characters and the underscore character, but may not include spaces.

Group names are left justified and surrounded by brackets. Keywords are left justified and followed by an equals sign (=). Comments are lines whose first character is a semicolon.

Appearance in file	Element name
[group] # keyword=parameter# ; groups and keywords ; may have comments	Group Valid Keyword Comment

The # in this table means CR LF (0x0A 0x0D) (carriage return, line feed). Comments must always be on a separate line from group names and keywords.

Example   This file was created by a Lesson to pass information to a CMI system.

```
[CORE]
LESSON_STATUS = Passed
LESSON_LOCATION = End
SCORE  =  87
TIME   =  00:25:30
; this is the core group of data
; this is the lesson performance data
; for a passed lesson that required a
; time of 25 minutes, 30 seconds and
; a score of 87
```

4.3.2   **Comments**

Definition   Comments are text that is of use to a human viewing a file. They are
                      essentially invisible to a computer processing the data in the file. No
                      action is taken by the processor as a result of comments.

Format   First character in the line is a semicolon. All characters following the
                      semicolon, up to and including the carriage return are considered part of
                      the comment.

Usage rules   Comments may appear anyplace in any order in the file. Comments are
                      only possible in INI files, they are not available in the Comma
                      Delimited table files.

Example

```
; 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 has no impact on the
;  processing of the file.
```

4.3.3	**Groups**

Concept

Groups provide a mechanism for dividing a file into manageable segments that are more easily accessed by data retrieval routines. They also provide a means to organize a file of data into logically related parts. This is helpful for human-processing of a file as well as computer processing.

Definition

Groups are logically related assemblies of data items, generally several lines in length. A group extends from its group identifier to the next group identifier, and may include multiple lines. Although groups may contain keywords, they may not contain other groups.

All carriage returns and symbols between group identifiers may be significant, depending on the definition of the specific group. However, if a group contains keywords, then blank lines and extra carriage returns are ignored.

Format

A group is identified by its name enclosed by square brackets. The left bracket is at the far left margin, or preceded by spaces or tabs. It cannot be preceded by any other characters. It cannot be embedded in a line of information. The name is case insensitive.

Spaces: The name must be an alpha-numeric string with no spaces, inside square brackets. There should be no spaces either preceding or following the name – in other words, no spaces should appear between the brackets.

Usage rules

Groups may appear in any order. Although groups may appear multiple times in the file, only the first occurrence of the group is meaningful.

Group name examples:

    [comments]
    [OBJECTIVES_STATUS]
    [student_demographics]

Document convention

When a group name appears in this document it is identifiable for one of two reasons:

1) It is surrounded by brackets, for example:
   [Objectives_Status]
   [COMMENTS]
   [student_data]

2) It is accompanied by the word "group", for example:
   the Objectives_Status group
   group COMMENTS
   student_data group

4.3.4 **Keywords**

Definition Keywords are names of data items that are limited in size to a single line. This generally limits the data to 60 or 70 characters. The data items associated with a keyword are referred to as *keyword arguments* or *keyword values*.

Format Keywords appear at the left-hand margin followed by an equals sign. Spaces before and after the equals sign are ignored. Keywords are case insensitive.

Keyword extensions Each keyword within a single group must be unique. If keywords are duplicated, only the first one is taken into account. To avoid duplicates, when there are multiple instances of a keyword inside a group, each keyword in the group has an extension. Keyword extensions consist of a period and a simple two digit number, 00 through 99.

Example An examples of a group with multiple instances of a keyword requiring an extension is the [Objectives_Status] group. It has multiple objective ID's and a different status for each objective recorded in the group

```
[Objective_Status]
J_ID.01= AB112
J_Status.01 = Pass
J_ID.02= AB124
J_Status.02 = Pass
J_ID.03= AB196
J_Status.03 = Fail
```

4.3.4 Keywords (cont.)

Argument format   The first non-space character after the "=" identifies the beginning of the data. Capitalization and spaces in the keyword data may or may not be significant, depending on the definition of the data associated with a specific keyword.

Examples:

       Student_ID = JQH2142
       OUTPUT_FILE=C:\STURECS\JQH2142.DTA
       postal_code = 98124-2207

Usage rules   Blank lines between keywords are ignored. Keywords are always members of a group, although there may be groups without keywords in them. An example of a group without a keyword is the [COMMENT] group.

Keyword order   The order in which the keywords appear within any group is irrelevant. In this document, they are ordered alphabetically for the convenience of the reader. However, it is important that the keyword appear within its proper group.

Sometimes the same keywords are used in different groups. For instance, in the same file there is a group called LESSON_DATA with a keyword ID, and a group STUDENT_DATA with a keyword ID. Obviously, these ID's are both different. They can only be different by being members of a different group.

Like group names, keyword names may only appear once. If there are multiple occurrences of the same keyword, only the first one is significant.

Document convention

When a keyword name appears in this document it is identifiable for one of two reasons:

    1) It is followed by an equals sign, for example:
        Score=
        TIME=
        max_time_allowed=

    2) It is accompanied by the word "keyword", for example:
        the score keyword
        the keyword Max_Time_Allowed
        time keyword

4.4   Comma Delimited ASCII

File flexibility Data stored in a comma delimited ASCII file can be imported easily into virtually any off-the-shelf database product or spreadsheet. Many programs use this format to exchange data.

This format is more than just a text file that is saved in ASCII form. Comma delimited format supplies a simple mechanism for separating records and fields, and for distinguishing data types.

Though some systems and applications may support delimiters other than a comma, AICC files of this type require the use of a comma as a separator.

Records and fields The format requires division of the data into records and fields. The record is the data found on a single line. The field is the data that is found between commas (comma delimited) on the line. There is no fixed length for each field, and there is no fixed length for the records in the file.

Critical characters There are certain characters that are important in this file format. The format does not allow carriage returns within a record, double quotes (") within any field, or commas within a number field. In this format, carriage returns, double quotes, and commas are interpreted as record or field delimiters. However, you can use commas and single quote marks (') within text field (fields delimited by double quotes).

Embedded carriage returns One unique addition to the standard comma delimited ASCII file is supported by this standard -- embedded carriage returns. Since each field may be up to 255 characters in length, it may be desirable to indicate where carriage returns are to be placed. Embedded carriage returns are indicated by the characters "<CR>".

When fields with the embedded <cr> are placed by the CMI system into a file in the INI format, the <cr> should be passed as an actual carriage return. This enables a single field to contain several lines of group-fields to contain group-keyword data that otherwise could not be held in a single field.

Tables and files   Notice in the example table below, there are labels for each column. Each entry in a column corresponds to a field. Each row in the table corresponds to a record. In the conversion of this table to a comma-comma-delimited file, the name of each field
is gone. Only the field data itself is in the file.only appears once, in the first record, at the top of the table.

Notice also, that empty field, or blank fields may have to exist in the comma delimited file. In the third record there are two blank fields. The first is an empty number field, and the second is an empty text field. This is true because all records, or rows, in a file must have the same number of fields.

Usage rules   Some files will have different numbers of meaningful data elements in each record. This means that records with fewer members must be padded with blank fields at the end of the record, so that all records have the same number of fields as the record with the most members.

This is necessary for some off-the-shelf database and spreadsheet products to import a comma-delimited file.

Notice the first record in the file. It identifies the name of each field. This identifies the order of the fields in any single file. Two files with exactly the same contents would not have to have the fields in the same order. The first record will always identify the order of all the fields to follow.

In the examples below the first field in the first file is "Lesson_ID". The first field in the second file is "Lesson_File_Name". The order in which the fields appear is different, but the content is the same. The CMI system must be able to interpret the two files, and determine that the information in each is the same.

Example Table

Lesson ID	Title	Type	Max Score	Max_Time_ Allowed	Lesson File Name
777APU-1	Auxiliary Power Unit	Tutorial	38	00:18:00	APU.EXE
777EL-1	Electrical Power, Part 1	Tutorial	41	00:23:00	ELEC1.EXE
777EL-2	Electrical Power, Part 2	Practice			ELEC2.EXE

Comma Delimited File with Same Contents

"Lesson_id","title","type","Max_Score","max_time_allowed","lesson_file_name"
"777APU-1","Auxiliary Power Unit","Tutorial",38,"00:18:00","APU.EXE"
"777EL-1","Electrical Power, Part 1","Tutorial",41,"00:23:00","ELEC1.EXE"
"777EL-2","Electrical Power, Part 2","Practice",,"","ELEC2.EXE"

Second Comma Delimited File with Same Contents

"lesson_file_name","Lesson_id","title","type","Max_Score","max_time_allowed"
"APU.EXE","777APU-1","Auxiliary Power Unit","Tutorial",38,"00:18:00"
"ELEC1.EXE","777EL-1","Electrical Power, Part 1","Tutorial",41,"00:23:00"
"ELEC2.EXE","777EL-2","Electrical Power, Part 2","Practice",,""

4.5   File Limits

Differences between INI and Table files

The following table summarizes some of the differences between the Keyword/Group file (sometimes called an INI file) and the Comma-Delimited Table file. It includes limits associated with the contents of each.

	Keyword/Group	**Table**
Smallest unit of data	keyword	field
Size limit of the value for the smallest unit (unless otherwise specified in the field/keyword description)	255 characters[2] unless otherwise specified	255 characters unless otherwise specified
Frequency that the unit's name appears in file	keyword name appears every time a keyword is used	field name appears only once, at beginning of file
Number of small data units in a line of data	one keyword per line	many fields per line
Largest unit of data	group	record
Number of lines allowed for large unit of data	each group can have many lines	only one line per record
Size limit for large unit of data.	limited only by the number and size of the keywords[3]	limited only by the number and size of the fields
Maximum size of one line of data	255 characters plus length of keyword and equals sign	unlimited (many times 255)

---

[2] In this document one character is assumed to require one byte.

[3] There are three exceptions: The [Comments] and [Core_Vendor] groups in the CBT/CMI communications files and the [Course_Description] group in the *.CRS course interchange file. These groups have a limit of 4096 characters.

5.0   CMI/LESSON COMMUNICATION

Two-way communication

CMI and Lesson communication is two way. The CMI system sends information to the lesson when it begins. The lesson sends information to the CMI system when the lesson ends.

The information is sent in a file -- two files actually. The first file is created by the CMI system, and the second is created by the lesson.

Chapter contents

Section	Subject
5.0.1	Launching CBT Lessons
5.0.2	Data Passing
5.0.3	Summary of launching a lesson & passing data to it
5.1	Contents of CMI to CBT Lesson files
5.1.1	[Core]
5.1.2	[Core_Lesson]
5.1.3	[Core_Vendor]
5.1.4	[Comments]
5.1.5	[Evaluation]
5.1.6	[Objectives_Status]
5.1.7	[Student_Data]
5.1.8	[Student_Demographics]
5.1.9	[Student_Preferences]
5.2	Contents of CBT Lesson to CMI files
5.2.1	[Core]
5.2.2	[Core_Lesson]
5.2.3	[Comments]
5.2.4	[Objectives_Status]
5.2.5	[Student_Data]
5.2.6	[Student_Preferences]
5.3	Error and Default Conditions
5.3.1	File Creation, File Read, and File Write Errors
5.3.2	Data and File Format Errors
5.3.3	CBT and CMI System Mismatch Errors

5.0.1   **Launching CBT Lessons**

The AICC CMI committee defines possible methodologies for the
initiation of a CBT system from a non-native CMI system. Note these
are possible methods but others do exist. CMI developers are
encouraged to use the method that is most efficient for their systems.

**General
Requirements**

CMI system developers must establish a mechanism to launch multiple
CBT vendor runtime systems. This mechanism will be different in the
DOS and Windows environment; however, two requirements will be the
same. They are:

- Students must have a single point of entry into and exit from the
  CMI ➔ CBT ➔ CMI run sequence. CMI developers must insure
  that the CBT application is well-behaved, meaning that CBT results
  get reported back to CMI system for current assignment (i.e. CMI
  initiates CBT, CBT runs, CBT returns to CMI with results of
  lesson).

- The CMI system may allow multiple lessons to run concurrently.
  However, once the lesson is begun, it is mandatory that the lesson
  returns its results to the CMI system when it finishes.

Example: DOS Environment Launch

This is an example at the conceptual level of how CMI can launch a CBT application in a DOS environment. This is not an official AICC recommendation or guideline. It is here to explain some of the challenges of interoperability.

The initiation of a non-native CBT runtime from a CMI application can be done in a number of ways. One method requires the use of a very small DOS shell executable to be resident during the execution of the CBT to ensure return of control to the CMI application and necessary cleanup of CBT-related memory resident code. Another task of the shell executable would be to clean up after the exiting CMI application by unloading drivers. The small DOS shell could also initiate the CBT application in one of several ways:

- Execute .bat file identified to run the desired lesson
- Load the necessary drivers and initiate the CBT application through command-line parameters.
- Perform a combination of the above where the shell has a skeleton .bat file and modifies it on the fly with lesson specific command line parameters

The shell executable performs these types of activities prior to startup of the CBT system:

- Unload all CMI system device drivers, clear all I/O addresses and interrupts, clear environment variables, clear out extended memory
- Load CBT device drivers, set system environment variables, and all other functions needed to ready the system for CBT execution

The shell executable would perform these types of activities after return from the CBT system and prior to startup of the CMI system:

- Unload all CBT system device drivers, clear all I/O addresses and interrupts, clear environment variables, clear out extended memory
- Initiate the CMI system including loading device drivers and all other functions needed to ready the system for CMI execution

This is a diagram of flow of control.
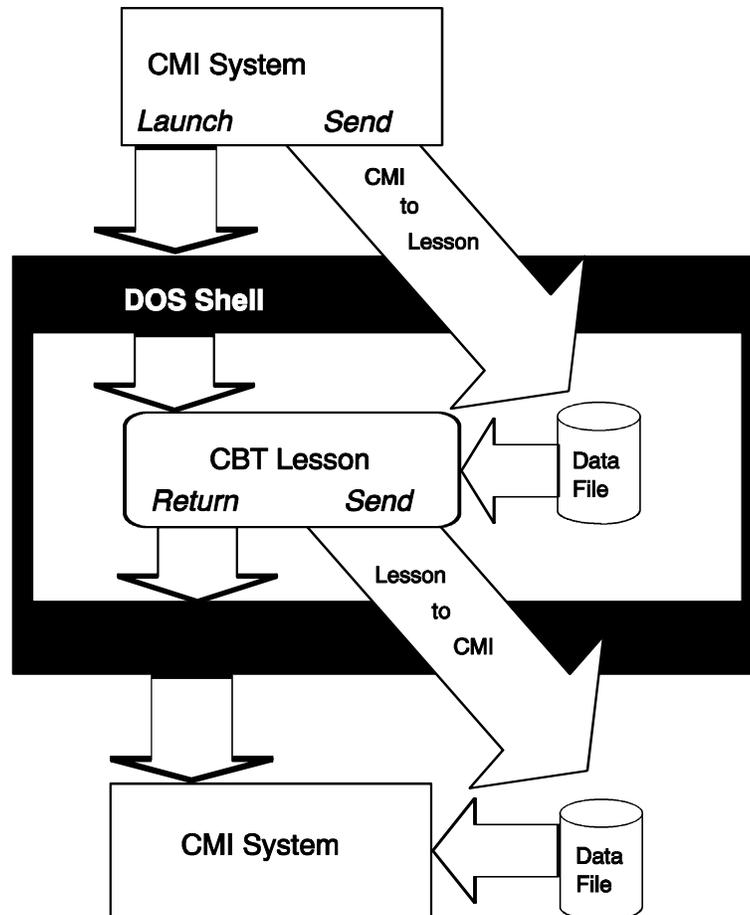
Example: Windows Environment Launch	This is an example at the conceptual level of how CMI can launch a CBT application in a Windows environment. This is not an official AICC recommendation or guideline. It is here to explain some of the challenges of interoperability.
	One method for launching a non-native Windows CBT application from a Windows CMI application is similar to that used in the DOS environment. The use of a CMI shell would be used to replace the Windows Program Manager. This would be done by changing the shell command in the SYSTEM.ini file from PROGMAN.EXE to the CMI shell executable developed. This allows the CMI shell to control the sequence of programs and insure that CBT will be a well-behaved application and return to CMI to update student records.

5.0.2   **Data Passing**

Data files are the means to
   a)   Initiate non-native CBT lessons and
   b)   Report results back to the calling CMI system work.

To do this there must be a common understanding of when, where, and how these data files are created, read, and deleted.

The data flow, with the inclusion of the files, looks like this.
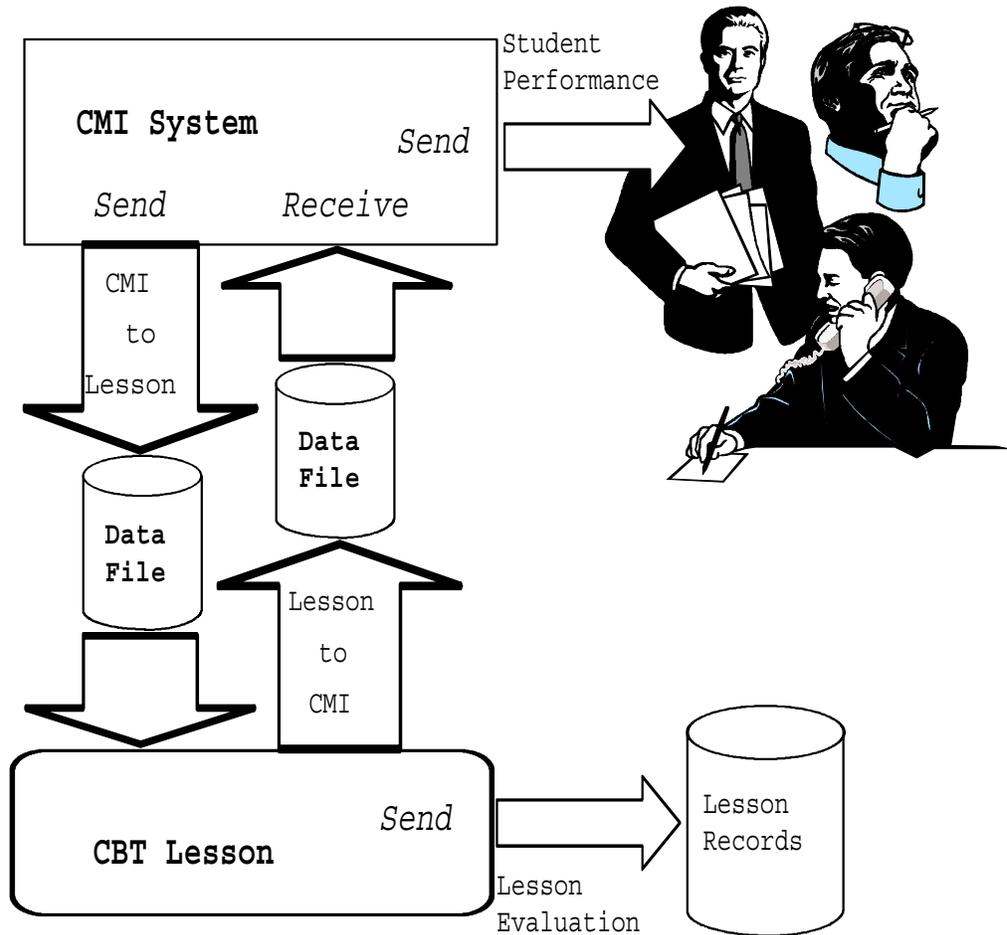
5.0.3   **Summary**

Essentially this is how the interoperability works:

CMI creates file, launches CBT

1. The CMI system creates a file containing the data necessary to start-up a CBT lesson. The file is created just prior to the initiation of the CBT system.

    Due to the temporary nature of the file, it normally resides on the local station, either the hard disk or floppy (if there is no local hard disk). However, if there is no requirement for local storage, the CMI system can decide where the file should reside.

CMI to CBT file name

The name of the CMI-to-CBT lesson data file must be made available to the CBT application at CBT system initiation. Three methods may be used to accomplish this.

- For some DOS applications, a DOS environment variable, PARAM$CMI, can be set by the CMI runtime and queried by the CBT runtime. PARAM$CMI contains a complete file name, including the drive and path, for the CMI-toCBT data file.

- An additional command line parameter containing the CMI to CBT lesson data file name can be processed by the CBT runtime.
- A standard file, PARAM.CMI, can be used which contains the CMI-to-CBT lesson data and which is placed by the CMI system in the directory specified in the Microsoft Windows "windir" environment variable. Examples of this directory are "c:\windows" for Windows95/98 and "c:\winnt" for Windows NT.

CMI developers must support all three recommended methods for CMI to CBT lesson data filename passing. CBT system developers must use one of the recommended methods for receiving the CMI to CBT lesson data file name.

In addition, CMI systems must also support both a vendor CBT system handling courseware using the CMI-to-lesson data file and courseware developed before the guidelines were instituted (e.g. courseware not expecting the CMI-to-lesson data and not generating the lesson-to-CMI data). Therefore, the CMI system generating the command line must provide for the following two situations:

- The case where the CBT runtime expects an additional command line parameter containing the CMI-to-lesson file name

- The case where older CBT courseware is to be run that cannot process the extra command line parameter

This could be done in several ways and it is up to CMI developers to determine the best methodology for their system.

CBT reads CMI-created file   2. Once the CBT system is initiated, it reads the data file created by the calling CMI system and then immediately deletes it.

Some lessons may not need this input file simply because student information is not necessary for the lesson. If the data is needed, the CBT system must use one of the recommended methods for receiving the CMI-to-CBT data filename.

CBT creates file   3. The CBT system must create a file containing data to be passed back to CMI so that the CMI system can update its student performance data and make the next assignment (perform routing activity).

The CMI system passes in the file name for the lesson-to-CMI data file as part of the CMI-to-lesson core data.

The creation of the CBT lesson to CMI data file with an attempt status of NOT ATTEMPTED must be one of the first activities that the CBT system performs. This is necessary so that the CMI system will know the correct status of the CBT lesson and will be able to recognize a resume/bookmark status versus a completion.

In order for a CBT systems resume/bookmark capability to work properly, the calling CMI system must be able to distinguish between possible reasons for leaving the CBT lesson, such as the following.

- A resume/bookmark condition.

- Lesson completion.

- Student logout.

Some CMI systems may wish to distinguish other reasons for leaving a lesson as well.

The existence of the CBT-to-CMI data file with a completion indication of NOT ATTEMPTED (i.e. ATTEMPT STATUS = NOT ATTEMPTED) signals to the CMI system that the lesson is to be resumed and no new assignment (routing activity) is to be generated.

Some of the data stored in the CBT-to-CMI data file will need to be stored temporarily in the CMI system and then fed back into the lesson. This data includes core keywords such as LESSON_LOCATION that the CMI system sends back to the lesson in a resume situation.

This means that the CMI system will allocate a certain amount of temporary storage for each student when a lesson resume/ restart is encountered to store data that will need to be feed back into the lesson.

The data that is to be stored over a restart/resume condition is shown in the following table.

CBT Lesson ➔ CMI

Group	Keywords
CORE	Lesson_Location Lesson_Status Score Time
CORE_LESSON	*lesson dependent*

The stored data is to be fed back into the lesson in the following groups and keywords.

CMI ➔ CBT Lesson

Group	Keywords
CORE	Lesson_Location Lesson_Status Score Time
CORE_LESSON	*lesson dependent*
CORE_VENDOR	*vendor dependent*

The temporary restart/resume data stored in the CMI system can be deleted by the CMI system upon lesson completion or change of assignment.

In order to report back student performance on the lesson, the CBT system must set the core data fields in the CBT-to-CMI data file at lesson completion. These data items are defined in Chapter 5.

If there is no CBT-to-CMI data file, the CMI system assumes the following:

- Current lesson is complete

- No time or score data exists so the CMI system should use its normal defaults (score="" and time=0)

- The next lesson (routing activity) should be determined and assigned

CBT finishes, CMI reads CBT-created file   4.   The CMI system reads the CBT-to-CMI data file and using the information passed from the CBT lesson updates applicable student data kept by the CMI system and determines the next student assignment or routing activity.

In the case of a resume/restart of an existing lesson, the CMI will temporarily store the core group data fields identified in item 3 above. When the lesson is to be re-initiated the CMI sends those applicable data items with the temporarily stored data. This means the CMI system will be expected to allocate a certain amount of temporary data for each student over a lesson restart/resume. AICC recommends an allocation of 512 bytes per student needed only if the lesson is interrupted.

It is the responsibility of the CMI system to delete the CBT-to-CMI data file either immediately after determining the student's next assignment/routing activity or in such a manner as to insure that the disk space is managed properly and that there isn't leftover data confusing the lesson.

5.1   CMI to CBT Lesson

Description   This is information that a typical lesson obtains from a CMI system to enable it to perform the functions expected of it. In this documentation, core items are listed first, followed by the optional group names alphabetically. (In the file, group names may be in any order.) After each group name are the keywords (if any) which are appropriate for that group. (In the file, keywords may appear in any order inside their group, unless their description defines a required sequence.)

Required item   A required item is one which must always be provided by the CMI system to be AICC compliant. Required items are those which a lesson may always depend upon being available. The lesson may or may not use the core items, but they are available if needed. Most core items are required. The exception is the Lesson_Mode keyword in [Core] which is optional.

Optional item   Optional items are group and keyword data which may be needed by a lesson to perform optimally. However, the lesson must be constructed such that there is a default to be used if these optional items are not provided by the CMI system.

Group Names and Keywords	Function of Group
[Core]	Information required to be
Student_ID	furnished by all CMI systems.
Student_Name	What all lessons may depend upon
Output_File	at start up, from any AICC
Credit	compliant CMI system.
Lesson_Location	
Lesson_Status	
Path	
Score	
Time	
Lesson_Mode	Optional item in this group.

[Core_Lesson]
    Data is undefined and may be
    unique to each lesson.
	Information held by the CMI
	system for the lesson since last
	student attempt.

[Core_Vendor]
    Data is undefined and may be
    unique to each vendor.
	Required information for some
	lessons. Must be furnished by
	CMI system.

[Comments] no key words <delimited>	E-Mail type information that an instructor or administrator wants to send to a student.

[Evaluation]
    Course_ID
    Comments_File
    Interactions_File
    Objectives_Status_File
    Path_File
    Performance_File
	File names and locations where the
	lesson should store the lesson
	evaluation information.

[Objectives_Status]
    J_ID.1
    J_Score.1
    J_Status.1
	Information on each objective in
	an assignable unit.

[Student_Data]
    Attempt_Number
    Mastery_Score
    Max_Time_Allowed
    Time_Limit_Action
    Lesson_Status.1
    Score.1
	Information on student
	performance expectations.

[Student_Demographics]
   City
   Class
   Company
   Country
   Experience
   Familiar_Name
   Instructor_Name
   Job_Title
   Native_Language
   State
   Street_Address
   Telephone
   Years_Experience
	Personal information on student. Characteristics relating to student before course entry.

[Student_Preferences]
   Audio
   Language
   Lesson_Type
   Speed
   Text
   Text_Color
   Text_Location
   Text_Size
   Video
   Window.1
	Student selected options collected in previous lessons, or previous instances of this lesson.

5.1.1   **[Core]**

Definition   This is a required group that must contain the following keywords.
                            Student_ID
                            Student_Name
                            Output_File
                            Lesson_Location
                            Credit
                            Lesson_Status
                            Path
                            Score
                            Time

                            The group may optionally contain the following keyword.
                            Lesson_Mode

Example

```
[core]
student_ID =  CAM-1942
student_name = McArthur, Christopher A. Jr.
OUTPUT_FILE= C:\CMI\STURECS\cam-1942.dta
lesson_location=0
credit=credit
lesson_mode=Sequential
lesson_status=complete
Path=C:\CMI\STUDENT\CAM-1942\
time=00:23:15
score=93
```

5.1.1 [Core] keywords

Student_ID=	**Definition**	Unique alpha-numeric code/identifier that refers to a single user of the CMI system.
	Format	Up to 255 alpha-numeric characters with no spaces. Additional legal characters in a student_id are the dash (or hyphen) and the underscore. Periods are illegal characters. Case insensitive.
	Examples	student_id=Jack_Hyde1 student_ID = JQH-1942 STUDENT_ID= jack1991-3

Student_Name=	**Definition**	Normally, the official name used for the student on the course roster. A complete name, not just a first name. There is a separate keyword in the group **Student_Demographics** for the student's nickname -- keyword **Familiar_Name**.
	Format	Last name, first name and middle initial. Last name and first name are separated by a comma. Spaces in the name must be honored.
	Examples	Student_name=Whiplash, William R. STUDENT_NAME= Grey, Jane S. student_name = McArthur, Christopher A. Jr.

5.1.1 [Core] keywords (cont.)

Output_File=	**Definition**	Name (including path) of the file which the lesson must construct if it is to pass information back to the CMI system
	Examples	output_file =d:\students\jqh\dtafile.txt OUTPUT_FILE= C:\CMI\STURECS\jqh-1942.dta

Lesson_Location=	**Definition**	This corresponds to the lesson exit point passed to the CMI system the last time the student experienced the lesson. This keyword provides one mechanism to let the student return to a lesson at the same place he left it earlier. In other words, this keyword can identify the student's exit point and that exit point can be used by the lesson as an entry point the next time the student runs the lesson.
	Data format	Implementation dependent. The CMI system simply holds this data and then returns it to the lesson when the student is re-entering it. Whatever the lesson passes back to the CMI system is returned. The format matches whatever the lesson expects -- the format is created by the lesson. The first time a student enters the lesson, or if there is no preferred starting point Lesson_Location may be an empty string or blank.

5.1.1 [Core] keywords (cont.)

Credit=	**Definition**	Indicates whether the student is being credited by the CMI system for his performance (pass/fail and score) in this lesson.

There are two possible arguments for this keyword, Credit or No-credit.

- **Credit**. This means that the student is taking the lesson for credit. The CMI system is telling the lesson that if the lesson sends data to the CMI system, the CMI system will credit it to the student.
- **No-credit**. This means that the student is taking the lesson for no credit. His current credit, if any (for instance a score of 80 and status of passed) will not be changed by his performance in this lesson. The CMI system is telling the lesson that if the lesson sends data to the CMI system it will not change the student's accreditation.

Default   If an unrecognized or unanticipated CREDIT argument is received, then Credit is assumed by the lesson.

Format   One of two words may be the argument for this keyword.
        Credit
        No-credit
Only the first character is significant. Capitalization is not significant.

Examples   Credit=c
Credit = no-credit
credit=N

5.1.1 [Core] keywords (cont.)

Lesson_Mode=	**Definition**	Identifies the lesson behavior desired after the launch.
	Lesson behavior	Many lessons have a single "behavior." Some lessons 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 decision. Designers may enable lessons to behave in a virtually unlimited number of ways. This standard supports the communication of three parameters that may result in different lesson behaviors.

Browse. The student wants to preview the materials, but not necessarily challenge the courseware for a grade.

Normal. This indicates the courseware 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..

> Lesson behaviors are designer defined. Behaviors not on this list may be used in lessons that are compliant with these guidelines. As additional behavior modes are defined by lesson designers it is recommended they be added to this list by submitting a request to the chairman of the AICC CMI subcommittee.

5.1.1 [Core] keywords (cont.)

Default	If an unrecognized or unanticipated LESSON_MODE is received, then the mode the lesson designer considers normal is assumed by the lesson. Whatever strategy or behavior preferred by the designer is the default.
Format	The vocabulary of legal values is: Browse Normal Review Only the first letter is significant. Capitalization is ignored.
Example 1	Lesson_Mode = B ; Student wants to browse the material.
Example 2	lesson_mode=Normal
Example 3	LESSON_MODE= Review
Example 4	; In the following example (illegal argument) the lesson ; should assume browse behaviors because the first ; letter is a "b". lesson_mode = begin

5.1.1 [Core] keywords (cont.)

Lesson_Status=   **Definition**   This is the current student status as determined by the CMI system, and sent to the lesson when it is launched.

Six statuses are possible. If it is the student's first attempt at the lesson the status may be not attempted, passed, or failed and shall include the "a" flag.

- **Passed** (or p or pass) Necessary number of objectives in the lesson were mastered, or the necessary score was achieved. Student is considered to have completed the lesson and passed.

- **Completed** (or c). The lesson may or may not be passed, but all the elements in the lesson were experienced by the student. The student is considered to have completed the lesson.

  For instance, passing may depend on a certain score known to the CMI system. The lesson knows the raw score but not whether that raw score was high enough to pass.

- **Failed** (or f or fail). The lesson was not passed. All the lesson elements may or may not have been completed by the student.

- **Incomplete** (or i). The lesson was begun but not finished.

- **Browsed** (or b). The student launched the lesson with a CMI mode of Browse on the initial attempt.

5.1.1 [Core] keywords

Lesson_Status=
(cont.)

- **Not attempted** (or n or na). Incomplete implies that the student made an attempt to perform the lesson, but for some reason was unable to finish it. Not attempted means that the student did not even begin the lesson.

  Maybe he just read the table of contents, or lesson abstract and decided he was not ready. Any algorithm within the lesson may be used to determine when the lesson moves from NOT ATTEMPTED to INCOMPLETE.

Determine status

The way in which the status decision is made, is defined as follows:
- Normally the lesson determines its own status and passes it to the CMI.
- If there is a mastery score in the assignable unit file (xxxxxxx.AU), the CMI can change status to either Pass or Fail depending on the student's score compared to the mastery score.
- If there is a completion requirements file (xxxxxx.CMP), the CMI shall change the status of a lesson based on the defined completion requirements.
- If a conflict exists, completion requirements has precedence over mastery score.
- If there is no mastery score or completion requirements file, the CMI cannot override lesson-determined status.
- If credit=n, there is no change in status, with one exception. If the lesson_mode is "browse", the status may change to "browsed", even with credit=n.

Flag

There are two possible flags: **Ab initio** and **Resume**. They required in the circumstances described below.

Ab initio (A or a). This indicates it is the first time the student is entering the lesson. Because the student may have passed all the objectives in a lesson by completing a pre-test, the status of Not Attempted is not a reliable indicator. That is, a lesson may be Passed without the student having ever seen it. Consequently, there is a need for this flag to indicate when the student is entering the lesson for the first time.

Resume (R or r). This indicates that the student was in the lesson earlier, and when he exited the lesson, the lesson_status had a suspend flag. The student is resuming a suspended lesson.

No Flag. The absence of a flag indicates that it is a normal re-entry.

Argument format	A word, character or phrase optionally followed by a coma and an additional character or word.. Only the first character is significant in each word.

5.1.1 [Core] keywords

Example 1   Lesson_Status = failed
; Student failed the lesson the last time he was in it.

Example 2   Lesson_Status = N,A
; Student is entering the lesson for the first time.
; The "A" flag along with the Not attempted status is
; required.

Example 3   lesson_status = p
; Student passed the lesson when he was in it previously
; Absence of the A flag indicates this is not his first time
; in the lesson. Absence of the R flag indicates he
; exited the lesson normally when he passed it (i.e. he did
; not generate a Suspend flag).

Example 4   lesson_status=i,r
; Student did not finish lesson. When he left, a suspend
; flag was generated. The Resume flag is therefore
; required.

Example 5   lesson_status = p,a
; Student has demonstrated mastery of the contents
; of the lesson – probably by passing a pre-test. This
; is his first time into the lesson, as indicated by the
; Ab initio flag.

Example 6   lesson_status=i
; Student did not finish lesson. When he left, a logout or
; some other flag may have been generated. No suspend
; flag was created.

5.1.1 [Core] keywords

Path=

Definition   The Path keyword indicates to the CBT lesson where additional files specific to student progress and lesson status may be written. The directory indicated by the Path keyword is specific to an individual student.

Usage rules   A separate directory (or storage management for all the files in that directory) must be maintained for each student.

Examples   Path=C:\CMI\STUDENTS\CAM-194\

5.1.1 [Core] keywords

Score=	**Definition**	Indication of the performance of the student during his last session in the assignable unit.
		This score 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.
	Optional values	The score may be followed by two values: a maximum and minimum.
		Maximum This is the largest score the student could have achieved with the interactions that he experienced.
		Minimum This is the smallest score that the student could have achieved with the interactions he experienced.
	Advantages of 3 scores	There are several advantages to providing a Score, Minimum, and Maximun:
		1) It removes all ambiguity about the scoring range, allowing a CMI to exploit the data in any way desired.
		2) It minimizes the complexity of managing and comparing summary data between learning or testing modules, and moves it from the application to the server. For example, it would allow comparing the score of adaptive tests that result in different raw score maxima because they vary in the number of questions asked.
		3) It allows a test to return a raw score, with all the precision needed; for example 456,800,200.
		4) It can degrade gracefully to handle 2-item score values where the min_score is assumed to be 0.

5) It can degrade gracefully to handle 1-item score values where the max_score is assumed to be 100 and the min_score is assumed to be 0.

Format   Decimal number or blank for all three values. Values separated by commas. The order is significant: Score, Maximum, Minimum.

Usage rules   When the file is created for a student's first attempt, SCORE ="". That is, the score is a blank "Score=". For additional attempts the score reflects what was recorded on the student's last previous attempt.

Examples   SCORE= 79
; Probably a percentage result.
SCORE= .79
; Certainly a percentage result.
Score = 8, 10, 0
; Raw score of 8 with a maximum of 10.
; Percentage score would be 80%.
score=1.3, 2
; Raw score of 1.3 with a maximum of 2.
score= 4, , -1
; The maximum is omitted here.. Raw score of 4 with
; a minimum of –1.
Score=
; Either the student's first entry or he did not attempt
; any scored interactions in his previous use of the lesson.

5.1.1 [Core] keywords

Time=

Definition	Total time of all uses of an assignable unit by a single student. Accumulated time of all the student sessions in the lesson.
Data format	HHHH:MM:SS.S Integer number representing hours, followed by a colon, an integer from 00 to 59 representing minutes, followed by a colon and a decimal or integer from 00 to 59.99 representing seconds.
Usage rules	Three numbers, separated by colons, are always required, even if only seconds or minutes are represented.
Examples	TIME= 00:29:00 ; Student spent 29 minutes in lesson. time=1:27:00 ; Student spent 1 hour 27 minutes in lesson. time= 00:01:27 ; Student spent 1 minute 27 seconds in lesson. time= 00:04:00 ; The student spent 4 minutes in the lesson during the ; current attempt.

5.1.2   **[Core_Lesson]**

Definition   Unique information applicable to a launching lesson. Normally this is
                        the group used by the lesson for restart information. This is normally
                        data that is created by the lesson and stored by the CMI system to pass
                        back to the lesson for the next time the lesson is run.

                        The CMI system must set aside space for this group for each lesson for
                        each student. It stores this data and returns it to the lesson when it is run
                        again. The CMI system shall retain this data as long as the student is in
                        the course.

Format   Lesson unique. The only limitations on this data are:

                        1.   Data must be transferred in ASCII format. The lesson may then
                             convert it to any form that it requires.

                        2.   To avoid over-burdening the CMI system this group should be
                             limited to 4096 bytes of data. If more space is needed for this data,
                             then this group can name a file containing complete data for the
                             lesson.

5.1.3   **[Core_Vendor]**

Definition   Unique information required by the lesson's design. Without this
                          information, a lesson may not execute. The contents of this group is
                          defined in the Assignable Unit structure file: xxxxxxxx.AU

Format   The format and content of this information is described in the
                          Assignable Unit structure file as the Core Vendor field on page 160.
                          The CMI system needs to store this information and pass it to each
                          lesson at the time of launch.

Core_Lesson vs.   There are several significant differences between core_lesson and
Core_Vendor data   core_vendor data.

- Core_Vendor data when available is always passed to a lesson.
  Core_Lesson data is only available for a restart.

- Core_Vendor will always be the same for a given lesson.
  Core_Lesson may be different depending on any restart factors that
  are stored in the group.

- Core_Vendor data is limited to 4096 characters. If more space is
  needed for this data, then this group can name a file containing
  complete data for the lesson.

5.1.4   **[Comments]**

Definition   Optional group. Instructor comments directed at the student that the
                          lesson may present to the student when appropriate.

Format   Any number of lines with any alphanumeric and special characters. The
                          carriage returns are not used at the end of every line in the file. Word-
                          wrap is allowed. Carriage returns, when in a string should be honored
                          by the lesson (application).

                          For example:

```
                                                                               ...abc
            def
```

                          when displayed to the student becomes

```
            ...abcdef
```

                          while:

```
            ...abc<cr>
            def
```

                          when displayed to the student becomes

```
            ...abc
            def
```

Usage rules   Any number of comments may be included and comments may be
                          nested. Each comment is tagged -- that is, it is preceded and followed
                          by a special series of characters. The first comment is preceded by a
                          less-than sign, the number one, and a greater than sign -- <1>. The end
                          of the comment is indicated by <e.1>. Each comment is numbered
                          sequentially with Arabic numerals in its tag. The tags do not appear on
                          screen to the student.

5.1.4 [Comments] keywords (cont.)

Size   4096 bytes.

Example 1   [Comments]
<1>The trainer session following this lesson will be at 13:15 Thursday instead of 9:00.<e.1><2>If you have passed practice session three in the simulator, please skip the Practice section in this lesson.<E.2>

Example 2   [Comments]
<1> On the pretest, you missed questions which indicate you should concentrate on the following topics:
AC power generation
DC power sources
AC power delivery
AC power usage <e.1>

5.1.5   **[Evaluation]**

Definition   Several files may be created to hold student-performance/lesson-
                        evaluation information.   This group enables the lesson to know if data
                        should be collected, and where to put the collected data.

                        The "file" keywords indicate a path and file name for the data. If the
                        keyword is sent to the lesson with a null (blank) argument, it tells the
                        lesson not to collect that data. **A blank argument turns off data
                        collection.** Not having any keyword causes the lesson to revert to its
                        default settings for data collection.

Keywords   **Course_ID**
                                Comments_file
                                Interactions_file
                                Objectives_status_file
                                Path_file
                                Performance_file

5.1.5 [Evaluation] keywords (cont.)

Course_ID=	**Definition**	Alpha numeric sequence that uniquely identifies a course.
		The course identifier is required in all of the output file formats. Since any given lesson may be used in several courses, it is necessary for the CMI system to tell the lesson (assignable unit) which course it is in. This ID is the first field in all of the evaluation output files.
	Argument format	Alphanumeric characters. All characters, beginning with the first printable character after the equals sign are significant.
	Examples	Course_ID = A320-FT-002
		Course_ID = FT747-302-4
		Course_ID = 767-224-4.MT
Comments_file=	**Definition**	The full identifier of the file containing the student's comments on a lesson. The name, includes path and extension, as listed by the operating system when disk contents are requested.
	Argument format	Alphanumeric characters. Includes path name. May be case sensitive, depending on the operating system.
	Usage rules	Including this keyword, but with a null value, turns off comment collection.
		If the file already exists, the lesson opens it and appends its information. If the file does not exist, the lesson creates it.
	Example	Comments_file = C:\STUDTA\CSF003.CMT

5.1.5 [Evaluation] keywords (cont.)

Interactions_file=	**Definition**	The full identifier of the file containing the record of the student's interactions on a lesson. (Interactions are described in Chapter 7) The name, includes path and extension, as listed by the operating system when disk contents are requested.
	Argument format	Alphanumeric characters. Includes path name. May be case sensitive, depending on the operating system.
	Usage rules	Including this keyword, but with a null value, turns off interaction data collection for this session.
		If the file already exists, the lesson opens it and appends the new information. If the file does not exist, the lesson creates it.
	Example	Interactions_file = C:\STUDTA\CSF003.INT
Objectives_status_ file=	**Definition**	The full identifier of the file containing information on the objectives covered in a lesson. The name, includes path and extension, as listed by the operating system when disk contents are requested.
	Argument format	Alphanumeric characters. Includes path name. May be case sensitive, depending on the operating system.
	Usage rules	Including this keyword, but with a null value, turns off objectives data collection for this session.
		If the file already exists, the lesson opens it and appends its information. If the file does not exist, the lesson creates it.
	Example	Objectives_status_file = C:\STUDTA\CSF003.OBJ

5.1.5 [Evaluation] keywords (cont.)

Path_file=	**Definition**	The full identifier of the file containing information on the path through the lesson taken by the student. The name, includes path and extension, as listed by the operating system when disk contents are requested.
	Argument format	Alphanumeric characters. Includes path name. May be case sensitive, depending on the operating system.
	Usage rules	Including this keyword, but with a null value, turns off collection of path information for this session.

If the file already exists, the lesson opens it and appends its information. If the file does not exist, the lesson creates it. |
	Example	Path_file = C:\STUDTA\CSF003.PTH
Performance_file=	**Definition**	The full identifier of the file containing information on the student's performance in complex scenarios, such as simulations.
	Argument format	Alphanumeric characters. Includes path name. May be case sensitive, depending on the operating system.
	Usage rules	Including this keyword, but with a null value, turns off collection of performance information for this session.

If the file already exists, the lesson opens it and appends its information. If the file does not exist, the lesson creates it. |
| | **Example** | Performance_file = C:\STUDTA\CSF003.PFC |

5.1.6   **[Objectives_Status]**

Definition   An objective identifier and an indication of what the student has done
                        on previous attempts on the lesson. The student can pass, fail, or not
                        attempt an objective. These objectives are those associated with the
                        current launching lesson, not all the objectives in the course/curriculum.

                        Three keywords are available in this group:

                        J_ID.1
                        J_Score.1
                        J_Status.1

Example   [objectives_status]
                        J_ID.1 = APU1684
                        j_status.1 = passed

5.1.6 [Objectives_Status] keywords (cont.)

J_ID.5=	**Definition**	An internally, developer defined, lesson-specific identifier for an objective. This is the same identifier as the one that appears in the courseware interchange Descriptor file in the field labeled "Developer ID." (The definition for Developer ID is on page 164.)
	Keyword format	Each J_ID keyword has an extension to make it unique. The extension is a period followed by a number -- from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Data format	Alpha-numeric string. No internal spaces.
	Usage rules	There may be multiple ID's in the group Objectives_Status but each must have a unique extension.
		Since the value of each J_ID is an a string representing objective ID internally defined in the CBT courseware, the CMI system needs to provide a means storing and referencing these (lesson specific) ID's.
		In addition , the CBT courseware use must include a list of these internally defined objectives in the course structure Descriptor (.DES) file.
	Examples	J_ID.1 = A1373

5.1.6 [Objectives_Status] keywords (cont.)

J_Score.1=	**Definition**	Indication of the score obtained by the student after each attempt to master an objective. A maximum and minimum may accompany score.
		A semicolon separates multiple attempts. Commas separate maximum and minimum values. If the CMI system only stores a single score for each objective, this status reflects the student's last attempt on the objective.
	Keyword format	The SCORE keyword has an extension to make it unique. The extension is a period followed by a one to four-digit number -- from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Optional values	The score may be followed by two values: a maximum and minimum.
		Maximum This is the largest score the student could have achieved with the interactions that he experienced.
		Minimum This is the smallest score that the student could have achieved with the interactions he experienced.
	Data format	One or more numbers that may include a decimal point, separated by commas or semicolons. Numbers separated by commas are Score, Maximum, Minimum in that order. Scores separated by semicolons represent different uses of the objective.
		If multiple attempts (or uses) are being stored, the score on most recent attempt is first. The earliest or first attempt appears last.
	Usage rules	Must have the same extension as its corresponding ID or J_ID. Indicates the score of the corresponding objective.

5.1.6 [Objectives_Status] keywords (cont.)

Example 1   J_Score.1 = 2,2
                     ; during student's first attempt, a score of 2 was
                     ; achieved, and the maximum possible was 2.

Example 2   J_ID.1= 1
                     J_Score.1 = 87
                     J_ID.2= 2
                     J_Score.2 = 3;5
                     ; during first attempt, student scored 5, on his
                     ; second attempt he scored 3

Example 3   J_Score.1 = 3,5;2,5
                     ; during the first use the student scored 2 out of a
                     ; possible 5. During the second use of the objective,
                     ; he scored 3 out of a possible 5.

Example 4   J_Score.1 = 6.5,10,-3
                     ; The student scored 6.5 out of a possible 10, and
                     ; could have done as poorly as to have scored -3.

Example 5   J_Score.1 = 9.5,10,0;6.3,10,0
                     ; On the first use of the objective, the student scored 6.3
                     ; out of a possible 10. Minimum would have been 0.
                     ; On the last (second) use, the student scored 9.5 out of
                     ; a possible 10.

5.1.6 [Objectives_Status] keywords (cont.)

J_Status.4=	**Definition**	Indication of the status of an objective at the time a lesson is launched. Five statuses are possible.

- **Pass** (or p or pass) - The student has mastered the objective.

- **Complete** (or c) - The student has gone through all segments of the lesson related to the objective. He may or may not have passed. The CMI system may make the judgment of whether he passed based upon his score.

- **Fail** (or f). Failed may be followed by a comma and an integer number indicating the number of times the objective has been failed.

- **Incomplete** (or i) - The student has not gone through all the segments of the lesson related to this objective.

- **Not attempted** (or n or NA) - The student has not gone through any of the segments of the lesson related to this objective.

- **Browsed** (or b). The student launched the lesson with a CMI mode of Browse on the initial attempt.

	Keyword format	Each STATUS keyword has an extension to make it unique. The extension is a period followed by up to a four-digit number -- from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Data format	Single word or letter. Only the first character is significant.

5.1.6 [Objectives_Status] keywords (cont.)

Usage rules   Must have the same extension as its corresponding ID or J_ID. Indicates the status of the corresponding objective.

There should never be more than one status keyword associated with a single objective. However, if more than one status does appear after an objective, then the first status to appear is the one that is assumed correct by the lesson.

If there is no corresponding ID or J_ID, the status is ignored.

If there is no status associated with an objective ID, then the assumed status is **na (not attempted)**.

Examples   j_id.3 = 1987
j_status.3=p
j_id.6 = 1942
J_STATUS.6 = f
J_ID.92 = 1847
J_Status.92 = N

5.1.7   **[Student_Data]**

Definition   Information in this data is to support customization of a lesson based on
                    a student's performance. For instance, the lesson could provide a
                    different entry point to the student based on this data.

                    Notice the difference between Student_Data and
                    Student_Demographics. Demographics data describes the student
                    before he begins the course. He brings it with him into the course.
                    Student_Data is what the CMI system learns of the student after he
                    begins and as he progresses through the course. Student_Data describes
                    the student's performance in the course.

                    There are currently 6 keywords defined for this group.

                    Attempt_Number
                    Lesson_Status.1
                    Mastery_Score
                    Max_Time_Allowed
                    Score.1
                    Time_Limit_Action

5.1.7 [Student_Data] keywords

| **Attempt_Number=** | **Definition** | Number of times the student has been in, or previously used the lesson. If the CMI system is launching the lesson for the student for the first time, the attempt_number = 0. |

| | | Note: A session may be defined as the number of uses of an assignable unit until it is completed or passed. A completed assignable unit represents an attempt at that assignable unit. Students may be allowed to take assignable units more than once for credit. The multiple completions of assignable units are attempts of that assignable unit.

Because of the definition for the attempt_number keyword in this document, systems following these guidelines are reporting session numbers (as session is defined in this note.) |

| | **Data format** | Integer number from 0 to 100. |

| | **Examples** | attempt_number = 3
Attempt_Number=16 |

5.1.7 [Student_Data] keywords (cont.)

Lesson_Status.1=   **Definition**   Indication of the status of the lesson after each attempt. If the CMI system only stores a single status for each lesson, this status reflects the student's last use of the lesson.

Possible statuses are described with the *Lesson_Status* keyword on page 69.

Keyword format   The LESSON_STATUS keyword has an extension to make it unique. The extension is a period followed by a one to four-digit number -- 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)

Usage rules   Normally, there is a separate LESSON_STATUS for each attempt indicated by the ATTEMPT_NUMBER. If the student has never looked at the lesson before, the attempt status number is 0 -- Lesson_Status.0. The first LESSON_STATUS following the ATTEMPT_ NUMBER is for the last time the lesson was attempted by the student.

If the ATTEMPT_NUMBER is greater than the number of LESSON_STATUS keywords following, or different than the Lesson_Status extension, then the first Lesson_Status line is assumed for the last or most recent attempt before the current lesson launch.

Example 1   Attempt_number = 3
Lesson_Status.3 = failed
Lesson_Status.2 = incomplete
Lesson_Status.1 = not attempted

5.1.7 [Student_Data] keywords (cont.)

	Example 2	Attempt_Number=4
		Lesson_Status.4=p
		Lesson_Status.3=p
		Lesson_Status.2=i,r
		Lesson_Status.1=f,l
		; In this example, the author of the lesson decides that
		; once a student achieves "Pass," in all subsequent
		; attempts at the lesson the student retains his
		; "Passed" status.

Mastery_Score=

Definition When the lesson score is greater than or equal to the mastery score, the student is considered to have passed, or mastered the content. In some cases, the lesson does not know what this passing score is, because it is determined by the CMI system (the human controlling the CMI system actually).

When the mastery score is not known but needed by the lesson, it is passed to the lesson by the argument of this keyword.

The mastery_score for each lesson is determined by the Mastery_Score field in the appropriate record in the Assignable Unit file, as described in Section 6.2, page 155.

Argument format Integer number. (-32,768 to +32,767)

Usage rules For a CMI system to support Mastery_Score, it must be able to change the lesson status based on the score passed to it from the lesson. Just passing a Mastery_Score to a lesson does not constitute full support for this feature.

Examples mastery_score = 75
Mastery_Score = 100
MASTERY_SCORE=5

5.1.7 [Student_Data] keywords (cont.)

Max_Time_   **Definition**   The amount of time the student is allowed to have in the
Allowed=   current attempt on the lesson. See **time_limit_action** for
                                           the lesson's expected response to exceeding the limit.

                                           Max_Time_Allowed is determined by the value of this
                                           field in the Assignable Unit file described in the Course
                                           Structure chapter in Section 6.2, page 155.

                       Data   Hours, minutes, and seconds separated by a colon.
                       format   hh:mm:ss

                       Usage rules   Three numbers separated by colons are always required
                                           to express time.

                       Examples   max_time_allowed = 0:14:30
                                           Max_Time_Allowed = 2:03:00
                                           MAX_TIME_ALLOWED = 1:09:00

5.1.7 [Student_Data] keywords (cont.)

Score.1=

Definition	Indication of the score obtained by the student after each previous attempt. A maximum and minimum may accompany score.
	Commas separate maximum and minimum values. If the CMI system only stores a single score for each lesson, this score reflects the student's last attempt in the lesson.
Keyword format	The SCORE keyword has an extension to make it unique. The extension is a period followed by a one to four- digit number -- from 00 to 99.1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
Format	One or more numbers that may include a decimal point, separated by commas. Numbers separated by commas are Score, Maximum, Minimum in that order.

Example 1
Score.1 = 2,3
; during student's first attempt, a score of 2 was
; achieved out of a possible 3..
Score.2 = 3,3
; in the second, and latest use of the lesson, the student
; scored 3 out of 3.

Example 2
score.1= 87
score.2 = 93
score.3 = 100

Example 3   Score.1 = .75

Example 4   Score.1 = .75,1,0

5.1.7 [Student_Data] keywords (cont.)

Time_Limit_   **Definition**   Tells the lesson (or test) what to do when the
Action=   max_time_allowed is exceeded. There are two
                                       arguments for this keyword.
                                       - What the lesson should do -- Exit or Continue
                                       - What the student should see -- Message or No
                                         message
                                       Time_limit_action is determined by the value of this
                                       field in the Assignable Unit file described in the Course
                                       Structure chapter in Section 6.2, page 155.

                     Format   Two letters, words, or phrases separated by a comma.
                                       The possible arguments are
                                       > **Exit** (or E or e)
                                       > **Continue** (or C or c)
                                       > **Message** (or M or m)
                                       > **No_Message** (or N or n)

                                       Only the first letter of each word or phrase is significant.
                                       Capitalization is ignored.

                     Examples   time_limit_action = Exit, Message
                                       ;   The lesson presents a message to the student
                                       ;   indicating he has exceeded the time
                                       ;   limit in the lesson, and then exit or quit.
                                       Time_Limit_Action=E,N
                                       ;   The lesson quits or exits with no message to the
                                       ;   student. He jumps to the CMI environment.

                                       time_limit_action = N,C
                                       ;   When the student exceeds his time limit in the
                                       ;   lesson, no message is presented, the lesson
                                       ;   continues. Notice that the order in which the
                                       ;   keywords appear is not significant.
                                       TIME_LIMIT_ACTION = continue, message
                                       :   The student receives a message when he exceeds
                                       ;   the time limit. The lesson continues
                                       ;   after presenting the message.

5.1.8   **[Student_Demographics]**

Definition   Attributes of a student, which he possessed prior to entering the course. Some of this information could be of use to a lesson. Each item of information is preceded by a keyword and therefore limited to a single line. Typical demographic data includes the student's name, job title, years of experience, and native language.

Each individual data item in this group is optional. As many or few of the following as desired may be used. Additional demographic items may be added that are unique to an implementation. However, if any of the items listed in this guideline are used, they should appear in the form and format described.

Keywords   The following keywords are defined for this group.

> **City**
> **Class**
> **Company**
> **Country**
> **Experience**
> **Familiar_Name**
> **Instructor_Name**
> **Job_Title**
> **Native_Language**
> **State**
> **Street_Address**
> **Telephone**
> **Years_Experience**

5.1.8 [Student_Demographics] keywords

City=	**Definition**	Portion of student's current address.
	Format	Alphabetic string. May include spaces.
	Examples	city = Seattle City= Montreal CITY =Long Beach

Class=	**Definition**	A predefined training group to which a student belongs.
	Format	Alphabetic string. May include spaces.
	Examples	class = BO767-21 CLASS=FT.767-1991-06

Company=	**Definition**	Student's place of employment.
	Format	Alphabetic string. May include spaces.
	Examples	COMPANY= BOEING company= United Airlines Company = Skywest Airlines, Ltd.

Country=	**Definition**	Portion of student's current address.
	Format	Alphabetic string. May include spaces.
	Examples	COUNTRY = USA Country=Canada Country= New Zealand

5.1.8 [Student_Demographics] keywords (cont.)

Experience=	**Definition**	Information on the student's past that might be required by a lesson to determine what to present, or what presentation strategies to use.
		For instance, a pilot may have experience flying a DC-9, 737-200, and 727. This would indicate no "glass cockpit" experience. Consequently extra information on electronic displays might be appropriate.
	Format	Alphabetic string. May include spaces. May be lesson design specific.
	Examples	experience = 737-200, DC-9, 727 Experience = 4000-4j ; Unique code. Student has 4000 hours experience in ; 4 engine jet transports.
Familiar_Name=	**Definition**	In some cases, a lesson 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 lesson how it should refer to the student.
	Format	Alphabetic string. May include spaces.
	Examples	familiar_name = Clint Familiar_Name= Mr. Gregory

5.1.8 [Student_Demographics] keywords (cont.)

Instructor_Name=	**Definition**	Name of the instructor responsible for the student's understanding of the material in the lesson.
	Format	Last name, first name and middle initial. Last name and first name are separated by a comma. Spaces in the name must be honored.
	Examples	Instructor_Name= Henry Willoughby instructor_name = Mark Ashtonbury Jr. instructor_name = Haight, Ash
Job_Title=	**Definition**	Title of the position the student currently has in the company which employs him.
	Format	Alphabetic string. May include spaces.
	Examples	job_title = Pilot Job_Title = Captain JOB_TITLE=First Officer job_title = Vice President Engineering
Native_Language=	**Definition**	The language with which the student is most familiar. This may not be the preferred language for the instructional delivery.
	Format	Alphabetic string. May include spaces.
	Examples	Native_Language=English NATIVE_LANGUAGE = FRENCH native_language = Dutch

5.1.8 [Student_Demographics] keywords (cont.)

State=	**Definition**	Portion of student's current address. Segment of a country, also called a province.
	Format	Alphabetic string. Spaces are significant.
	Examples	state = Illinois State = British Columbia

Street_Address=	**Definition**	Portion of student's current address.
	Format	Alphabetic string. Spaces are significant.
	Example	1993 West Oak Street

Telephone=	**Definition**	Telephone number of a student. May or may not include area and country codes. May be work phone or home phone.
	Format	Alphanumeric string. Spaces are not significant.
	Examples	telephone = (514) 239-6115 Telephone=61 93 20 12 TELEPHONE = 011-44-482-663622

Years_Experience =	**Definition**	Number of years the student has performed in current or similar position.
	Format	Integer number.
	Examples	Years_Experience = 3 years_experience=15 YEARS_EXPERIENCE = 8

5.1.9   **[Student_Preferences]**

Definition   Frequently lessons are designed to allow students to select options that
                        are appropriate for subsequent lessons. For instance, a lesson may have
                        audio with a software volume adjustment. When the volume is adjusted
                        it may be desirable to pass that volume preference to a lesson taken the
                        next day.

Usage rules   A CMI system must be prepared to hold this information and pass it on
                        to each lesson as long as the student remains in the course. One set of
                        preferences per student is needed.

                        The CMI holds each keyword and argument (the CMI may ignore blank
                        lines), then passes it to the next lesson. This enables the creation and
                        use of student preferences that are not yet defined in this standard.

                        Although there is no limit on the amount of space that may be needed
                        by Student_Preferences, the lesson author should keep in mind the limit
                        of 32K for the PARAM.CMI file.

Lesson behavior   A lesson using preference keywords only needs to pass changes back to
                        the CMI system. The CMI is responsible for maintaining the
                        accumulated set of preferences and passing the latest version of all
                        preference keywords to each lesson when it is launched.

Key words   The following key words are defined for this group.

                            Audio
                            Language
                            Lesson_Type
                            Speed
                            Text
                            Text_Color
                            Text_Location
                            Text_Size
                            Video
                            Window.1

5.1.9 [Student_Preferences] Keywords

Audio=

Definition	Audio may be turned off, or its volume controlled. The keyword arguments indicate whether the audio is turned off, or on.
Value format	Digit from -1 to 100.

> -1 is off
> > Any minus number is an off command.
>
> 0 is a no-change status.
> > The lesson uses its defaults or the status of the audio remains the same as the last lesson taken on the terminal.
>
> 1 to 100 is volume level.
> > 1 is soft, 100 is loudest possible.

Examples	audio= -1 AUDIO = 33

Language=

Definition	For lessons with multi-lingual capability, this keyword should be used to identify in what language the information should be delivered.

A student may prefer to experience a lesson in a language other than his native language.

Theoretically a lesson could make decisions based on the contents of **Language** and **Native_Language**. For instance, if **Language** does not equal **Native_Language**, the audio could be played back slower than normal.

Format	Alphabetic string. May include spaces.
Examples	Language=English LANGUAGE = FRENCH language = Dutch

5.1.9 [Student_Preferences] Keywords (cont.)

Lesson_Type=

Definition Student preferences set in one type of lesson may be meaningless when applied to another type of lesson.

Assume for instance that there are two different authoring systems used to develop two lessons, one following the other. If the second lesson receives student preferences for Text_Color from the first lesson, the meaning could be entirely different. Text_color of white for the first lesson could be defined as 16. Color white in the second lesson could be defined as 256,256,256.

To verify that the parameters are applicable to a lesson, a data needs to be established to indicate the type of lesson. The lesson receiving student preferences can then determine if the preferences are applicable to itself.

The following keywords are sensitive to lesson type:

- Text_Color
- Text_Location
- Text_Size
- Video

Value format Alpha-numeric. No spaces. At least three characters.

Usage rules The type identifier should be at least three characters in length. This will ensure a very low probability that two lessons of different types will have the same type identifier by coincidence.

5.1.9 [Student_Preferences] Keywords (cont.)

Speed=   **Definition**   Lessons may sometimes be difficult to understand because of the pace. This is especially true if the language of the CBT presentation is not the students first language. In such cases, understanding can be helped by slowing the lesson flow.

Sometimes, a student may be bored or irritated because the lesson seems so slow. In such cases comprehension and interest level may be increased by increasing the speed of the lesson delivery.

This parameter allows retaining the student's preference for faster or slower lesson flow.

Value format   Digit from -100 to 100.

-100 is slowest pace available in the system

0 is a no-change status.
    The lesson uses its defaults. Lesson moves at its normal speed.

100 is maximum pace available in the system.

Example 1   speed= -100

Example 2   ; If a system only has three speeds, slower, normal, and
; faster, any positive number (+1 and above) would
; result in the use of the faster speed.
SPEED = 33

5.1.9 [Student_Preferences] Keywords (cont.)

Text=

Definition   In a lesson designed for audio, it may be possible to turn off the audio, and view the audio content in a text window. Or it may be possible to leave the audio on, and request that the text be presented simultaneously with the audio. Or it may be possible to make the text disappear so that only the audio and the screen graphics are available.

This keyword identifies whether the audio text appears in the lesson.

Value format   One of three digits.

    -1   text is off, not shown

    0   no change in status. Use default.

    1   text is on screen, shown to student

Examples   Text = -1
text=0
TEXT = 1

5.1.9 [Student_Preferences] Keywords (cont.)

Text_Color=	**Definition**	When there are preference options for a student, text color and text background may be selections available. This keyword enables storing these two parameters.
	Value format	Alpha-numeric. Text color first, then background color, separated by a comma. Note that color definitions cannot be separated by a comma. An R,G,B definition of color for instance, could not be written 256,123,215. It would have to have a different separator, like 256:123:215 or 256-123-215.
		The first time this command is used, its argument may be 0. However, when a lesson is run and the student selects a new text color and background, those colors are passed to the CMI system as TEXT_COLOR. They are saved in whatever format is provided by the lesson. Then it is passed back to the next lesson as the data of this keyword.
	Usage rules	0 (zero) is a no change status -- that is no change from the lesson default colors. Any other argument is system unique.
		If only a single color is defined, it is the assumed character color, and background color remains unchanged.
	Examples	text_color = 234-89-196,0-0-0 Text_Color = Light Blue, Dark Blue Text_Color = Light green, Black Text_color = 0

5.1.9 [Student_Preferences] Keywords (cont.)

Text_Location=

Definition

When a lesson has text on screen in a window, it may be possible for the student to move the text window to a location of his choice. This keyword allows that preferred location to be passed to subsequent lessons.

Value format

Alpha-numeric. 0 (zero) is a no change status -- that is no change from the default in the lesson. From the student's view there may be a significant change in colors from the previous lesson.

Any other argument is system unique. The first time this command is used, its argument may be 0. However, when a lesson is run and the student selects a new location for the text window, that location is passed to the CMI system as a TEXT_LOCATION. It is saved in whatever format it is provided by the lesson. Then it is passed back to the next lesson as the argument in this keyword.
.

Examples

Text_Location = 243,128
text_location=Lower-Right
TEXT_LOCATION = 1
text_location = 0

5.1.9 [Student_Preferences] Keywords (cont.)

Text_Size=	**Definition**	When a lesson has text on screen in a window, it may be possible for the student to select the size that is most comfortable for his eyes and viewing distance. This keyword allows that preferred size to be passed to subsequent lessons.
	Value format	Alpha-numeric. 0 (zero) is a no change status -- that is no change from the default in the lesson. From the student's view there may be a significant change in text size from the previous lesson..
		Any other argument is system unique. The first time this command is used, its argument may be 0. However, when a lesson is run and the student selects a new size for the text, that size is passed to the CMI system as a TEXT_SIZE. It is saved in whatever format it is provided by the lesson. Then it is passed back to the next lesson as the argument in this keyword.
	Examples	. Text_Size = 14 point text_size= 134% TEXT_SIZE = 1.5 text_size = 0
Video=	**Definition**	Video controls may be available in software for the student's use in a lesson with video. Options on tint, brightness, size, and centering of the image may be available. These controls are saved as a value in this keyword.
	Value format	System unique. Data is saved in whatever format it is created by the lesson, and passed to the next lesson in that format.
	Examples	Video= -1 video = 33, 16, 85

5.1.9 [Student_Preferences] Keywords (cont.)

Window.1=	**Definition**	Some lessons may allow the student to set the size and location of windows for some information in a lesson. For instance, video, help, and glossary information may be available in a window whose size and location is set by the student.
	Keyword format	The Window keyword has an extension to enable two or three window specifications to be set by the student. The extension is a period followed by a number -- from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Value format	System unique. Data is saved in whatever format it is created by the lesson, and passed to the next lesson in that format.
	Examples	Window.1=200x100@45,80 Window.2=220x100@540,80

5.2   CBT Lesson to CMI

Description   Information that a lesson must/may make available for a CMI system. The core items (which the lesson MUST make available) are first, followed by the optional items listed alphabetically. Constructing this file should be the first thing done by the lesson after launch

Group Names and Keywords	Function of Group

[Core]
    Lesson_Location
    Lesson_Status
    Score
    Time
	Information required by the CMI system to function.

[Core_Lesson]
    data is undefined and may be
    unique to each lesson
	Information required by the lesson for the student when he next uses it. Passed to the CMI system to hold and to return the next time the student starts this lesson.

[Comments] no key words <delimited>	Student comments on lesson.

[Objectives_Status]
    J_ID.1
    J_Score.1
    J_Status.1
	Information on objectives contained in the lesson.

[Student_Data]
    Tries_During_Lesson
    Try_Score.1
    Try_Status.1
    Try_Time.1
	Information on student performance for each attempt on a selected segment of the lesson without leaving the lesson.

[Student_Preferences]
    Audio
    Language
    Lesson_Type
    Text
    Text_Color
    Text_Location
    Text_Size
    Video
    Window.1
	Student selected options to be
	passed to next lesson he enters.

5.2.1   **[Core]**

Definition   This is the group name associated with the following keywords. Items
in this group are required in the file. The lesson may or may not update
the file periodically to indicate changes to these core items. However,
the file should be updated when the student leaves the lesson to indicate
the final status.

> **Lesson_Location**
> **Lesson_Status**
> **Score**
> **Time**

Examples

[CORE] Lesson_Status = Passed Lesson_Location = end score = 87 time = 00:18:00
[core] Lesson_Status=NA Lesson_Location=Intro score= time=00:02:38

5.2.1 [Core] keywords

Lesson_Location=	**Description**	See the Lesson_Location keyword on page 65.

Lesson_Status=	**Description**	This is the status determined by the lesson based on the student activity in the lesson. It may be accompanied by a flag.

Six statuses are possible. They are described in Section 5.2.1. on page 69.

Flags   Each status must be accompanied by a flag if one of the circumstances described below occurs. The additional flags are T, S, and L.

- **Time-out** (or t). This indicates the lesson ended because the lesson has determined an excessive amount of time has elapsed, or the **Max_Time_Allowed** has been exceeded.
- **Suspend** (or s). This indicates the student leaves the lesson with the intent of returning to it later at the point where he left.
- **Logout** (or l). This indicates that the student logged out from within the lesson instead of returning to the CMI system to log out. This implies that the lesson passed control to the CMI system, and the CMI system automatically logged the student out of the course -- after updating the appropriate files.
- **No Flag**. This indicates that the student had a normal exit**.**

> **Note**: These flags do not accompany the lesson status passed from the CMI to the CBT lesson. The only flags possible when passing information back to the lesson are the **ab initio** and **resume** flags.

Lesson_Status= **(cont.)**	**Argument format**	A word, character or phrase, optionally followed by a comma and one additional word or character. Only the first character is significant in each word.
	Example 1	Lesson_Status = incomplete,logout ; Student logged out without completing lesson
	Example 2	Lesson_Status = incomplete, suspend ; Student left without completing lesson. ; Student probably intends to return to the lesson.
	Example 3	Lesson_Status = not attempted ; Student looked at some part of the lesson, did not ; attempt to challenge it, and then left normally.
	Example 4	lesson_status = p,l ; Student passed the lesson, and wants to log out of the ; course

Score=		
	Definition Format	Definition and format are the same as described under the *Score* keyword on page 74.
	Usage rules	If there is no score to report, the lesson returns a blank or null, that is "Score="

| **Time=** | **Definition** | This is the amount of time in hours, minutes and seconds that the student has spent in the assignable unit at the time he leaves it. That is, this represents the time from beginning of the session to the end of a single use of the unit.

Other aspects of time are the same as described under the Time keyword on page 76. |

> Note: It is assumed that the CMI system will accumulate the individual session times into the total time for all of the assignable unit uses.

5.2.2   **[Core_Lesson]**

Definition   This is normally data that is created by the lesson and stored by the CMI
                        system to pass back to the lesson for the next time the lesson is run.
                        Information needed by the lesson for a restart.

                        See amplified description in Section 5.1.2, page 77.

5.2.3   **[Comments]**

Description   This group contains freeform feedback from the student. He may have
the option of leaving comments at any point in the lesson, or he may be
asked for comments at the end of the lesson. In any case, each comment
is preserved in this group and identified as comment 1, 2, 3 etc.

The comment may also have an indication of where or when in the
lesson it was created. A location may be tagged and embedded in the
comment.

Format   Any number of lines with any alphanumeric and special characters. The
carriage returns are not used at the end of every line in the file. Word-
wrap is allowed. Carriage returns, when in a string should be honored
by the lesson (application).

For example:

```
                                                              ...abc
def
```

when displayed to the student becomes

```
...abcdef
```

while:

```
...abc<cr>
def
```

when displayed to the student becomes

```
...abc
def
```

Comment   Each comment is preceded by a less-than sign, the number of the
delimiters   comment, and a greater-than sign. Each comment is ended by a less-
than, an "e", a period, the comment number, and a greater-than. The E is
not case sensitive.

Delimiter examples   <1>This was a lousy lesson!!!<e.1>
<13>You mispellled the word defueling.<E.13>
<6> I don't understand why answer B is correct. <e.6>

Comment data   The lesson location of the comment and any other data that may be of value to the developer or instructor can also be included. The less-than and greater-than symbols are used with an "L." embedded. The location identifier which follow the "L." may include spaces. The location identifier that follows the period is a function of how the lesson author wants to identify location.

Data examples   <l.frame12>
<L. page 36>
<L.fuel3-21>
<L. Fuel part 3: interaction 24>

The location parameter indicates exactly where in the lesson the student was when he created the comment. The location is placed inside the comment delimiters along with any other information deemed desirable.

Size   4096 bytes.

Cross reference   Locations can correspond to lesson elements. Lesson elements are arbitrary divisions of an assignable unit that have been uniquely named (ID). They are described in Section 7.5 under the group name [PATH].

Comment examples

[Comments] <1><L.apu.intro>Why is the APU swich labels reversed from my airplane here?<E.1> <2><L.apu.q3>I don't understand why B is right. Shudn't the fire handle be checked first?<E.2>
[Comments] <1> Electrical is mispelled here.<E.1> ; Without a location, some comments are less useful.

5.2.4   **[Objectives_Status]**

Similar to CMI to   This group corresponds to the OBJECTIVES_STATUS group in section
Lesson data   "5.1 CMI to Lesson." It contains the same information in the same
                         format as that which the CMI system sends to the lesson. However, the
                         data in this group represents only the status during the current attempt.
                         This group does not have statuses for objectives which are not
                         challenged during the current attempt on the lesson.

Definition   An objective identifier and an indication of what the student has done
                         on previous attempts on the lesson. The student can pass, fail, or not
                         attempt an objective. These objectives are those associated with the
                         current launching lesson, not all the objectives in the course/curriculum.

                         Four keywords are available in this group:

                         J_ID.1
                         J_Score.1
                         J_Status.1

Example   [objectives_status]
                         J_ID.1 = APU1684
                         j_status.1 = passed

5.2.4 [Objectives_Status] keywords

J_ID.2=	**Description**	See **J_ID.1** on page 86

J_Score.1=	**Description**	See **J_Score.1** on page 87.

J_Status.4=	**Definition**	See **J_Status.1** on page 89.

5.2.5   **[Student_Data]**

Definition   This group provides more information on an individual student's
performance than the data available in the [CORE] group.

For instance, the *Score* and *Status* returned in [CORE] are the final, or
last score and status achieved in the lesson. In some lessons, a student
may be allowed to take a test multiple times without leaving a lesson.
The score, or result of each try could be of interest to the instructor.
This group allows the capture of multiple scores for multiple tries.

There are currently 4 keywords defined for this group.

    Tries_During_Lesson
    Try_Score.1
    Try_Status.1
    Try_Time.1

5.2.5 [Student_Data] keywords

Tries_During_Lesson=	**Definition**	Total number of efforts to complete the lesson before leaving it. This may correspond to the number of attempts to complete an embedded test or exercise.
	Keyword format	The TRY_STATUS keyword has an extension to make it unique. The extension is a period followed by a number - - from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Data format	Integer number from 0 to 100.
	Examples	tries_during_lesson = 3 Tries_during_Lesson=16

Try_Score.1=	**Definition**	The score at the completion of each attempt during a single use of a lesson. This may correspond to the number of times an embedded test or exercise was completed.
	Keyword format	The TRY_STATUS keyword has an extension to make it unique. The extension is a period followed by a number - - from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Data format	Integer number. For letter grades, numbers must be substituted.
	Usage rules	The numerical extension (.1, .2, etc.) identifies for which attempt the score is appropriate.
	Examples	Try_Score.1=95 TRY_SCORE.2 = 80 try_score.3 = 100

5.2.5 [Student_Data] keywords (cont.)

Try_Status.1=	**Definition**	The status of the lesson after each attempt during a single use of the lesson. This may correspond to the status at the completion of each attempt to complete an embedded test or exercise in the lesson.

		Possible statuses are the same as for the Lesson_Status:
		    Passed
		    Completed
		    Failed
		    Incomplete
		    Not attempted

	Keyword format	The TRY_STATUS keyword has an extension to make it unique. The extension is a period followed by a two-digitnumber -- from 00 to 99.1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Usage rules	Only the first character in each status is significant.
		The numerical extension (.1, .2, etc.) identifies for which attempt the score is appropriate.
	Example 1	Tries_During_Lesson = 3 Try_Status.1 = failed Try_Status.2 = incomplete Try_Status.3 = not attempted
	Example 2	Tries_During_Lesson=4 Try_Status.4=p Try_Status.2=p Try_Status.3=i Try_Status.1=f ; Order of listing is irrelevant. The extension indicates ; the sequence of the statuses.

5.2.5 [Student_Data] keywords (cont.)

Try_Time. 01=	**Definition**	For each effort to get a passing score, or complete an exercise, a time can be recorded. The first effort time corresponds to . 01, the second effort to . 02, and so forth.
	Keyword format	The TRY_STATUS keyword has an extension to make it unique. The extension is a period followed by a number -- from 1 to 9999. This number should not be zero padded. (.0009 for instance is an illegal extension for the number .9)
	Data format	Hours, minutes, and seconds separated by a colon. hh:mm:ss
	Usage rules	The numerical extension (.1, .2, etc.) identifies for which attempt the time is appropriate. Three numbers separated by colons are always required to express time.
	Examples	try_time.1 = 00:14:37 Try_Time.2 = 00:00:54 TRY_TIME.3=01:14:21

5.2.6   **[Student_Preferences]**

Definition   Frequently lessons are designed to allow students to select options that
                          are appropriate for subsequent lessons. For instance, a lesson may have
                          audio with a software volume adjustment. When the volume is adjusted
                          it may be desirable to pass that volume preference to a lesson taken the
                          next day.

Key words   The following key words are defined for this group.

                               Audio
                               Language
                               Lesson_Type
                               Text
                               Text_Color
                               Text_Location
                               Text_Size
                               Video
                               Window.1

5.2.6 [Student_Preferences] Keywords

Audio=	**Description**	See Audio= under [Student_Preferences] on page 104.
Language=	**Description**	See *Language* under [Student_Preferences] on page 104.
Lesson_Type=	**Description**	See *Lesson_Type* under [Student_Preferences] on page 105.
Text=	**Description**	See *Text* under [Student_Preferences] on page 107.
Text_Color=	**Description**	See *Text_Color* under [Student_Preferences] on page 108.
Text_Location=	**Description**	See *Text_Location* under [Student_Preferences] on page 109.
Text_Size=	**Description**	See *Text_Size* under [Student_Preferences] on page 110.
Video=	**Description**	See *Video* under [Student_Preferences] on page 111.
Window.1=	**Description**	See *Window.1* under [Student_Preferences] on page 111.

5.3   Error and Default Conditions

Error conditions and recommended actions are defined as follows:

5.3.1   **File Creation, File Read, and File Write Errors**

---

1. ERROR   CMI system cannot create the CMI-to-CBT lesson file

SUGGESTED ACTION :
CMI system sends an error message to the student that the lesson cannot be run, and does not initiate the CBT lesson.

2. ERROR   CBT system cannot read the CMI-to-CBT lesson file

SUGGESTED ACTION :
CBT system puts out an error message to the student and returns to CMI; CBT system does not create the CBT-to-CMI file.

3. ERROR   CBT system cannot create the CBT-to-CMI file

SUGGESTED ACTION :
CBT system runs as normal without the writing the file; CMI handles as error condition 5.

4. ERROR   CBT system cannot write the CBT-to-CMI file

SUGGESTED ACTION :
CBT system puts out an error message and returns to CMI without writing the file; CMI handles as error condition 5.

5. ERROR   CMI system cannot read the CBT-to-CMI file

SUGGESTED ACTION :
CMI system puts out an error message, assumes assignment is complete, stores default student performance data, and makes the next assignment.

Suggested minimum default performance data:
    time   = 0
    score   =
    lesson status = complete

5.3.2   **Data and File Format Errors**

1. ERROR   Missing group

   SUGGESTED ACTION :
   If core group, use default values. If optional group, process as if no data supplied.

2. ERROR   Illegal or misspelled group name

   SUGGESTED ACTION :
   If core group, use default values. If optional group, process as if no data supplied.

3. ERROR   Duplicate group

   SUGGESTED ACTION :
   Process only the first instance of the group.

4. ERROR   Missing keyword item

   SUGGESTED ACTION :
   If core group, use default value. If keyword in optional group, process as if no data supplied.

5. ERROR   Illegal or misspelled keyword ID

   SUGGESTED ACTION :
   If core group, use default value. If keyword in optional group, process as if no data supplied.

6. ERROR   Duplicate keyword

   SUGGESTED ACTION :
   Process only the first instance of the keyword.

7. ERROR   Illegal keyword data (e.g. SCORE = ABV)

                    SUGGESTED ACTION :
                        If core group, use default value. If keyword in optional group, process as if no data supplied.

8. ERROR   Illegal comment delimiter (e.g. missing <#>)

                    SUGGESTED ACTION :
                        Process as if no data supplied.

5.3.3   **CBT and CMI System Mismatch Errors**

---

1. ERROR   CMI system creates CMI-to-CBT lesson data and CBT system is not modified to process the data

SUGGESTED ACTION :
> CBT systems runs as normal and does not put out the CBT ➔ CMI data file. The CMI system handles the situation as if there was an error reading the CBT ➔ CMI file.

2. ERROR   CBT system creates CBT-to-CMI data and CMI system is not designed to process the data

SUGGESTED ACTION :
> CMI system runs as normal without the results being reported back. Probably should make sure that the CBT systems deletes any old instances of the lesson->CMI file before writing initial data to the file.

3. ERROR   CBT system does not create CBT-to-CMI data and CMI system is ready to process the data

SUGGESTED ACTION :
> CBT system runs as normal not putting out the CBT ➔ CMI data file. The CMI system handles the situation as if there was an error reading the CBT ➔ CMI file.

6.0   COURSE STRUCTURE DATA

Purpose   The purpose of defining a CMI structure interchange format, is to simplify
                     the process of moving a course from one CMI system to another.

                     After moving a course, a review-and-modify effort is going to be required
                     The existence of standard interchange files however, should eliminate a
                     large number of the man-hours necessary to input a new course from
                     scratch.

Chapter contents This chapter describes the basic concepts upon which the course
                     structure description is based. It also describes the levels of complexity
                     that may be used to describe a course structure using this system.

                     This is followed by a detailed description of each of the seven possible
                     files that may be used in describing a course.

                     Finally, there are a couple of examples of course descriptions.

Section	Subject
6.0.1	Basic Concepts
6.0.2	Course Building Blocks
6.0.3	Levels of Complexity
6.1	The Course File
6.2	Assignable Unit Table
6.3	Descriptor Table
6.4	Course Structure table
6.5	Objectives Relationships Table
6.6	Prerequisites Table
6.7	Completion Requirements Table
6.8	Examples

6.0.1   **Basic Concepts**

The basic concepts   The files containing the structure of a course need to answer the question, "What information does a CMI system need, to present the training material to the student in the way desired by the designer?"

Table: implied order   The approach taken by this document assumes that the answer can be *implied* in a table that contains all of the lessons and lesson groups in a course.

The answer can be made *explicit* by stating prerequisites for each lesson (or assignable unit) in the course. When pre-conditions are set that must be met before a student can select or be assigned a lesson, each lesson, assumes a place in the course structure.

For instance, assume there is a course of six lessons. The order of the lessons can be implied by putting them in a simple table; then reading the table left to right, and top to bottom.

Course Hierarchy Table

Root	Lesson 1	Lesson 2	Lesson 3	Lesson 4	Lesson 5	Lesson 6

Prerequisites: explicit order   To make this order explicit, assume lesson 6 has a prerequisite of the student having completed lesson 5, and lesson 5 requires passing lesson 4, and lesson 4 requires completion of lesson 3, etc. This results in the linear presentation of the lessons in sequence from 1 through 6.

Prereq for 2   Prereq for 3   Prereq for 4   Prereq for 5   Prereq for 6

```
┌─────────┐      ┌─────────┐      ┌─────────┐      ┌─────────┐      ┌─────────┐      ┌─────────┐
│ Lesson  │ ══>  │ Lesson  │ ══>  │ Lesson  │ ══>  │ Lesson  │ ══>  │ Lesson  │ ══>  │ Lesson  │
│   1     │      │   2     │      │   3     │      │   4     │      │   5     │      │   6     │
└─────────┘      └─────────┘      └─────────┘      └─────────┘      └─────────┘      └─────────┘
```
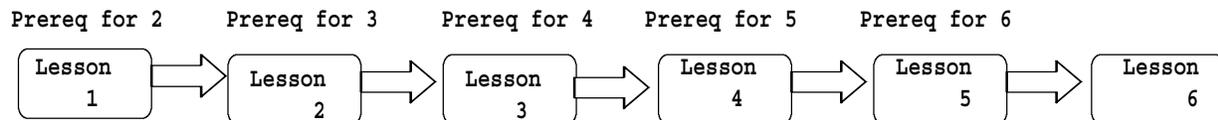
Prerequisites   Of course, even with prerequisites there are cases where it is desirable to let the student chose the order. If three lessons have exactly the same prerequisites, then the student has an option -- after meeting the prerequisites -- of selecting any of the three.

In this approach, prerequisites can be defined in terms of completed lessons, or mastered objectives.

Prerequisite table

Assignable Unit	Prerequisites
Lesson 1	None
Lesson 2	Lesson 1
Lesson 3	Lesson 2
Lesson 4	Lesson 3
Lesson 5	Lesson 4
Lesson 6	Lesson 5

Descriptions   In addition to files describing the course hierarchy and prerequisites there need to be files describing the elements in the course. This is textual information and not required to determine the order in which the student can take the course material. This information includes the titles of the various items in the course and a narrative description of them when desired.

Status   In the Chapter on CMI/Lesson communication (Chapter 5) there is a description of information passed back to the CMI system as the student leaves the lesson. This information includes *Lesson Status* and *Objectives Status*.

These statuses can be used to determine whether prerequisites are met for each structure element (assignable units and blocks) in the course.

6.0.2   **Course Building Blocks**

Structure elements   The parts of the course that can be rearranged to define the order in which a student can experience a course are referred to in this document as *structure elements*. These are the *assignable unit* and the *block*. The order in which these appear in the course defines its structure.

- Assignable unit, also referred to as a lesson. This is one of the two structure elements.

- Block. This is the second structure element. A block is simply a grouping of lessons and other blocks.

Another building block, which may be required to define prerequisites for a course, is the objective.

- Objective. This is also mostly content information, that includes a title for each objective, and a narrative description if desired.

Course elements   These three items
- Assignable unit (lesson),
- Block, and
- Objective
are referred to as *course elements*.

6.0.3   **Levels of Complexity**

3 Levels

This guideline defines three levels of complexity in describing the course structure. Increasing the level of complexity from level 1 to 2 to 3 should result in:

- Less effort to review and modify the CMI system after importing the data.

- More complete description of the designer's intended usage of the course material.

Course description data

There are seven files that can be used to describe a course's content and structure. The level of complexity determines the number of files required and the amount of information required in each file.

Level 1

Attributes

- Description. (**Course**, **Descriptor**, & **Assignable Unit** files)

- Course structure. (**Course Structure** file)

This is the simplest level. It describes the contents of the course -- the lessons or 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.

Attributes

Level 2

> - Description. (**Course**, **Descriptor**, & **Assignable Unit** files)
>
> - Course structure(**Course Structure** file)
>
> - Simple Prerequisites (**Prerequisites** file)
>
> - Simple Completions (**Completion Requirements** file)

This level of complexity adds a possible single prerequisite for each structure element -- an assignable unit or a block.[4] The status of each prerequisite is simple: complete or incomplete. The order in which the student moves through the course can be forced by prerequisites.

This level also introduces the ability to identify a structural element whose 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).

Level 3

Level 3 is divided into two parts. A level 3 conforming system may support features described as Level 3a or Level 3b or both feature sets. Level 3a adds the ability to define complex prerequisites and complex completion requirements. Logical expressions may be used to describe these requirements. Level 3b describes the relationship of objectives to the course structural elements.

Supporting 3A and 3B allows the use of complex prerequisites and completions with objectives.

---

[4] At this level, block completion is determined by default rules. Specific and unique completion requirements for a block are defined only in level 4 and above structure descriptions.

Level 3a

| **Attributes** |
| --- |
| • Description. (**Course**, **Descriptor**, & **Assignable Unit** files)

• Course structure(**Course Structure** file)

• Complex Prerequisites (**Prerequisites** file)

• Complex Completions (**Completion Requirements** file) |

This level adds the ability to use logical expressions to describe prerequisites and completion requirements.

Level 3b

Attributes
• Description. (**Course**, **Descriptor**, & **Assignable Unit** files) • Course structure(**Course Structure** file) • Prerequisites (**Prerequisites** with objectives) • Completions (**Completion Requirements** with objectives) • Objectives (**Objective Relationships** file)

This level adds objectives to the structure. Mastery of these objectives can be used as prerequisites, as well as determining completion of assignable units and blocks.

Multiple prerequisites for each block or assignable unit can be defined with logic statements using **and** (&) and **or** (|) if 3A is supported as well as 3B..

Level	Attributes	
	Content Attributes	**Sequencing Attributes**
Level 1 Files Required	Course contents described CD AU DES	Order of elements is implied CS
Level 2 Additional File Required		Order of elements can be made explicit with simple prerequisites and completion requirements. CR PRE
Level 3a Additional and Augmented Files		Order of elements can be described with complex logical expressions. CR PRE
Level 3b Additional and Augmented Files	Objectives are added to course contents and described OR DES	Objectives may be used in logic statements. CR PRE
Legend: CD = Course Descript. PRE = Prerequisites AU = Assignable Unit CR = Completion Req. DES = Descriptor OR = Objective CS = Course Structure Relationships		Gray file symbols represent augmented files.
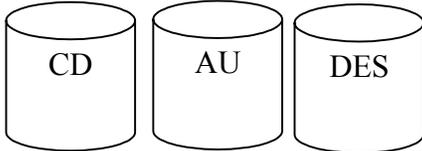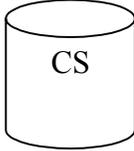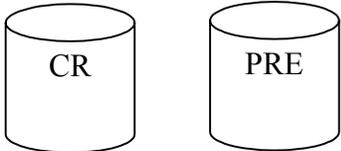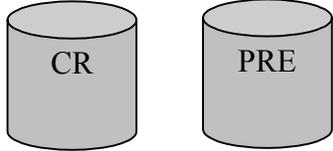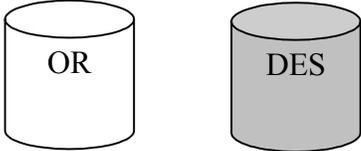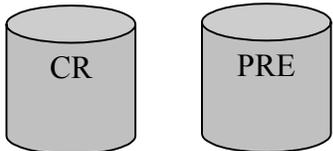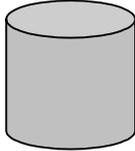
Example   Here is an example of a simple course structure. It has three blocks: Electrical, Power Plant, and Fuel. It has nine lessons. The student may select any block at any time, but inside a block, he must take the lessons in sequence.

For instance, when Power Plant is selected, the student must first complete the lesson on the Power Plant Fuel system, then complete Oil, complete Pneumatics, and finally complete Procedures. The student cannot select another block, until he has completed the Procedures lesson on Power Plant.

```
                    Electrical    (Block)

                         ┌───────────────────────────────────────┐
                         │     ⇓                            ⇑     │
              ══════⇒    │  ┌──────┐    ┌──────┐    ┌──────┐     │
                         │  │  AC  │ ⇒  │  DC  │ ⇒  │ Proc.│     │
                         │  └──────┘    └──────┘    └──────┘     │
                         └───────────────────────────────────────┘
                                   (Assignable Units)


                    Power Plant              (Block)

                         ┌──────────────────────────────────────────────┐
                         │     ⇓                                  ⇑      │
              ══════⇒    │ ┌──────┐   ┌─────┐   ┌───────┐   ┌──────┐   │
                         │ │ Fuel │ ⇒ │ Oil │ ⇒ │ Pneum.│ ⇒ │ Proc.│   │
                         │ └──────┘   └─────┘   └───────┘   └──────┘   │
                         └──────────────────────────────────────────────┘
                                    (Assignable Units)


                    Fuel      (Block)

                         ┌───────────────────────────┐
                         │    ⇓              ⇑        │
              ══════⇒    │ ┌────────┐   ┌──────┐    │
                         │ │ System │ ⇒ │ Proc.│    │
                         │ └────────┘   └──────┘    │
                         └───────────────────────────┘
                              (Assignable Units)
```
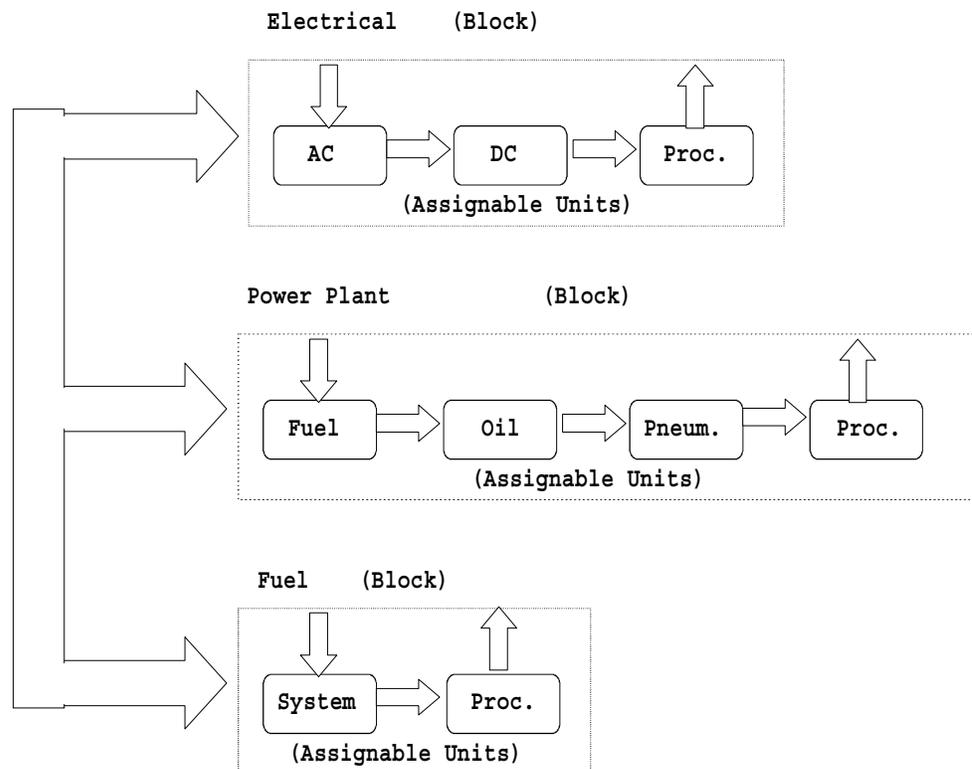
Level 1   The following table is a level 1 description of this example course.

Course Structure Table

Owner	Member	Member	Member	Member
root	Electrical (Block)	Power Plant (Block)	Fuel (Block)	
Electrical (Block)	AC	DC	Electrical Procedures	
Power Plant (Block)	Fuel	Oil	Pneumatics	Power Plant Procedures
Fuel (Block)	System	Fuel Procedures		

Level 2   Adding the following table enables a level two description. There are no separate pre-tests or other features that would require a completion requirements table.

Prerequisites Table

Structure Element	Prerequisite
Electrical (Block)	None
AC	None
DC	AC
Electrical Procedures	DC
Power Plant (Block)	None
Fuel	None
Oil	Fuel
Pneumatics	Oil
Power Plant Procedures	Pneumatics
Fuel (Block)	None
System	None
Fuel Procedures	System

Levels 3a & 3b   This simple course does not use objectives in its description; and does not require any logical statements to better describe its structure. Level 3 descriptions would be no different, and require no additional tables (other than description tables).

6.1   The Course File

Description   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.

File type   Group/Keyword (MS Windows INI)

File name   xxxxxxxx.CRS
                      The extension for this file is CRS. Any OS-legal set of characters may
                      be used for the primary file name.

Groups and keywords

This file contains three groups.

Group Names and Keywords	Function of Keywords
[Course] 　Course_Creator 　Course_ID 　Course_System 　Course_Title 　Level 　Max_Fields_CST * 　Max_Fields_ORT 　Total_AUs 　Total_Blocks * 　Total_Objectives * 　Total_Complex_Obj 　Version	This group contains information that applies to the course as a whole. Some of this data is designed to help in processing the other files that provide more detailed information on the elements in the course and how they are ordered.
[Course_Behavior] 　Max_Normal	Defines lesson behavior defaults expected in the delivery of the course.
[Course_Description]	Textual description of the course.

* These keywords are NOT required for level 1 compliance to the AICC guidelines.

6.1.1   **[Course] Keywords**

Keywords   This list of keywords is in a logical order for their appearance in the
                      file.

	Course_Creator	Name of group that authored the course.
	Course_ID	Identifier for the course
	Course_System	Name of the authoring system used to create the course.
	Course_Title	Common name given to the course.
	Level	The complexity level of the file and the course description contained in the file.
	Max_Fields_CST	Maximum number of fields appearing in the Course Structure Table file.
*	**Max_Fields_ORT**	Maximum number of fields appearing in the Objectives Relationships Table file.
	Total_Aus	Total number of AUs in the course.
	Total_Blocks	Total number of blocks in the course.
*	**Total_Complex_Obj**	Total number of complex objectives defined in the file.
*	**Total_Objectives**	Total number of objectives in the course. This number includes simple and complex objectives.
	Version	The revsion number of the guideline on which the Course Structure is based

* These keywords are NOT required for level 1 compliance to the AICC guidelines.

6.1.1 [Course] Keyword

Course_Creator=	**Definition**	Name of the vendor and/or author of the course.
	Data format	Alphanumeric. All characters, beginning with the first printable character after the equals sign are significant.
	Examples	Course_Creator = Boeing Commercial Airplane Group, \ Customer Services
		Course_Creator = Airbus

Course_ID=	**Description**	See Course_ID keyword on page 82.

Course_System=	**Definition**	Name of the predominant authoring system used to create the course. The authoring system used to create the greatest number of units in the course.
	Data format	Alphanumeric. All characters, beginning with the first printable character after the equals sign are significant.
	Examples	Course_System=Authorware
		Course_system = PCD3
		Course_System=WISE
		Course_System=VACBI
		course_system = AIS II

6.1.1 [Course] Keywords (cont.)

Course_Title=	**Definition**	Common name given to the course. Probably used by the CMI system when identifying course for student.
	Data format	Alphanumeric. All characters, beginning with the first printable character after the equals sign are significant.
	Examples	Course_Title = 747 Flight Crew Training
		Course_Title = Maintaining 747 Avionics

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.
	Usage rules	If **Level** is not defined, level 1 complexity is assumed.
	Format	Alphanumeric characters. Allowed vocabulary is: 1 - Supports level 1 course interchange. May support 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.
	Examples	Level = 3
		Level = 2

6.1.1 [Course] Keywords (cont.)

Max_Fields_CST=

Definition   Identifies the maximum number of fields that are in the
course structure table (any.CST file).

Format   Numeric characters.

Examples   Max_fields_CST=12
; There is at least one block (or the course itself) that
; has 11 members.

Max_Fields_CST = 9

Max_Fields_ORT=

Definition   Identifies the maximum number of fields that are in the
objectives relationships table (any.ORT file).

Format   Numeric characters.

Examples   Max_fields_OBJ=12
; There is at least one element in the left-most column
that
; has 11 members.

Max_Fields_OBJ = 9

6.1.1 [Course] Keywords (cont.)

Total_AUs=	**Definition**	The total number of unique assignable units in the course. This information aids in the processing of information in the file.
		This number does not necessarily represent the largest digit used to identify an AU. If there are 5 lessons in a course, they do not have to be identified as A .001, A.0021, A2, A3, A .003, A.004, and A.004, and A5. AU identifiers do not have to be consecutive. A course with 5 lessons (Total_AUs=5) could have the identifiers A .00008, A.00064, A.00512, 8, A64, A512, A4096, A2768
	Format	Numeric characters.
	Examples	Total_AUs = 3 ; There are three assignable units in the course.
		Total_AUs= 84
		total_aus = 138
Total_Blocks=	**Definition**	The total number of unique blocks in the course. This information aids in the processing of the rest of the data in the file.
		This number does not have to be equal to the largest number used in an extension. Identifier extensions do not have to be consecutive.
	Format	Numeric characters.
	Examples	Total_Blocks = 3 ; There are three blocks in the course.
		Total_Blocks= 84
		total_blocks = 138

6.1.1 [Course] Keywords (cont.)

Total_Complex_ Obj=	**Definition**	The total number of unique complex objectives in the course. This information aids in the processing of the rest of the data in the file.
		This number does not have to be equal to the largest number used in an extension. Identifier extensions do not have to be consecutive.
	Format	Numeric characters.
	Examples	Total_Complex_Obj = 3 ; There are three complex objectives in the course.
		total_complex_obj = 138

Total_Objectives=	**Definition**	The total number of unique objectives in the course. This information aids in the processing of the rest of the data in the file.
		This number does not have to be equal to the largest number used in an extension. Identifier extensions do not have to be consecutive.
	Format	Numeric characters.
	Examples	Total_Objectives = 3 ; There are three objectives in the course.
		total_objectives = 138

6.1.1 [Course] Keywords (cont.)

Version =

Definition	Identifies the ***CMI Guidelines for Interoperability*** document revision number on which the Course Structure data files are based.
	Every time this document is updated, it receives a different revision number. This number appears on the cover page, along with the date of the revision.
Format	Numeric characters with a decimal point.
Examples	Version =1.2
	version = 2.0

6.1.2 **[Course_Behavior] Keywords**

Purpose This group is used to define keywords that affect the behavior of the CMI system.

Max_Normal	The maximum number of assignable units that may be taken for credit and incomplete.

6.1.2 [Course_Behavior] Keywords (cont.)

Max_Normal=	**Definition**	The maximum number of assignable units that may be taken for credit simultaneously. That is, this value indicates how many lessons launched with credit = credit are allowed to be incomplete.
		When this number is exceeded, subsequent launches must be with credit=no_credit. Further, default behavior is to launch all additional lessons in the Browse CMI mode.
	Format	A single integer number. Number must be less than 100.
	Default	If no number is indicated, 1 is assumed. If a number greater than 99 is indicated, then 99 is assumed.
	Examples	Max_Normal=1 ; only 1 lesson being taken for credit can be incomplete. Max_Normal = 5

6.1.3   **[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. The content of this field is determined by the desires of the
                        author.

Format   Freeform text. Carriage returns are implied (explicitly) at the end of
                        each line.

Size   4096 bytes for the entire group, including all lines.

6.2   Assignable Unit File

Description   Information relating to the assignable units (AU) in the course. Each
                         AU has its own record (or row in the table).

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.AU
                         The extension for this file is AU. Any OS-legal set of characters may
                         be used for the primary file name.

Fields   The first record contains the field identifiers. The order in which these
                         field identifiers appear determines the order of the data in subsequent
                         records. Each following record in this file describes a different
                         assignable unit. Each record has the following fields.

<div align="center">

Assignable Unit File: the fields

</div>

System ID	Type	Command Line	File Name
System_ID	Type	Command_Line	File_Name

Continued ➔

Max Score	Mastery Score	Max Time Allowed
Max_Score	Mastery_Score	Max_Time_Allowed

Continued ➔

Time Limit Action	System Vendor	Core Vendor
Time_Limit_Action	System_Vendor	Core_Vendor

Although all field identifiers must be in the file for level 1 compliance, only the following field
values are required:
        System_ID
        Command_Line
        File_Name
        Core_Vendor

Note that Core_Vendor may be a blank field in some cases. However, if it exists, data in this
field must be supported by the CMI system for level 1 compliance.

Example file contents   The three records below are extracted from the beginning of a hypothetical file.

"system_id","type","command_line","Max_Time_Allowed",”time_limit_action”,"file_name"," max_score","mastery_score","system_vendor","core_vendor”

"A11","lesson","APU1 -nuv","00:16:00", ‘‘Exit”,"APU1.EXE",80,80, "APW",""

"A12","test","APU2 -nuv","00:26:00",”E,Message”, "APU2.EXE",100,90, "APW","test = on"

"A13","lesson","ELEC -nuv","00:28:00",”E,N”, "ELEC1.EXE",50,50, "APW",""

6.2 Assignable Unit Fields

System ID	**Description**	See the *System ID* field description on page 163.

Type	**Definition**	Assignable units may be categorized. **Type** identifies a user defined category. These are determined by the designer/developer of the assignable unit.
	Data format	Alphanumeric. Not case sensitive. May contain spaces and commas.
	Examples	"Lesson" "Criterion Test"

Command Line	**Definition**	The string of characters needed to successfully launch an executable program in the DOS environment. Environment variables may be embedded in the command line.
	Data format	Alphanumeric. Not case sensitive. Limited to 255 characters.
	Examples	"APU /UAL /MN" "ELEC3 -nuv3" "%lesloc%ELEC3 -nuv3"

6.2 Assignable Unit Fields (cont.)

File Name	**Definition**	The full identifier of the file containing the most critical content of the assignable unit. (An assignable unit may require several files -- a graphics library file, a digitized audio file, etc.) The name, as listed by the operating system when disk contents are requested.
		Environment variables may be embedded in the file name to indicate a "soft" path.
	Data format	Alphanumeric characters. May be case sensitive, depending on the operating system.
	Usage rules	Should not include an explicit path name, because that path could be invalid in the target system..
	Examples	"APU.EXE" "%lespath%\APU.WIS"

Mastery Score	**Description**	See *Mastery_Score* description under CMI to CBT [Student_Data] keywords, page 94.

6.2 Assignable Unit Fields (cont.)

Max Score	**Definition**	When the student exits a lesson a raw **score** is returned as one of the keywords in the group [CORE]. (See *Lesson to CMI* section 5.2.1) This keyword (Max_Score) allows the CMI system to compute a percent from the student's raw **score**.
		This score may not be the same as the reported as Max Score by the assignable unit. The AU max score is the maximum that the student could have achieved with the interactions that he experienced. This Max Score is the maximum possible when experiencing all interactions in an AU.
	Data format	Decimal number.
	Examples	100
		"6"
		27.31

Max Time Allowed	**Description**	See *Max_Time_Allowed* description under CMI to CBT [Student_Data] keywords, page 95.

Time Limit Action	**Description**	See *Time_Limit_Action* description under CMI to CBT [Student_Data] keywords, page 97.

System Vendor	**Definition**	Authoring system used to create the lesson.
	Data format	Alpha-numeric. Any characters, including spaces, up to end-of-line and carriage return.

6.2 Assignable Unit Fields (cont.)

Core Vendor	**Definition**	Unique information required by the lesson's design. Without this information, a lesson may not execute.
	Data format	Text field. This contains whatever system-unique information is necessary for this lesson to function well. This field is limited to 4096 characters. If more information is required, the field may contain a reference to a separate file with the necessary data.
		To enable the storage of several keywords and their values in this field, embedded carriage returns (<CR>) may be used. When this information is passed to a lesson in an INI (group-keyword) file, a real carriage return is substituted for each "<CR>" symbol.
	Example	The following field in an assignable unit file

"Testmode=on<cr>Special_add=0<cr>Backon=off"

goes into the INI file for the lesson at launch and becomes

```
[Core_Vendor]
Testmode=on
Special_add=0
Backon=off
```

6.3   Descriptor File

Description   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. This file also contains any textual description created for an element in the course. Course elements include
- Assignable Units
- Blocks
- Objectives
- Complex Objectives

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.DES
The extension for this file is DES. Any OS-legal set of characters may be used for the primary file name.

Fields   Each record in this file describes a different element in the course. Each record has the following fields. Their order is determined by the order in which the field titles appear in the first record.

Required fields Although all field titles must be in the file for level 1 compliance, only the following field values are required:
System_ID
Developer_ID
Title

Descriptor File: the fields

System ID (for course element)	Developer ID (for course element)	Title	Description
System_ID	Developer_ID	Title	Description

6.3 Descriptor Fields

Example file contents   The records below are extracted from the beginning of a hypothetical file.

"system_id","developer_id","title","description"
"A1","PP1-2","Power Plant Introduction","An overview of the operation of the primary systems in the Pratt & Whitney PW2037 engine."

"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."

6.3 Descriptor Fields (cont.)

System_ID   **Definition**   System assigned ID. The exporting system for the
                                       course structure, generates a simple ID for every course
                                       element. That ID must appear in this file.

                                       This simple ID has two parts. A letter and a number.

                                       The letter identifies to what category element the record
                                       refers. Possible categories (types) are:
                                       ▪ A -- Assignable Unit
                                       ▪ B -- Block
                                       ▪ J -- Objective or complex objective

                                       The number is a simple integer to distinguish each
                                       unique item in a category.

                    Data   Alphanumeric. Not case sensitive. The first letter is an
                    format   A, B or J. That is followed by an integer number.

                    Examples   "A15"
                                       "J237"
                                       "B1"

6.3 Descriptor Fields (cont.)

Developer_ID	**Definition**	Developer assigned ID. Unique identifier for an assignable unit, block, objective, or complex objective. Used outside of this structure file to refer to a specific element.

Lesson ID and Objective ID are both types of Developer ID.

Note: Because this is a developer-assigned ID, it may be unique only to a specific project. It is possible that multiple developers, at different times and locations, working on different projects can create the same ID. If a course is assembled from disparate lessons, these two lessons could be in the same course. There would be several identical Developer IDs in such circumstances.

This is why the course interchange files includes System IDs. These are created by the CMI system and are guaranteed unique in each course.

Data format   Alpha-numeric string. No internal spaces.

Examples   "APU-747-003"

"747-423-ELEC-001"

6.3 Descriptor Fields (cont.)

Title	**Definition**	Commonly used name for an assignable unit, block, objective, or complex objective. Probably used by CMI system in menu screens where students can select an assignable unit or block, or select to see the status of an objective.
	Data format	Alphanumeric. Not case sensitive. May contain spaces and commas.
	Examples	"Auxiliary Power Unit, Part 1" "Auxiliary Power Unit Start" "Electrical Power, Part 3"

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. The content of this field is determined by the desires of the author.
	Data format	Text. Limited to 4096 characters.

6.4   Course Structure File

Description   This file contains the basic data on the structure of the course. It includes all of the assignable units and blocks in the course. The order in which these appear in the file implies (but does not force) an order for presentation to the student.

Even though the student may have the option of selecting any assignable unit or block, the CMI router will probably list them in the order in which they appear in this file.

If a specific order is required by the developer, that order is specified in the **prerequisites** table.

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.CST
The extension for this file is CST. Any OS-legal set of characters may be used for the primary file name.

Records   Each record in this file describes the members of a course or block, and implies a level in the course hierarchy. The order of the records must be respected both upon import and export to achieve minimum AICC compliance.

Fields   Each record has a variable number of fields, limited by the Max_Fields_CST keyword in the Course file. Each different assignable unit or block that appears in the file must have a unique identifier.

The first field in each record is always the course or block identifier. The course is always identified by the word "root", and the block identifier is always arbitrarily determined by the course generation routine. The block identifier is found in the "system_id" column of the Descriptor File.

Each block identifier will always appear more than once in a file -- the first appearance identifies where the block is in the hierarchy; the second appearance identifies the members of the block. Assignable units may appear more than once.

Usage rules   The first entry in the file is always "root."

Each subsequent entry is a system generated ID. The ID indicates the type of element and that it is a member of the course (root) or block that is identified in the first field.

A block will always appear in the file first as a member of another group (another block or the root). The second appearance of the block usually defines the membership of the block. (In some cases a block may appear in more than a single block in the course, in which case the membership may be described in the third or fourth appearance of the block ID.)

Course Structure Table, v1.1

Block	Members -- Assignable units & other blocks			
Block	Member	Member	Member	Member
root				

6.4.1   **Example 1**

Description   This is a simple course that is described in three ways. The first
                       description is a diagram, the second is a table, and the third is the contents
                       of a Course Structure File.
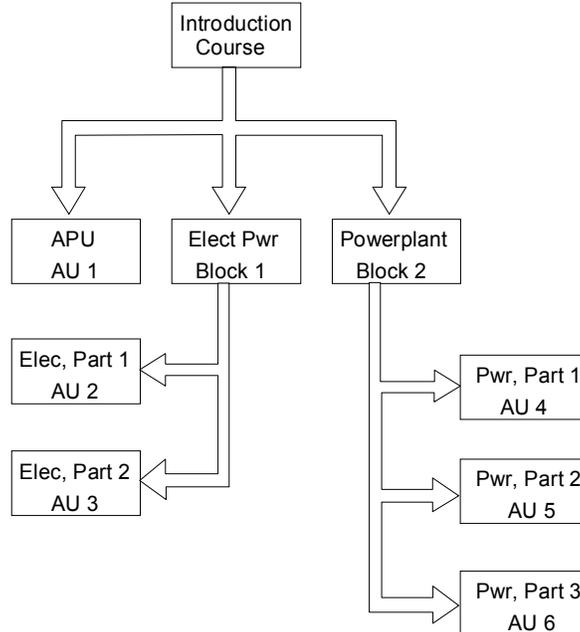
Example table   The table below reflects the diagram above. Each course element in this
                       table uses the "Developer ID" -- the unique identifier assigned by the ISD
                       organization during development of the course.

<div align="center">

Table for *Introduction Course*

Root	AU 1	Block 1	Block 2
Block 1	AU 2	AU 3	
Block 2	AU 4	AU 5	AU 6

</div>

**Example file   The records below reflect the table and diagram above. Each ID is a
contents**   "System ID" -- ID assigned by the system that generated the files for the
                       export of this course.

```
"block","member","member","member"
 "root","A1","B1","B2"
"B1","A2","A3",""
"B2","A4","A5","A6"
```

6.4.2   **Example 2**

Description   This is a simple course that was described in the introduction to this chapter. It is described here in three ways. The first description is a diagram, the second is a table, and the third is the contents of two key files: the Descriptor File and the Course Structure File.
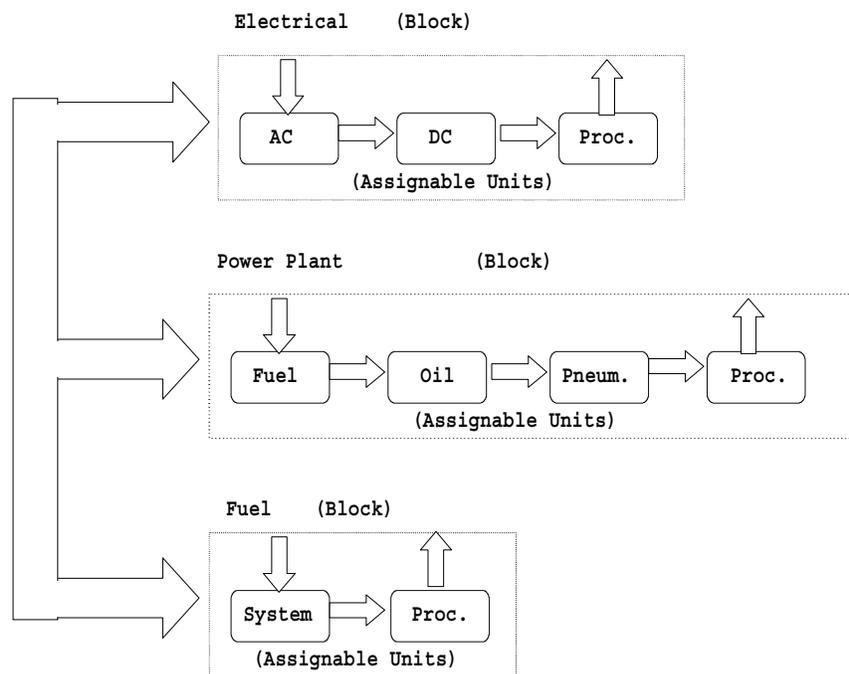
Example table   The table below reflects the diagram above. Because each entry in the file must be a unique identifier, the table also includes the exporting-system generated ID (System ID).

Table for Example Course

Root	Electrical B1	Power Plant B2	Fuel B3	
Electrical B1	AC A1	DC A2	Procedures A3	
Power Plant B2	Fuel A4	Oil A5	Pneumatics A6	Procedures A7
Fuel B3	System A8	Procedures A9		

Example file contents   The records below represent the contents of the Descriptor File, and reflect the table and diagram above.

Filename: example.DES

```
"system_id","developer_id","title","description"
"A1",        ”AC-01”,        "AC Electrical",,
"A2",        ”DC-01”,        "DC Electrical",,
"A3",        ”EP-01”,        "Electrical Procedures",,
"A4",        ”PP-01”,        "Power Plant Fuel",,
"A5",        ”PP-02”,        "Power Plant Oil",,
"A6",        ”PP-03”,        "Power Plant Pneumatics",,
"A7",        ”PP-04”,        "Power Plant Procedures",,
"A8",        ”FS-01”,        "Fuel System",,
"A9",        ”FS-02”,        "Fuel Procedures",,
“B1”,        ”BEP-01”,       ”Electrical Power”,,
“B2”,        ”BPP-01”,       ”Power Plant”,,
“B3”,        ”BFS-01”,       ”Fuel”,,
```

Example file contents   The records below represent the Course Structure File, and reflect the table and diagram above.

Filename: example.CST

```
"block","member","member","member","member"
"root", "B1",    "B2",    "B3",
"B1",    "A1",    "A2",    "A3",
"B2",    "A4",    "A5",    "A6",     "A7"
"B3",    "A8",    "A9",         ,
```

6.5   Objectives Relationships File

Description   Objectives have complex and variable relationships to other elements of a course. For instance, a lesson may cover several objectives. A single objective may require mastery of several lessons. Other objectives may require the mastery of many sub-objectives.

The Objectives Relationship file is able to define all of these relationships. However, not all CMI systems depend upon objectives for routing decisions. Not all objectives are critical to the functioning of a CMI system.

This file is optional for course descriptions that do not have objectives as prerequisites for assigning lessons. All objectives that are part of a prerequisite are required in this file.

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.ORT
The extension for this file is ORT. Any OS-legal set of characters (or less) may be used for the primary file name.

Records   Each record in this file describes the objectives that are included in a given course element. (assignable unit, block, complex objective).

Fields   The left-most field contains the exporting-system generated ID of a course element, all the fields to the right list the system IDs of objectives that are in that element.

Example table   In this table, B13 is a block with 3 objectives requiring mastery to consider the block complete.

A48 is a lesson that contains two objectives which must be mastered before the lesson is complete.

J16 is a complex objective requiring three other objectives be complete before it is considered complete.

B14 is a block which requires the mastery of 5 objectives before it is complete.

Objectives Relationships Table, v1.1

Course Element	Member objectives				
Course_Element	Member	Member	Member	Member	Member
B13	J23	J24	J25		
A48	J27	J28			
J16	J93	J94	J95		
B14	J16	J26	J29	J30	J31

Example file contents   The records below are extracted from the beginning of a hypothetical file. They reflect the table above. The maximum number of fields required in this file is 6.

```
"course_element","member","member","member","member","member"
"B13","J23","J24","J25",,
"A48","J27","J28",,,
"J16","J93","J94","J95",,
"B14","J16","J26","J29","J30","J31"
```

6.6   Prerequisites File

Description   Sometimes it may be desirable to prevent a student from entering a lesson 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.

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.PRE
The extension for this file is PRE. Any OS-legal set of characters may be used for the primary file name.

Records   Each record allows a single prerequisite (Level 2) or list of prerequisites (Level 3b) to be defined for a block or AU.

Fields   The first record identifies the order of the fields with the field names: Structure_Element, and Prerequisite.

The system generated ID is in the **structure_element** field. The **prerequisite** field is an expression (See the section on Logic Statements) that identifies the course elements that determine whether a student can begin the block or AU.

Prerequisites File

Level 2

Structure Element (Block or AU)	Prerequisite (Block or AU)
Structure_Element	Prerequisite
System ID	System ID
System ID	System ID

Prerequisites File

	Structure Element (Block or AU)	Prerequisite Logic Statement (Blk, AU or Obj)
Level 3a	Structure_Element	Prerequisite
	System ID	System ID & System ID
	System ID	System ID | System ID
	System ID	System ID
	System ID	System ID & (System ID | System ID)

Usage rules

When there is no prerequisite defined for an AU or block, the student may select that course element at any time.

When there is no prerequisite for an AU that is part of a block, and the block does have prerequisites, then that AU may be taken anytime the block prerequisites are met.

When an AU or block does not appear in the file, it is assumed to be an AU or block with no prerequisites. (i.e. The above rules apply.)

Example file contents

The records below are extracted from the beginning of a hypothetical file.

```
structure_element, prerequisite
"B13","B12 & A14"
"A48","B12 | B11"
"A49","A48"
"A50","B12 & (A15 | A16)"
```

6.6.1 **Assignable Unit and Objective Status**

Assignable Unit (lesson) status

Prerequisites are a listing of those course elements that a student has passed or completed. **Completed** is a status. Lesson status is often determined within the lesson by the logic designed into it.

There are six possible statuses for each lesson.

- Passed
- Completed
- Browsed
- Failed
- Not attempted
- Incomplete

In any logic statement, a structure element may be made equal to any of these statuses. However, if not explicitly identified these five statuses are resolved into two statuses: complete or incomplete as follows:

- ◼ Complete = true
  - Passed
  - Completed
- ◼ Incomplete = false
  - Browsed
  - Failed
  - Not attempted
  - Incomplete

In a prerequisite record the following statement

"A4","A3"

Means that the status of lesson 3 must be **Complete or Pass** before the student can begin lesson 4. Explicitly, this would be expressed as

"A4","A3=P | A3=C"

In a prerequisite record the following statement

"A4","A1 = P & A2=P & A3 = P"

Means that the status of lessons (assignable units) 1, 2, and 3 must be **Pass** before the student can begin lesson 4. If any one of those lessons was completed instead of passed, that would not satisfy the prerequisite for lesson A4.

Objective status   Objective statuses can also be used to determine if a student has met the prerequisites for a lesson or a block. Objective statuses are determined by the lesson. The six possible statuses sent to the CMI system are resolved into complete or incomplete the same way they are for lessons.

In an objectives oriented course, the following prerequisite record could appear:

"A4","J1 & J2 & J3"

Meaning that the student must complete or pass three objectives before entering assignable unit number 4.

6.6.2   **Logic Statements**

Logic statement   A logic statement is a list of course elements (block, assignable unit, objective) with their status (Complete, Incomplete, etc.) separated by logic operators (&, |, ~). A special logic statement is the single word "never". This is used to prevent a student from ever entering the lesson in the mode (normal, review, browse) for which the record is applicable.

Logic operators   A logic operator describes how course elements are to be combined to determine whether a logical prerequisite is complete or incomplete. This table lists the available logic operators.

Operator Meaning	Symbol
and	&
or		
not	~
equals	=
group or set	{ }
separator for set members	,
complete X number out of a set	X*{ }
evaluate first	( )

Definitions   When evaluating course elements in a logic statement, and status is not explicitly stated, one of two states is possible: complete or incomplete.[5] These correspond to the traditional logical values of true and false. The following operators can be used to create a logical statement with course elements.

---

[5]   These Boolean statuses are defined in the section titled "Assignable Unit and Objective Status" on page 175.

and	All elements separated by an & must be compete for the expression to be evaluated as complete. **A34 & A36 & A38** Assignable units number 34, 36, and 38 must all be completed or passed for the group to be considered complete.
or	If any of the elements separated by an | are passed the expression is considered true. **A34=P | A36=P | A38=P** If any one of the lessons, 34, 36, or 38, are passed then the group is considered complete.
not	An operator that returns incomplete (false) if the following element or expression is complete, and returns complete (true) if the following element or expression is incomplete (false). **A34, ~A35** The student may enter unit 34 as long as unit 35 has not been completed (that is, the status of A35 must be Browsed, Incomplete, Failed, or Not attempted). If assignable unit 35 is complete, the student may not enter unit 34.
equals	Evaluates true when elements on both sides of the sign have the same value.
set	A list of course elements separated by commas and surrounded by curly brackets -- { }. A set differs from a block, in that the set is defined only for purposes of the prerequisite (or completion requirements) file. A set has no effect on the structure of the course. **{A34, A36, A37, A39}** Assignable units 34, 36, 37, and 39 are part of a set.

separator	The comma is used to separate the members of a set. Each member of the set can be evaluated as a Boolean element – complete (true) or incomplete. (false) **{A34, A36, A37, A39}** Assignable units 34, 36, 37, and 39 are each separated by a comma in this set.
X*	X is an integer number. This operator means that X or more members of the set that follows must be complete for the expression to be complete (true). **"A38", "3*{A34, A36, A37, A39}"** Any three or more of the following units – 34, 36, 37, 39 -- must be complete before the student can enter unit 38.
evaluate 1st	The expression inside the parenthesis ( ) must be evaluated before combining its results with other parts of the logical statement. Parentheses may be nested.[6] "A39", "A34 & A35 | A36" In this statement, completing A36 all by itself enables the student to enter A39. **"A39", "A34 & (A35 | A36)"** Adding the parenthesis, makes it necessary to complete at least two units (A36 all by itself is no longer enough) to enter unit A39.

Examples   These records are from prerequisites files.

---

[6] Operator precedence is the same as in the C programming language – including the use of parenthesis.

Level 3a

> **A31,A23 & A28[7]**

Means that before the student can begin Assignable Unit #31 in the normal mode, he must complete units 23 and 28. This record

Level 3a

> **"A31","3*{A23 , A25 , A26 , A28 , A29}"[8]**

Means that before he begins unit 31 in the normal mode, the student must complete at least three of the five lessons listed in the parentheses.

Level 3a

> **"A31","3*{A23 , (A25 & A26) , A28 , A29}"**

In this case units 25 and 26 together comprise one member of the set. Therefore, the student may have to complete 4 units in order to enter lesson (assignable unit) number 31. For instance, having completed A23, A25, and A28, he would NOT be able to enter lesson 31.

Level 3a

> **"B31","~J31"**

Means that he may begin any unit in block 31 if he has not completed objective 31 (that is, if Objective 31 has a status of Incomplete, Fail, or Not Attempted the student may begin Block 31). After completing objective 31, he may not enter block 31.

Level 3a

> **"B31","~(J31=F)"**

---

[7] Quotation marks are not required in a comma delimited data file unless there are commas in one of the fields. However, quotation marks are commonly used around all fields for the sake of consistency.

[8] In this record, quotation marks are required. Otherwise, the commas separating set members would be interpreted as field separators.

Means that he may begin any unit in block 31 if he has not failed objective 31 (that is, if Objective 31 has a status of Fail, the student may not begin Block 31). After failing objective 31, he may not enter block 31.

Level 3a

> **A15, A14 & ~J15**

Means that before he begins unit 15 the student must complete unit 14 and not have completed objective 15. If he has mastered objective 15 he may not enter lesson 15. If he has not completed lesson 14, he may not enter lesson 15.

Level 3a

> **A24, ~(J13 & J14 & J15)**

Means that lesson 24 may be started if any of the following objectives are not complete: 13, 14, and 15. Completing one or two of these objectives does not prevent the student from entering lesson 24. Completing all three objectives will prevent entry into 24.

6.7   Completion Requirements File

File justification   While lesson and objective status is frequently determined within
the lesson by the logic designed into it, this is not always true. For
instance, there may be an assignable unit designed to pre-test the
student. By demonstrating mastery of some objectives in this pre-
test, the student may get credit for passing parts of a lesson – or even
a complete lesson – without ever having seen it

In other words, the CMI system may sometimes determine the status
of a element by factors outside the element itself. Similarly block
and complex objective status is defined in terms of other structure
elements. Therefore, block and complex objective status must be
determined by the CMI system.

One additional function that this file may perform, is to enable the
course designer to tell the CMI system to launch the next lesson in a
sequence automatically. This ability allows the seamless integration
of several assignable units.

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.

Default status

Block	Block status is determined by the status of all of its members. Unless specially defined in a completion requirements file, a block is considered complete when all members of the block are complete.	
Complex objective	A complex objective is considered complete when all of its members are complete.	
Lesson	Lesson status is determined by the lesson when the student leaves the lesson. Additionally, a pass or fail status can be determined by the CMI by comparing the lesson's score with the lesson's mastery score.	
Simple objective	A simple objective's status is determined when a lesson sends information to the CMI to indicate its status.	

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.CMP
                     The extension for this file is CMP. Any OS-legal set of characters may
                     be used for the primary file name.

Record rules   Each record in this file defines how the CMI system may determine the
                     status of an assignable unit, block or objective.

                     There may be an unlimited number of logic statements to determine the
                     status of each lesson. For instance, just to define Pass, Fail, Complete,
                     and Incomplete for a single lesson would require 4 completion records.

                     The order of these records is significant. To determine the status of a
                     lesson, the CMI system should evaluate each statement relating to the
                     lesson in the same order in which it appears in this file. The first
                     statement to evaluate True determines the status of the lesson.

                     It is important that the order of these records be respected by the CMI
                     system during the import and export of the Completion Requirements
                     file.

Fields   Each record has three or four parts (fields). These fields may be in any
                     order. The first record identifies the order with the field titles:
                     Structure_element, Requirement, Result and Next. The next field is
                     optional for any record.

                     STRUCTURE_ELEMENT: This field contains the exporting-system
                     generated ID of an assignable unit, block or objective.

                     REQUIREMENT: A logic statement that enables a true or false decision
                     to be made by the CMI system. The logic notation is the same as
                     described in Section 6.6.3.

                     RESULT: This field indicates the status of the element when the
                     requirement statement is found TRUE. This does NOT mean that if a
                     PASS is in the result field and the requirement statement is false, then
                     FAIL must be assumed. If the requirement statement evaluates as
                     FALSE, the status of the element is determined by other factors – either
                     an additional completion record or default.

                     NEXT: This field identifies a lesson to be launched automatically
                     without pause by the CMI system. This lesson is to be launched
                     whenever the result in the RESULT field is achieved. When this field is
                     blank, a typical behavior of the CMI would be to return the student to
                     the course menu.

RETURN. This field indicates to which assignable unit the CMI shall return after completion of the NEXT AU. This field takes priority over the NEXT field that may be in the completion requirements file for the AU from which the CMI is returning.

Level differences   Level 2 Completion Requirements files are limited to a single element in the logic statement. Level 3a files may have complex logical expressions as well as a single element.

Completion Requirements File
Level 2

Course Element (Block Objective or AU)	Completion Logic Statement (Blk, AU or Obj)	Status Result if True	Next AU if Result Achieved	Return to after Next
Structure_ Element	Requirement	Result	Next	Return
System ID	System ID	passed	Sys ID	Sys ID
System ID	System ID=passed	passed		
System ID	System ID	complete	Sys ID	

Completion Requirements File
Level 3A

Course Element (Block Objective or AU)	Completion Logic Statement (Blk, AU or Obj)	Status Result if True	Next AU if Result Achieved	Return to after Next
Structure_ Element	Requirement	Result	Next	Return
System ID	System ID=P & System ID=P	Pass	Sys ID	Sys ID
System ID	2*{System ID, System ID, System ID}	Pass		
System ID	System ID	Comp	Sys ID	
System ID	System ID & (System ID | System ID)	Comp		

Example records   The records below are extracted from the beginning of a hypothetical file.

Level 2

```
Structure_Element, Requirement, Result, Return
A4, A1, Passed
```

If the assignable unit A1 gets a status of completed or passed, then the assignable unit A4 is given the status of pass. If the student later enters A4, the CMI must pass a **lesson_status** of P,A.

Level 2

```
Structure_Element, Requirement, Result, Next, Return
A4, A4=P, Passed, A5
```

If the assignable unit A4 gets a status of passed, then the assignable unit A5 is launched automatically by the CMI system. After finishing A5 the student continues to any AU determined by the prerequisites file or the system defaults.

Level 2

```
Structure_Element, Requirement, Result, Next, Return
A4, A4=F, Passed, A5, A4
```

If the assignable unit A4 gets a status of failed, then the assignable unit A5 is launched automatically by the CMI system. After the student finishes A5 (any status) he is automatically returned to A4.

Level 2

```
Structure_Element, Requirement, Result, Next, Return
A4, A1, Pass
A4, A2, Pass
A4, A3, Pass
```

If the assignable unit A1 or A2 or A3 gets a status of complete or pass, then the assignable unit A4 is given the status of pass. Because the records are evaluated sequentially and the first to evaluate True determines status, if A1 is Passed then it makes no difference whether A2 and A3 are passed or failed or not attempted.

Level 2


```
Structure_Element, Requirement, Result, Next, Return
A4, A4, Passed, A5
A17, A17= failed, failed, A4, A17
```
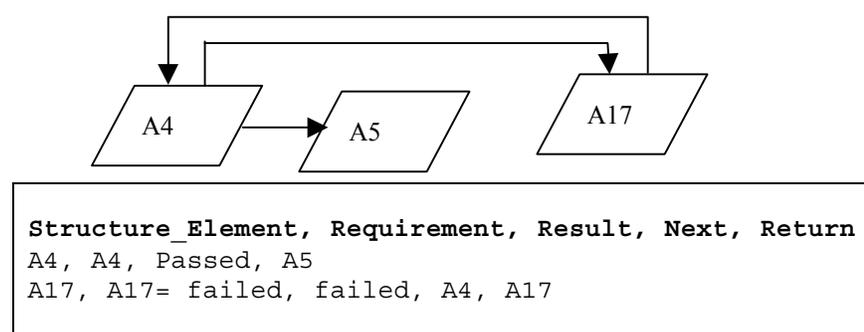
If A4 is passed or completed then the student is automatically assigned to A5. If the student fails A17, he is automatically assigned back to A4. When he finishes A4 (even if his status is "passed") he must return to A17.

Level 2


```
Structure_Element, Requirement, Result, Next, Return
A4, A4, Passed, A5
A17, A17, passed, A18
A17, A17= failed, failed, A4, A18
```
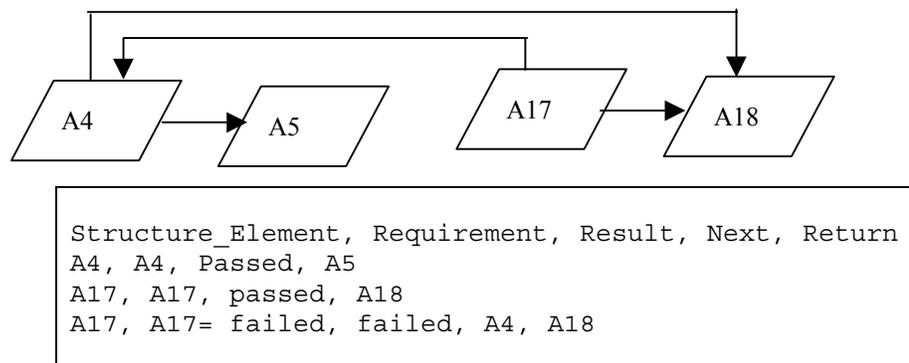
This is similar to the previous example. The only difference is that 1) if the student passes A17 he is automatically assigned to A18, and 2) if the student fails A17, and finishes A4, he must return to A18 (not A17 as in the previous example).

Level 2

```
Structure_Element, Requirement, Result
B4, A3, Pass
```
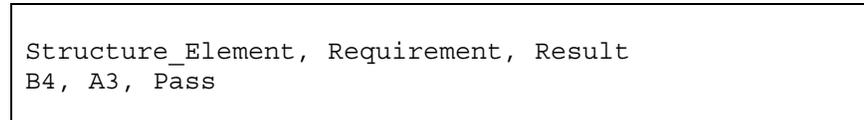
If the assignable unit A3 gets a status of complete or pass, then the entire Block 4 (and all of its members) is given the status of pass.

Level 3a

```
"Structure_Element", "Requirement", "Result", "Next"
"B13","J23=P & J24=P & J25=P & J 26=P", "Completed"
```
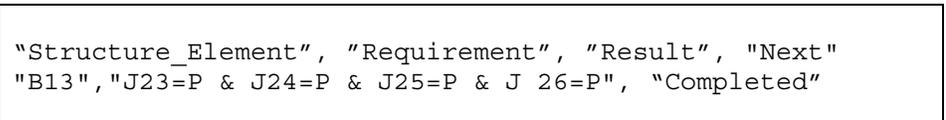
Block 13 is not dependent on completion of its members (assignable units and other blocks) but rather on passing of objectives.

Level 3a

```
"Structure_Element", "Requirement", "Result"
"B8", "A14 |  A15 | A16","Completed"
```

Block 8 is considered complete when any one of these three assignable units is complete or passed.

Level 3a

```
"Structure_Element", "Requirement", "Result"
"B21", "3*{A36 | A37 | A38 | A39 | A40}", "Completed"
```

This tells the CMI system that the block is complete when any 3 of these 5 assignable units is complete or passed. An example of when this might be useful would be in a block with 5 exercises. The course designer wants the student to perform at least 3 of the five exercises. That is what this logic statement is indicating.

Level 3a

```
"Structure_Element", "Requirement", "Result" , "Next"
"B13","A8=P | A9=P | A10=P | A11=P", "Incomplete"
"B13"," A8=P & A9=P & A10=P & A11=P ", "Completed"
```

Notice that in this case, Block 13 will never be considered Complete. The first statement will always evaluate True before the second. And the first statement to evaluate True determines the status of the course element.

Level 3a

```
"Structure_Element", "Requirement", "Result" , "Next"
"B13","A8=P & A9=P & A10=P & A11=P ", "Completed"
"B13","A8=P | A9=P | A10=P | A11=P", "Incomplete"
```

This corrects the problem in the example above. Now, as soon as the student has passed a single lesson in the block the block status will be changed from Not attempted to Incomplete. When the student has passed all of the lessons, the status of the block will change to Complete because the first statement will evaluate True and the CMI system will never get to the second statement to re-evaluate the Block status.

Level 2

```
"Structure_Element", "Requirement", "Result", "Next"
"A14", "A14=F","Fail","A36"
```

If the student fails lesson A14, he is immediately forced to go to the remedial lesson A36.

6.8   Structure Considerations

This section contains a series of examples that may clarify many of the principles that have been discussed in this chapter.

Example 1   Five lessons to be taken in sequence from 1 to 5. Student has to select the next lesson.

**Prerequisite file
EXAMPLE1.PRE**
```
"Structure_Element","Prerequisite"
"A2","A1"
"A3","A2"
"A4","A3"
"A5","A4"
```

Example 2   Five lessons. One must be taken first. Two, three, four and five can be taken in any order.

**Prerequisite file
EXAMPLE2.PRE**
```
"structure_element","prerequisite"
"A2","A1"
"A3","A1"
"A4","A1"
"A5","A1"
```

Example 3   In this block (B3) there are 6 assignable units. A mastery test (A14), 4 lessons, and a remedial unit (A18). Each lesson has an objective associated with it. The objective just happens to have the same identifying digit as the lesson. Lesson A15 has objective J15, lesson A16 has Objective J16, etc.

Upon entering the block the student has the option of taking lesson 15 or the test (A14). The lessons must be taken in order. Objectives must be specified explicitly if they are in the entry criterion.
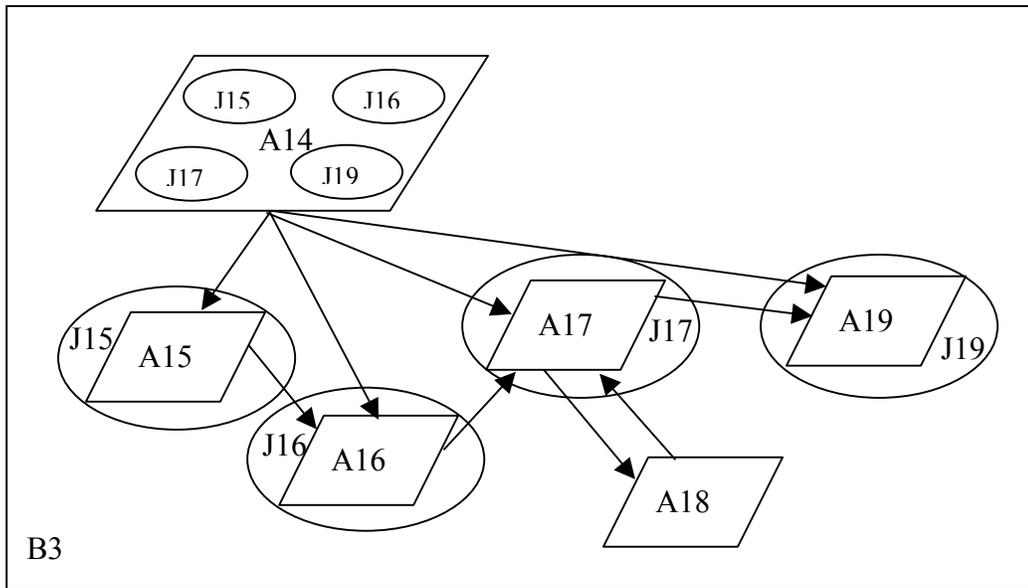
Objectives relationship file EXAMPLE3.ORT

```
Structure_Element,Member,Member,Member,Member
A14, J15, J16, J17, J19
J15, A15
J16, A16
J17, A17
J19, A19
```

Completion requirements file EXAMPLE3.CMP

```
Structure_Element,Requirement,Result,Next,Return
B3,J15 & J16 & J17 & J19, passed
A17, A17=failed, failed, A18, A17
J15, A15=passed, passed
J16, A16=passed, passed
J17, A17=passed, passed
J19, A19=passed, passed
```

Objectives   The objectives relationship table shows that A15 is a member of
                    objective J15, and A16 is a member of J16, and so forth. When this is
                    the case, the completion requirement entries shown above for the
                    objectives are superfluous. By default, an objective is considered
                    passed when all of its members are passed.

Prerequisite file
EXAMPLE3.PRE

```
"Structure_Element","Prerequisite"
"A16","J15"
"A17","J15 & J16"
"A19","J15 & J16 & J17"
```

Remediation   The Completion Requirements file forces the student into the
                    remediation unit if he fails objective 17. After finishing the remedial
                    unit, A18, he is forced to return to A17.

7.0 LESSON EVALUATION DATA

Single student covered

Lesson evaluation data is contained in several files. The file names are passed to the lesson. If the file already exists, the lesson appends the data. If the file does not exist, the file is created and the data deposited.

The CMI system therefore, determines if multiple lessons append data to a single file, or if each lesson creates a different file. Normally, it is expected that a single file will contain the data accumulated over multiple lessons for a single student.

With this information, analysis tools and CMI systems are able to assemble information on multiple lessons, multiple uses of the same lesson, and multiple students.

Raw data

The analysis of the information is not the subject of these guidelines. What is covered here is essentially raw data.

Chapter contents   The following is a table of files and their fields.

Section	Files	Fields
7.1	**Comments** • Name: any.any (name defined by CMI) • Contents: Comments made by student while taking a lesson. • Type: Table (Comma Delimited)	Course ID Student ID Lesson ID Date & Time Location Comment
7.2	**Interactions** • Name: any.any (name defined by CMI) • Contents: Detailed information on each interaction measured as the student takes a lesson. • Type: Table (Comma Delimited)	Course ID Student ID Lesson ID Date & Time Interaction ID Objective ID Type interaction Correct response Student response Result Weighting Latency
7.3	**Objectives Status** • Name: any.any (name defined by CMI) • Contents: Repeat of information that is in the CBT to CMI file under the group [Objectives_Status]. • Type: Table (Comma Delimited)	Course ID Student ID Lesson ID Date & Time Objective ID Local ID Score Status Mastery time
7.4	**Path** • Name: any.any (name defined by CMI) • Contents: A description of the path the student took through the lesson. (What he experienced first, second, third, and so forth.) • Type: Table (Comma Delimited)	Course ID Student ID Lesson ID Date & Time Element location Status Time in element

7.1   Comments File

Description   This is a journal file that contains freeform feedback from the student. It is a duplicate of the [Comments] group that is passed to the CMI system in the CBT-to-CMI file.

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.xxx
The file name is determined by the CMI system and provided to the lesson in the CMI-to-CBT file group [Evaluation] with the keyword Comments_file=.

Fields   Each record in this file describes either a whole or part of a comment. Because a field is limited to 255 characters, some comments may have to extend over several records.

Records   The first record must contain the AICC field identifier for each field in the file. The field identifiers are not case sensitive. Subsequent records have the field information in the same order as the identifiers in the first record.

Comments File: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Continued ➜

Location	Comment
Location	Comment

Example file contents
The four records below are extracted from a hypothetical file and represent a single comment.

```
"course_id","student_id","lesson_id","date","time","location",
"comment"

"MD80FT-2","ua36","APU1","1994/01/15","00:14:23","f3", "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."

"MD80FT-2","ua36","APU1","1994/01/15","00:14:36","f16", "Why
did you change colors? I was just getting used to purple on
orange."
```

7.1 Comment Fields

Course ID	**Description**	This ID is supplied by the CMI system in the CMI-to-CBT file under the group name [Evaluation] with the keyword Course_ID=.
		See *Course_ID* description on page 82.
	Field ID	Course_ID

Student_ID=	**Description**	See *Student_ID* description on page 64.
	Field ID	Student_ID

Lesson ID	**Definition**	Alphanumeric identifier for the lesson.
		This is unique to, and inherent in each lesson. It is normally determined by the course/lesson designer at the time of creation.
		See also Developer ID on page 164.
	Field ID	Lesson_ID
	Data format	Alphanumeric. All characters, inside the quotation marks are significant.
	Examples	"FT747-4-302"
		"A330.MT.246"

7.1 Comment Fields (cont.)

Date	**Definition**	The calendar day on which the record is made
	Data format	YYYY/MM/DD Four digits for year, two for month, and two for day. Separated by slashes. Single digit months and days must be zero padded.
	Examples	"1993/01/15" "2001/12/03"

.

Time	**Definition**	Identification of when the comment is made. This represents the time of day with a 24 hour clock.
	Field ID	Time
	Usage rules	Three numbers, separated by colons, are always required, even if only hours and minutes are represented.
	Data format	Same as *Time=* keyword on page 76.

Location	**Definition**	Indication of where in the lesson the comment is made.
	Field ID **Data format**	Location This may be provided in any way the designer wanted to identify the different elements in a lesson -- by frame, by interaction, by page, etc.
	Examples	"frame 13" "13" "f6" "Interaction 12"

7.1 Comment Fields (cont.)

Comment	**Definition**	The recorded statement of a student.
	Field ID	Comment
	Data format	Text enclosed by quotation marks.
		If the student places double quotes in his comment ("), the system must change them to single quotes (') before attempting to store them in a comma delimited file.
		If it is desired to place multiple lines in a single record (total characters must be 255 or less) it is possible to embed a carriage return in the text to replace the carriage return and line feed characters. This is done by using <CR> (less-than sign, the letter **c**, the letter **r**, and the greater-than sign)
	Examples	"No<cr>way<cr>Jose."
		"I still thing the anser is B."

7.2 Interactions File

Interaction definition

In this context, an interaction is a recognized and recordable input or group of inputs from the student to the computer. All of the items in this group are related to a recognized and recorded input from the student (or lesson user.)

If the program does not recognize that the student is attempting to input, then the input is not an interaction in this limited use of the word. For instance, a program may present a question on the screen, and wait for the student to touch or click on the correct answer. The program may not be designed to look at keyboard inputs during this question. If the student inputs the correct answer by typing a letter, on the keyboard, it is not an interaction by this definition, because it is not recognized and therefore not recorded by the computer.

Notice that by this definition, a single interaction can involve more than one discrete measurable input. A single interaction may involve several inputs. For instance, assume there is a piano-like keyboard connected to the computer and the student is asked to play a chord. The interaction involves pressing and holding three keys that make up the chord. Therefore, each note in the chord may be considered one input. with the three keys together making up the entire input to the single interaction.

Single interaction or many?

In grouping inputs and determining what constitutes a single interaction as opposed to a series of interactions, the designer of the lesson is the final arbiter.

However, in grouping inputs and deciding whether multiple inputs are to the same interaction or separate interactions, consider time. If all inputs may be provides simultaneously, or rapidly in any order, they are probably inputs to the same interaction. If they are going to be provided in a sequence, and especially if the sequence is important, they are probably inputs to separate interactions.

Description

This group provides extensive information on student interactions which the designer of the lesson wants to preserve and analyze. Normally, the interactions preserved are student responses to a question

File type

Table (Comma-delimited ASCII)

File name   xxxxxxxx.xxx

The file name is determined by the CMI system and provided to the lesson in the CMI-to-CBT file group [Evaluation] with the keyword Interactions_file=.

Fields   The first record in the file contains the field identifiers. The field identifiers can be in any order; however the order in which the identifiers appear, establishes the order for the fields that occur in each subsequent record. Each subsequent record describes a single interaction.

Interactions File: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Interaction ID	Objective ID	Type Interaction	Correct Response	Response Value
Interaction_ID	Objective_ID	Type_Interaction	Correct_Response	Response_Value

Continued ➔

Student Response	Result	Weighting	Latency
Student_Response	Result	Weighting	Latency

Example file contents   The four records below are extracted from a hypothetical file and represent a single comment.

```
"course_id","student_id","lesson_id","date","time","interaction_id","objecti
ve_id","type_interaction","correct_response","student_response","result","we
ighting","latency"
"A340ft-2","jqh085","APU1","1994/01/15","15:14:23",37,ft1016,C,A,C,W,,
00:00:3
"A340ft-2","wam016","APU1","1994/01/15","15:14:23",38,ft2223,t,t,t,,,
00:00:01
"A340ft-2","dag085","APU1","1994/01/15","15:14:23",39,ft1134,C,B,B,C,,
00:00:02
"A340ft-2","trd018","APU1","1994/01/15","15:14:23",40,ft1156,C,C,C,C,,
00:00:04
```

7.2 Interactions Fields

Course ID	**Description**	See the *Course_ID* keyword on page 82
	Field ID	Course_ID
Student ID	**Description**	See the *Student_ID* keyword on page 64
	Field ID	Student_ID
Lesson ID	**Description**	See the *Lesson ID* field under Comment file on page 195
Date	**Description**	See the *Date* field on page 196
Time	**Definition**	The chronological point at which a student may begin interacting.
	Format	See *Time* field under Comments file on page 76.
Interaction ID	**Definition**	Unique identifier for an interaction.
		Identifier created by the lesson designer/developer.
	Field ID	Interaction_ID
	Format	Alpha-numeric string. May include internal spaces.
	Examples	"1" "apu2" "fuel2-04"

7.2 Interactions Fields (cont.)

Objective ID	**Definition**	Unique identifier for an objective, or complex objective. Used outside of this file to refer to a specific objective. An Objective_ID is one kind of Developer_ID.
		See also Developer ID on page 164.
	Field ID	Objective_ID
	Format	See the *J-ID.2* keyword on page 86 for data format and example names.

Type interaction	**Definition**	Indication of which category of interaction is recorded.
		The type of interaction determines how the interaction response should be interpreted. Seven possible question types are defined below. They are not meant to be limiting. There are other types of questions. However, if one of these seven types is used, these are the identifiers that match those types.
	Field ID	Type_Interaction
	Data Format	A keyword is used to describe each interaction. Only the first letter is significant and capitalization is ignored.

Type of Question	Keyword	Description
True/False	true-false	A question with only two possible responses.
Multiple Choice	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.
Fill in the Blank	fill-in	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).
Matching	matching	A question with one or two sets of items. Two or more of the members of these sets are related. Answering the question requires finding and matching related members.

Simple Performance	performance	A performance question is in some ways similar to a multiple choice question. However, instead of selecting a written answer, the student must perform a task or action. This task or action when input to the computer is translated and stored as one or more alpha-numeric codes. These are some performance questions: Using the panel on screen, input the correct latitude. Perform the next action required to set the fuel panel for take-off.
Sequencing	sequencing	In a sequencing question, the student is required to identify a logical order for the members of a 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.
Likert	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 ➢ FROM Strongly agree TO Strongly disagree ➢ FROM Way too much TO Way too little ➢ FROM Understand completely TO Do not understand at all
Numeric	numeric	Simple number with or without a decimal point required answering the question. Correct answer may be a single number within a range of numbers.

Examples   "true-false"
"multiple choice" -- this is type **matching**
                               -- only the first letter is significant
"C"
"F"
"Performance"

7.2 Interactions Fields (cont.)

Correct response	**Definition**	Description of possible responses to the interaction. There may be more than one correct response, and some responses may be more correct than others.
	Field ID	Correct_Response
	Data format	Normally, responses are considered case insensitive. However, in instances where case is important in judging a response, the correct response is preceded by <case>.

If there is more than one response described, a semi-colon separates the responses.

If the correct response requires multiple inputs, then the required inputs are surrounded by a curly bracket -- { }.

- **True/ False**

  A single character or numeral. Legal characters are 0,1,t,and f. 0 corresponds to false. If the response is a complete word (i.e. "true") only the first letter is significant.

- **Choice:**

  One or more characters separated by a comma.. Integers (0..9), letters (a..z) or both may be used. Each possible response is limited to a single character.   If there are more than 26 possibilities, then a performance type response must be used.

  Normally, there is only one correct response to a CHOICE question. But two other possibilities exist.

7.2 Interactions Fields (cont.)

1) There is more than one correct response. The answer is considered correct when any of these alternatives is selected. However, one of these responses may be more correct than the others.

2) Sometimes, a question may have two or three correct alternatives, and the student must select all of these alternatives to get the question correct. In these cases, the correct responses are grouped by surrounding them with curly brackets -- { }.

- **Fill-in:**

   An alphanumeric string. Spaces after the = are ignored up to the first printable character. After the first printable character, spaces are significant.

- **Numeric**

   A single number. The number may or may not have a decimal.

- **Likert -- likert** (or L or l)

   There is no incorrect response for a Likert question. The **Correct_Response** field may be left blank.

- **Matching:**

   Pairs of identifiers separated by a period. Each matching possibility consists of a *source* and a *target* (or *stem* and *alternative*). Each source and target possibility must have a unique identifier. An identifier can be an integer number or a letter.

7.2 Interactions Fields (cont.)

Two integer numbers or letters separated by a period represent the source and target (or stem and alternative). If there is more than one matching pair is considered correct in the interaction, the pairs are separated by a comma.

If more than one matching pair is required to consider the interaction response correct, the pairs must be separated by commas and surrounded by curly brackets.

- **Performance:**

Alpha-numeric field limited to 255 characters.

There are three things that distinguish a **Performance** response from a **Choice** response.

1) Because a choice response is limited to a single character there can be no more than 36 alternatives (a...z and 0...9). With a performance response, there can be thousands of alternatives.

2) In a choice response the order of the answers, or the sequencing of the inputs is irrelevant. In a performance response, sequence is significant and may be used in judging the response.

3) In a performance response, a range of correctness may be specified. For instance, a student could be asked to adjust the throttle for cruise. A throttle angle of 34 to 38 degrees could be an acceptable response.

7.2 Interactions Fields (cont.)

When a PERFORMANCE question has a range of correct responses, the range is expressed as two numbers separated by two dashes or hyphens. There must be two dashes to make sure that it cannot be confused with a minus sign preceding a number.

When a PERFORMANCE question has a sequence of correct responses, each element of the sequence is separated by a comma.

Correct answers may be a sequence of ranges.

When sequence is not important, the elements of the response must be surrounded by a curly bracket.

- **Sequencing**

    In a performance question, the order in which the actions are performed, or the order in which the elements are identified is important.

    In a sequencing question, the elements may be identified in any order. The final positioning of the elements is used to determine correctness, not the order in which they were re sequenced.

Examples   **Choice**   "b; d"
                      Example 1   -- The response is considered correct when the student
                                          -- selects either "b" or "d".

                      Choice   "{b,d}"
                      Example 2   -- The response is considered correct only when the
                                          -- student has selected both "b" and "d" in his response
                                          -- to the question

                      Matching   "1.c; 2.b; 3.a; 4.d"
                      example 1   -- answer is considered correct if any one of these four
                                          -- matches is made by the student

7.2 Interactions Fields (cont.)

Matching   "{1.c,2.b,3.a,4.d}"
example 2   -- answer is considered correct only if all of these four
                  -- matches is made by the student

Matching   "{3.4,1.6,5.2}"
example 3   -- answer is considered correct only if all of these three
                  -- matches is made by the student

Matching   "{a.e, d.g, c.f, b.h};{a.h, d.g, c.f, b.h}"
example 4   -- Student must make four matches to have answer
                  -- judged correct. However, he may match "a" to
                  -- either "e" or "h" and still have a correct match.

Mixed   "<case> Washington"
examples

                  "2300-2400"
                  --   This can only be legal as a response to a fill-in
                  --   question. If it represents a range on a performance
                  --   question, there must be two hyphens.
                  "2300 -- 2400"

                  "b,c,e,a,d"
                  -- A sequencing question. This is a single response.

                  "23291"
                  --   Notice that for some types of questions, large
                  --   numbers cannot have commas in them. 23,291
                  --   could be interpreted as two correct alternative
                  --   responses.

7.2 Interactions Fields (cont.)

**Rule for source
and target ID's**

Note that the sources and targets cannot use the same identifiers. That is, stem items cannot be identified as 1, 2, 3, 4, and alternative items also identified as 1, 2, 3, and 4. This rule is related to student response recording.

Assume for a moment that the stem items are labeled 1, 2, 3, and the alternatives are also labeled 1, 2, and 3. Further assume, stem 1 must be paired to alternative 3. If during data analysis a 1.3 pair occurs, is it correct or incorrect?

The computer records student inputs in the order in which they occur. The student could have selected the alternative item 1 first and matched it with the stem item 3. This would make the response incorrect.

However, if the student had selected the stem item 1 first, and then the alternative 3, the response would be correct.

Using the same identifiers for both sources and targets could make it impossible to determine whether the student chose item 1 from the stem items or the alternative items. Therefore, in this example, the stem items should be labeled 1, 2, 3, and the alternative items 4, 5, and 6, or the stem items labeled 1, 2, and 3, with the alternatives labeled a, b, and c. In any case, all identifiers should be unique.

Why not force all targets to be numeric and all stems to be alphabetic. Assume there is a group of items in a circle. The student is to match related pairs in this large circle. It would be very inconvenient to have to label the members of the circle (clockwise) 1, 2, a, 3, b, c, d, 4, e, 5. Far better to be able to label the circle (clockwise) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

7.2 Interactions Fields (cont.)

Student response	**Definition**	A description of the computer-measurable action of a student in a situation where his action is of interest to an analyst.
	Field ID	Student_Response
	Data format	The following **Response** data formats are described under **Correct response** on page 203.

- True-false
- Multiple choice (choice)
- Fill in the blank (fill-in)
- Numeric
- Matching
- Simple performance (performance)
- Sequencing
- Likert

Matching examples

"1.2"
"{1.c,2.b,3.a,4.d}"
"{3.4,1.6,5.2}"
"{a.e, d.g, c.f, b.h}"
"{12.2, 11.3, 10.11}"

Mixed examples

"1.2"
"1.2,1.3"
"Washington"
"C"
"b,c,e"
"23291"
-- Notice that for some types of questions, large
-- numbers cannot have commas in them. 23,291 is
-- interpreted as two alternative responses.

7.2 Interactions Fields (cont.)

Result	**Definition**	Judgment of the acceptability of the student response.

Field ID   Result

Data format   Only the first character is significant. Possible arguments include correct or wrong. If all responses are acceptable (correct) for a given ID, this keyword is omitted.

- Correct, c, or C.
- Wrong, w, or W.
- Unanticipated response, u, or U.
- Neutral, n or N.
- a number (may be a decimal)

For multiple judgments on a complex response (such as a matching response) commas separate the judgments. Another judgment for a complex question, like a matching question, could be a number like .75, indicating 3 out of 4 matches were made.

Usage rules   When the **correct response** field is not blank, this keyword becomes redundant, since the result can be computed from comparing **response** to **correct response**. However, both the **correct response** field and the **result** field may be filled in. There is no rule against redundancy in this file.

Result Examples
"c"
"c,c,w,c,c"
"unanticipated response, correct, correct, wrong, c"
"neutral"
"Correct"

7.2 Interactions Fields (cont.)

Weighting	**Definition**	Interactions vary in importance. The weighting is a factor which is used to identify the relative importance of one interaction compared to another. For instance, if the first interaction has a weight of 15 and the second interaction has a weight of 25, then any combined score that reflects weighting would be more influenced by the second interaction.
		If all interactions are equal in importance, then each interaction has the same weight. For instance each could have a weight of 1, or each could have a weight of 37. The final weighted score from the lesson where all interactions are equal to 1 is the same score as that lesson if all of its interactions are equal to 37.
		A weight of 0 indicates that the interaction should not be counted in the weighted final score.
		This refers to weighting of the question, not the student response.
	Field ID	Weighting
	Data format	A single floating point number. The decimal point is optional and does not have to appear in every **Weighting.**
	Usage rules	An interaction may have a weight, and similarly, individual actions or responses inside a complex interaction may have a weight. The WEIGHTING factor applies to any element of the interaction with the same four-digit extension.
	Examples	.66 1.25 5

7.2 Interactions Fields (cont.)

Latency	**Definition**	The time from the presentation of the stimulus to the completion of the measurable response.
	Field ID	Latency
	Data format	hh:mm:ss (Hours:Minutes:Seconds.)
	Usage rules	If latency is recorded, there can be a latency figure for each response. For multiple responses in an interaction, the latencies must be separated by a comma, and in the same order as the responses.
	Examples	"00:00:23" -- the student required 23 seconds to respond "00:23:00" -- The student required 23 minutes to respond "00:00:03" -- The student required 3 seconds to respond "00:01:13

Correct response	Student response	Latency
{a.h, b.f, c.i, d.k, e.f}	a.h, c.i, b.f, d.k, e.g	00:01:01

Partial record

"{a.h,b.f,c.i,d.k,e.f}","a.h,c.i,b.f, d.k,e.g","","","00:01:01"

7.3   Objectives Status

Definition   An objective identifier and an indication of what the student has done
on previous attempts on the lesson. The student can pass, fail, or not
attempt an objective. These objectives are those associated with the
current launching lesson, not all the objectives in the course/curriculum.

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.xxx
The file name is determined by the CMI system and provided to the
lesson in the CMI-to-CBT file group [Evaluation] with the keyword
Objectives_status_file=.

Fields   The first record contains the field IDs. Each additional record in this
file describes a single objective.

Objectives Status: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Continued ➜

Objective ID	Score	Status	Mastery Time
Objective_ID	Score	Status	Mastery_Time

Example file contents   The example below is extracted from a hypothetical file and represent a single record.

```
"COURSE_ID","STUDENT_ID","LESSON_ID","DATE","TIME","OBJECTIVE_I
D","SCORE","STATUS","MASTERY_TIME"

"MD80-2","STU1009","APU1","1994/01/15","10:14:23","APU1684",3,,
"passed","00:02:37"
```

7.3 Objectives Status Fields

| **Course ID** | **Description** | See *Course ID* field under Comment file on page 82 |
| | **Field ID** | Course_ID |

| **Student ID** | **Description** | See *Student_ID* keyword on page 64 |
| | **Field ID** | Student_ID |

| **Lesson ID** | **Description** | See *Lesson ID* field under Comment file on page 195 |
| | **Field ID** | Lesson_ID |

| **Date** | **Description** | See *Date* field under Comments file on page 196 |
| | **Field ID** | Date |

Time	**Definition**	The chronological point at which the student begins work on the objective.
	Description	See *Time* field under Comments file on page 76
	Field ID	Time

| **Objective ID** | **Definition** | See Objective ID field description on page 201. |
| | **Field ID** | Objective_ID |

| **Score** | **Description** | See **J_Score.1** on page 87. |
| | **Field ID** | Score |

7.3 Objectives Status Fields (cont.)

Status	**Definition**	See **J_Status.4** on page 89.
	Field ID	Status

Mastery time	**Definition**	The time required by the student to master (or fail) the objective.
	Description	See *Time=* on page 76.

7.4   Path File

Description   This file allows an analysis of what path the student took through a lesson. It enables the analyst to determine when the student asked for help, when he selected alternative branches, if he selected optional instruction, and the order in which he proceeded through the lesson.

Lesson elements   In order to track a path through a lesson, the lesson must arbitrarily be divided into some finite number of elements. There can be as many or few as the designer feels are desirable. This group enables the analyst to determine how many of the lesson elements were entered by the student, the order in which the student experienced the elements, and the time he spent in each element.

An element may be thought of as a unit of instruction. It may be defined as the content between any two points to which a designer may wish a student to be able to jump. An element begins at the point where a student would be when jumping to that element, and ends at the point where the student would be if jumping to the next element.

Because the number of elements in a lesson is at the discretion of the designer, there can be thousands. For instance, if a lesson is designed to teach a student how to play a song, each chord, or even each note could be considered an element. The lesson could be designed to allow the student to return to any point in the song to practice a section of the piece.

To allow for a large number of elements in a lesson, a four digit extension is used for each of the keywords associated with the element.

File type   Table (Comma-delimited ASCII)

File name   xxxxxxxx.xxx
The file name is determined by the CMI system and provided to the lesson in the CMI-to-CBT file group [Evaluation] with the keyword Path_File=.

Fields   The first record contains the field identifier names. Each subsequent
                      record in this file describes a single element in the student's path. The
                      order in which the records appear in the file, describes the path through
                      a lesson.

Path File: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Continued ➜

Element Location	Status	Why Left	Time in Element
Element_Locati on	Status	Why_Left	Time_in_Elem ent

Example   The order in which the student progresses through the lesson is reflected
                      by a series of locations. The order of the location records reflects the
                      order in which the location was accessed. The **Element Location** data
                      or value indicates the ID of the element that was entered by the student.

                      For instance, assume there is a lesson with 6 elements. The element
                      locations are labeled A, B, C, D, E, and F. The student takes the path
                      described here, proceeding from point 1 to point 2, back to 3, and so
                      forth.

Lesson with 6 elements
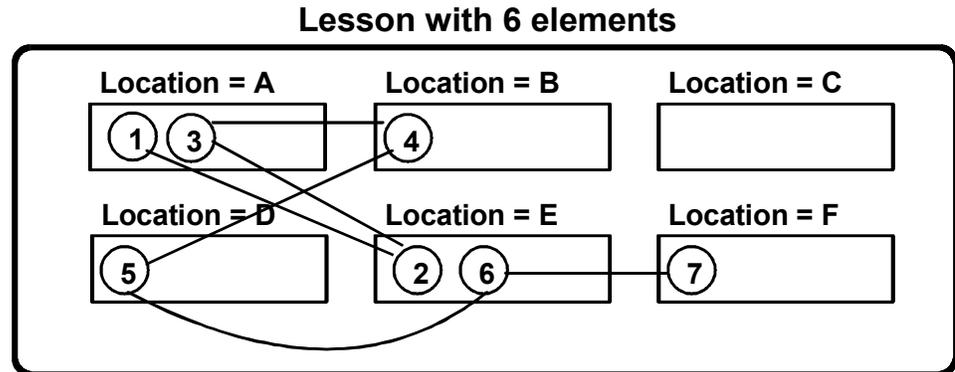
This progression through the lesson would result in a file with the following records to describe the path.

"course_id","student_id","lesson_id","date","time","element_location","status","why_left","time_in_element"

"course6","stu2310","first1","1998/06/05","14:10:31","A","P","S","00:00:24"

"course6","stu2310","first1","1998/06/05","14:10:55","E","P","S","00:01:06"

"course6","stu2310","first1","1998/06/05","14:12:01","A","I","L","00:02:24"

"course6","stu2310","first1","1998/06/05","14:13:25","B","P","S","00:00:54"

"course6","stu2310","first1","1998/06/05","14:14:19","D","P","L","00:02:40"

"course6","stu2310","first1","1998/06/05","14:16:59","E","P","S","00:03:03"

"course6","stu2310","first1","1998/06/05","14:20:02","F","P","E","00:02:12"

7.4 Path Fields

| Course ID | Description | See *Course ID* field under Comment file on page 82 |
| | Field ID | Course_ID |

| Student ID | Description | See *Student_ID* keyword on page 64 |
| | Field ID | Student_ID |

| Lesson ID | Description | See *Lesson ID* field under Comment file on page 195 |
| | Field ID | Lesson_ID |

| Date | Description | See *Date* field under Comments file on page 196 |
| | Field ID | Date |

Time	Definition	The chronological point at which the student enters the lesson element.
	Field ID	Time
	Data Format	See *Time* field on page 76.

7.4 Path Fields (cont.)

Element location	**Definition**	Identification of where in a lesson the student is/was. The element is identified by a location ID.
		A single lesson element can have a large number of interactions associated with it, or only a single interaction associated with it. The number of interactions per element is at the designer's discretion.
	Field ID	Element_Location
	Data format	Alpha-numeric string.
	Usage rules	An element may be entered multiple times, resulting in multiple records with the same location.
		Students may also jump out of a lesson to a location in another lesson, then return to the base lesson. This is reflected in the **lesson ID** field as well as the **element location** field.
	Examples	"1" "apu02"

7.4 Path Fields (cont.)

Status	**Definition**	A record of the student's relationship to an element each time he leaves that element. The element is identified by the **Location** field.
	Field ID	Status
	Data format	Status is described with a single character or an integer number. If a word or several letters appear as an argument, only the first character is significant. The following statuses are possible:

Status	Symbol
Passed	P
Complete	C
Incomplete	I
Not attempted	N
Failed	F
Score	integer number

(Notice that "not attempted" is subtly different than "not entered." If a student does not enter an element, there is no record.)

Examples "P"
"pass"

7.4 Path Fields (cont.)

Why Left	**Definition**	This allows a record to be kept indicating why a student departed an element in the lesson. There are four possibilities that may be recorded:
	Move from one element to another	Student selected: The student pressed NEXT or selected some option which resulted in his leaving the element. Lesson directed: The logic of the lesson moved a student out of this element to some other element in the course.
	Leave lesson entirely	Exit by student: A complete departure from this lesson. For instance the student may have selected LOGOUT, or BREAK or a time out may have occurred. Directed departure: The lesson has forced the student out of it. An example might occur when the assignable unit time limit is exceeded.
	Field ID	Why_Left
	Data format	Alpha-numeric string or character.
	Usage rules	Only the first character is significant. Case is not significant. S or s = student selected L or l = lesson directed E or e = exit D or d = directed departure
	Examples	"Why_Left" "S" "student selected" "s"

7.4 Path Fields (cont.)

Time in Element	**Definition**	How long the student spent in the element of the lesson identified in the record.
	Field ID	Time_in_Element
	Description	For data format and examples see *Time=* keyword on page 76.

8.0 GROUP AND KEY WORD SUMMARY

Description These tables list all of the groups, key words, and fields in this document. In many cases the same data can be found in several files. An effort has been made to be sure that when the same data appears in more than one file, it uses the same keyword.

8.1   CMI/Lesson Communication Files

Description   Two files are required for CMI/Lesson Communications. One for the CMI-to-CBT Lesson communication. The other is for the CBT Lesson-to-CMI communication.

CMI-to-CBT	Pg	CBT-to-CMI	Pg
[Core] Student_ID Student_Name Output_File Lesson_Location Credit Lesson_Mode Lesson_Status Path Score Time	63	[Core] Lesson_Location Lesson_Status Score Time	114
[Core_Lesson] data is undefined and may be unique to each lesson	77	[Core_Lesson] data is undefined and may be unique to each lesson	114
[Core_Vendor] data is undefined and may be unique to each vendor	117		
[Comments] no key words <delimited>	79	[Comments] no key words <delimited>	118

CMI-to-CBT (cont.)	Pg	CBT-to-CMI (cont.)	Pg
[Evaluation]	81		
Course_ID			
Comments_File			
Interactions_File			
Objectives_Status_File			
Path_File			
Performance_File			
[Objectives_Status]	85	[Objectives_Status]	120
J_ID.1		J_ID.1	
J_Score.1		J_Score.1	
J_Status.1		J_Status.1	
[Student_Data]	91	[Student_Data]	122
Attempt_Number		Tries_During_Lesson	
Lesson_Status.1		Try_Score.1	
Mastery_Score		Try_Status.1	
Max_Time_Allowed		Try_Time.1	
Score.1			
Time_Limit_Action			

CMI-to-CBT (cont.)	Pg	CBT-to-CMI (cont.)	Pg
[Student_Demographics]	98		
City			
Class			
Company			
Country			
Experience			
Familiar_Name			
Instructor_Name			
Job_Title			
Native_Language			
State			
Street_Address			
Telephone			
Years_Experience			
[Student_Preferences]	103	[Student_Preferences]	126
Audio		Audio	
Language		Language	
Lesson_Type		Lesson_Type	
Speed		Speed	
Text		Text	
Text_Color		Text_Color	
Text_Location		Text_Location	
Text_Size		Text_Size	
Video		Video	
Window.1		Window.1	

8.2 Course Structure Files

Description
There are a variable number of files required for describing a course structure. The number of files depends on the level of complexity of the course description.

The comma-delimited data files all contain the required field keyword names in the first record.

In the Course Structure Table, the Objectives Relationships Table, the Prerequisites File, and the Completion Requirements File there is no meaningful data in the first record. However, becuase consistency may make the parser easier to construct, the first record contains keywords like the other files.

Course File	Pg
[Course]	143
Course_Creator	
Course_ID	
Course_System	
Course_Title	
Level	
Max_Fields_CST	
Max_Fields_ORT	
Total_AUs	
Total_Blocks	
Total_Complex_Obj	
Total_Objectives	
Version	
[Course_Behavior]	152
Max_Normal	
[Course_Description]	154

Assignable Unit File: the fields

System ID	Type	Command Line	File Name
System_ID	Type	Command_Line	File_Name

Continued ➜

Max Score	Mastery Score	Max Time Allowed
Max_Score	Mastery_Score	Max_Time_Allowed

Continued ➜

Time Limit Action	System Vendor	Core Vendor
Time_Limit_Action	System_Vendor	Core_Vendor

Descriptor File: the fields

System ID (for course element)	Developer ID (for course element)	Title	Description
System_ID	Developer_ID	Title	Description

Course Structure Table

Block	Members -- Assignable units & other blocks			
Block	Member	Member	Member	Member
Root	System ID	System ID	System ID	
System ID	System ID	System ID	System ID	System ID
System ID	System ID	System ID		

Objectives Relationships Table

Course Element	Members -- Assignable units, blocks, and objectives			
Structure_Element	Member	Member	Member	Member
System ID	System ID	System ID	System ID	System ID
System ID	System ID	System ID		
System ID	System ID	System ID	System ID	
System ID	System ID	System ID	System ID	System ID
System ID	System ID	System ID		

Prerequisites File

Level 2

Structure Element (Block or AU)	Prerequisite (Block or AU)
Structure_Element	Prerequisite
System ID	System ID
System ID	System ID
System ID	System ID

Completion Requirements File

Level 2

Structure Element (Block Objective or AU)	Completion Logic Statement (Blk, AU or Obj)	Status Result if True	Next
Structure_Element	Requirement	Result	Next
System ID	System ID	Pass	
System ID	System ID	Pass	Sys ID
System ID	System ID	Comp	

Prerequisites File

Level 3a

Structure Element (Block or AU)	Prerequisite Logic Statement (Blk, AU)
Structure_Element	Prerequisite
System ID	System ID & System ID
System ID	System ID | System ID
System ID	System ID
System ID	System ID & (System ID | System ID)

Completion Requirements File

Level 3a & 3b

Course Element (Block Objective or AU)	Completion Logic Statement (Blk, AU or Obj)	Status Result if True	Next
Structure_Element	Requirement	Result	Next
System ID	System ID=P & System ID=P	Pass	
System ID	2*{System ID, System ID, System ID}	Pass	Sys ID
System ID	System ID	Comp	Sys ID
System ID	System ID & (System ID | System ID)	Pass	

Level 3a & 3b

Prerequisites File

Structure Element (Block or AU)	Prerequisite Logic Statement (Blk, AU or Obj)
Structure_Element	Prerequisite
System ID	System ID & System ID
System ID	System ID | System ID
System ID	System ID
System ID	System ID & (System ID | System ID)

8.3 Lesson Evaluation Files

Description All of these files are optional. Up to five of them may be required to store all of the information desired from a CBT lesson.

Comments File: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Continued ➜

Location	Comment
Location	Comment

Interactions File: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Interaction ID	Objective ID	Type Interaction	Correct Response
Interaction_ID	Objective_ID	Type_Interaction	Correct_Response

Continued ➜

Student Response	Result	Weighting	Latency
Student_Response	Result	Weighting	Latency

Objectives Status: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Continued ➔

Objective ID	Score	Status	Mastery Time
Objective_ID	Score	Status	Mastery_Time

Path File: the fields

Course ID	Student ID	Lesson ID	Date	Time
Course_ID	Student_ID	Lesson_ID	Date	Time

Continued ➔

Element Location	Status	Why Left	Time in Element
Element_Location	Status	Why_Left	Time_in_Element

Appendix A: HTTP-based CMI Protocol

A.1   Introduction

Purpose   This appendix defines how Hyper-Text Transfer Protocol (HTTP) is to be used as an optional additional means of launch, control, and data transport for AICC/CMI systems and AICC/CBT Assignable Units. (This will allow AICC/CMI systems to deliver CBT using the World-Wide Web.)

Scope   This appendix describes:

- The rationale for adding an HTTP-based method AICC/CMI communication
- The HTTP message format for AICC CMI/CBT (Assignable Unit) communication
- The CBT Assignable Unit launch and control mechanisms for HTTP-based AICC CMI systems and CBT Assignable Units
- The differences between the file-based method and HTTP-based methods of CMI/CBT Assignable Unit communication.
- The differences between the file-based method and HTTP-based methods of CMI launch and control of CBT Assignable Units.
- Required and optional features for HTTP-based AICC/CMI Protocol (HACP)

Rationale   HTTP protocol was specifically selected as a transport mechanism for AICC data for the following reasons:

- HTTP Web Browsers and Web Servers are widely used for training delivery.
- HTTP is a hardware platform independent protocol.
- Most internet security firewalls allow HTTP request/response messages passage (as opposed to other internet protocols like TCP/IP Sockets or IIOP)

HTTP Compatibility	HTTP clients and Servers used to deliver AICC/CMI systems and CBT must comply with HTTP/1.0.
	Please refer to RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0. for more information.

A.2   Overview

A.2.1   File-Based (Local) Launch and Control

The file-based (aka LAN-based) method for launching a CBT Assignable Unit (AU) is a simple "synchronous" launch. The CMI is a "Router" program that uses the operating system to launch another program (i.e. creating a new process). The CMI launches the Assignable Unit and "waits" until that Assignable Unit completes execution. When that happens, the CMI resumes execution processing the output from the Assignable Unit and displaying the next assignment to the student user.

This process assumes two things:
1.   Both the CMI and the CBT (AU) programs are in files somewhere on the local file system (either a volume provided by a LAN fileserver or a local disk drive.)
2.   Both the CMI and the CBT (AU) programs are local processes running on the student's workstation

A.2.2   HTTP-Based Launch and Control

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 protocol, client and server programs maybe running on the same computer or on different computers at different locations.

Some portions of the CMI run as part of the Web Server and other portions (The student User interface) run as part of the Web Browser.

A.2.3   **Assignable Unit Launching Sequence**

In general, a Web-based CBT launch sequence is as follows:

1. Student Selects a CBT Assignable Unit (AU) to launch from the CMI's user's interface (Menu)
2. The CMI pushes a URI (Uniform Resource Identifier) Location (Complete with startup parameters) to the HTTP client
3. The HTTP Client Requests the CBT Assignable Unit from the HTTP server.
4. The HTTP server program copies program and data to the HTTP Client program
5. The CBT Assignable Unit grabs the URI parameters from the HTTP client upon startup and initiates a "session" with the CMI system.
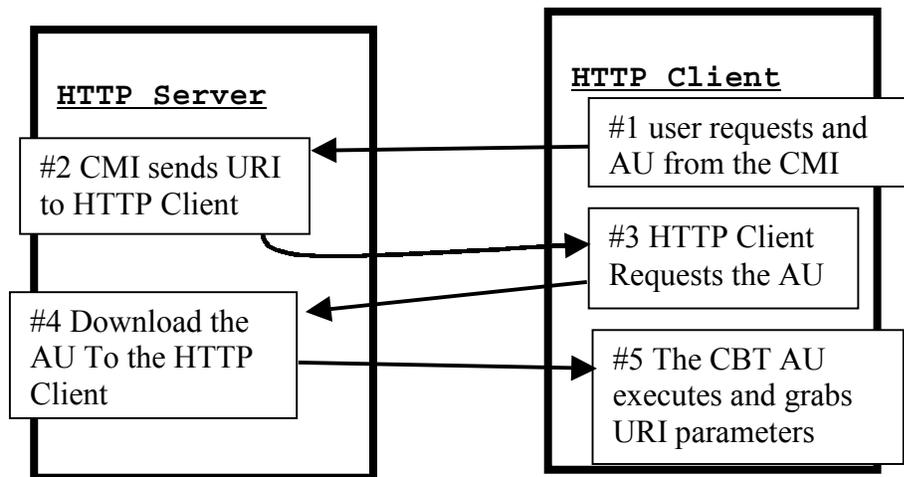
A.2.4   **CBT/CMI Communication Session**

The CBT and the CMI have a client/server relationship where the CMI is the Server and the CBT Assignable Unit is the client. Each request that the CBT makes of the CMI is made by calling the CGI URL (Uniform Resource Locator) using the POST method. (See Section A.5) The CGI URL is specified in the URI launch parameters

A typical communication session between a CMI and CBT Assignable Unit is as follows:

1. The CBT Assignable Unit spawns a separate HTTP session
2. The CBT Assignable Unit sends a message requesting startup information from the CMI System. (This is the information found in PARAM.CMI when files are used for communication.)
3. Just before the end of the Assignable Unit session, the CBT AU sends student performance and lesson status data to the CMI system.
4. When the student exits the CBT Assignable Unit, the AU sends an "end session" message to the CMI.
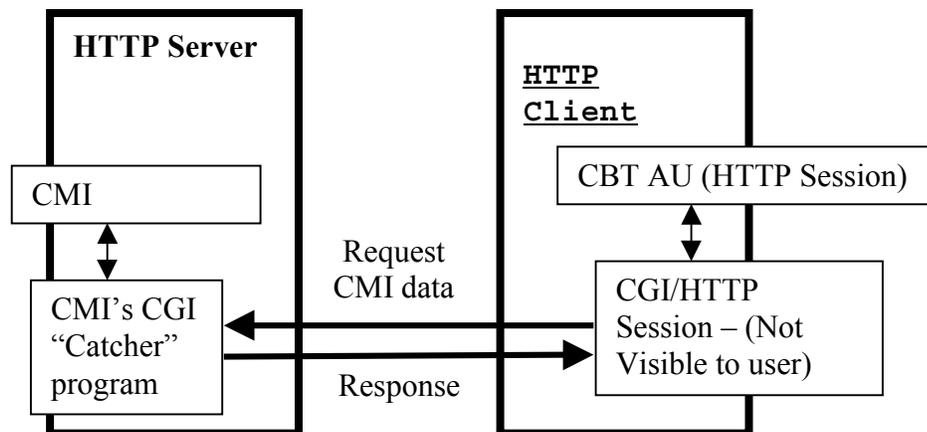
A.3   Differences between HTTP-Based and File-based Methods

There are differences in all three major parts of the CMI Guidelines.
These are discussed in the following three sections:

- Communication Differences
- Course Structure Interchange Differences
- Lesson Evaluation Differences

A.3.1   Communication Differences

Communication differences

The primary differences in HTTP-based vs. file-based AICC CBT/CMI communication are as follows:

1.   CBT Assignable Unit is launched via HTTP (Described in Section A.2.3.)
2.   The input parameter (PARAM.CMI) file, output parameter file, and all files specified in the Lesson Evaluation (Chapter 7) are replaced with corresponding messages (to transfer the data that would have been written to files).
3.   The PATH Keyword in [CORE] is not used in the input parameters. (PATH is found in the PARAM.CMI file)
4.   The CBT Assignable Unit sends an "Assignable Unit Complete" message to the CMI upon exiting.

The commands required to initiate these communications are as follows

Command	Function
GetParam	Reads param.cmi data (See 5.1)
PutParam	Writes ouput data (See 5.2)
PutComments	Writes Comments data (See 7.1)
PutPath	Writes Path data (See 7.4)
PutInteractions	Writes Interactions data (See 7.2)
PutObjectives	Writes Objectives data (See 7.3)
PutPerformance	Writes Performance data (See 5.1.5)
ExitAU	Notifies the CMI that CBT Assignable Unit has exited

A.3.2   Course Structure Interchange Differences

Course structure interchange differences

The only difference in the course structure interchange is related to the fact that the CMI system needs more information in order to launch an Assignable Unit over the Web. This information is placed in the Assignable Unit File (Described in Section 6.2).

Fields

The File_Name field definition requires amplification, and two additional fields are necessary in this file. The first new field describes the launch parameters – in a URL-encoded format. The title of the field is Web_Launch.

The second new field is optional. It provides an Assignable Unit password. The field name is AU_Password

Assignable Unit File: the fields

System ID	Type	Command Line	File Name
System_ID	Type	Command_Line	File_Name

Continued ➜

Max Score	Mastery Score	Max Time Allowed
Max_Score	Mastery_Score	Max_Time_Allowed

Continued ➜

Time Limit Action	System Vendor	Core Vendor
Time_Limit_Action	System_Vendor	Core_Vendor

Continued ➜

Web Launch Parameters	Assignable Unit Password
Web_Launch	AU_Password

Requirements   Although all field identifiers must be in the file for level 1 compliance, only the following field values are required:

    System_ID
    Command_Line or Web_Launch
    File_Name
Core_Vendor

File Name   **Definition**   The full identifier of the file containing the most critical content of the assignable unit. (An assignable unit may require several files -- a graphics library file, a digitized audio file, etc.) The name, as listed by the operating system when disk contents are requested.

The purpose of this field is to enable the CMI to locate the primary file needed to launch an AU.

The AU filename location is installation specific. It is the course vendor's responsibility to either have an automated installation process or 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.

In cases where the AU's use file-based communication, it will be necessary to include a file path so that the CMI can locate the primary AU file on the local file system (on a LAN or local disk drive) for launching purposes.

In other cases where the AU may reside on a remote web server, a URL is necessary for the CMI system to reference the AU.

Data format   Alphanumeric characters. May be case sensitive, depending on the operating system.

Examples   "APU.EXE"
"C:\LESSONS\APU.WIS"
"http://somedomain.org/cbt/apu.html"

Web Launch Parameters	**Definition**	Lesson specific launch parameters. The string of characters that needs to be appended to the file name and CMI-generated (required) parameters, in order to successfully launch an Assignable Unit in the World Wide Web environment.
		The launch parameters for a web-based AU.
		This data is in the "query" portion of the URL command line after the "?" separator. (See section A4).
		There are two parameters required by the AICC. The "Web Launch Parameters" are any additional parameters required by individual assignable units.
	Data format	Alphanumeric. The values may be case sensitive. The field identifier is **web_launch**.

Usage rules

1. Values of the parameters are communicated in the form <parameter Name> = <Parameter value>".
2. These "Name Value Pairs" are separated by an ampersand ("&").
3. Name Value Pairs can be in any order.
4. Parameter names are not case sensitive.
5. Parameter values may be case sensitive.
6. All parameters must be URL-encoded

Assignable Unit Password	**Definition**	A string of characters sent to the CMI system that enables the CMI system to authenticate an assignable unit.
		The AU_password is an optional feature that allows for additional security. The password value is specific to the AU and is sent with HACP request messages, so that the CMI system can authenticate the AU making the request.
		The entry in the AU file is the value that the CMI checks for. The CMI compares the value of this entry with the value passed by the AU.
		The value for the AU_password should be configurable for individual CBT AU's by the system administrator rather than being a static value embedded or "hard-coded" in the AU.
	Data format	Alphanumeric. The values may be case sensitive. Limited to 255 characters. The field identifier is **AU_Password**.
	Examples	rtjh4578gh trust!1

A.3.3   Lesson Evaluation Data Differences

Lesson evaluation differences

Lesson evaluation data files are optional. Their purpose is to enable the use of standard performance analysis tools with many different courses.

In a file-based CMI system, the CBT lesson (Assignable Unit) is responsible for saving student data directly into standard files. The files are described in Chapter 7 and have the following names:

Comments	Section 7.1
Interactions	Section 7.2
Objectives Status	Section 7.3
Path	Section 7.4

CBT
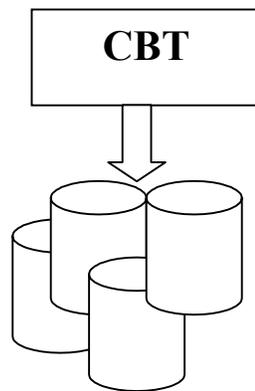
Web-based system

In a Web-based system, it may not be possible for the Assignable Unit to write any files to a disk. Therefore, the CMI system becomes responsible for creating the standard files for analysis,. The data is passed to the CMI system, which then must write out the evaluation data in AICC standard files as described in Chapter 7.

CBT

Intern

CMI
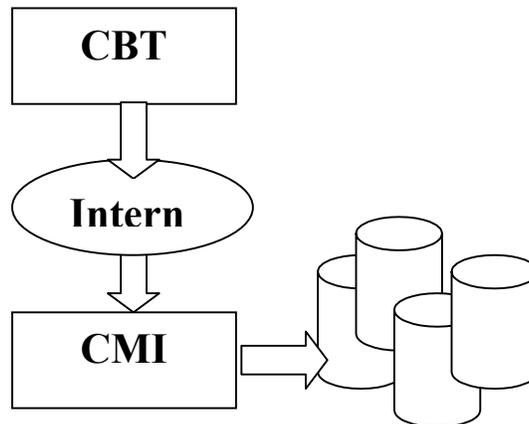
A.4 CBT Assignable Unit URL Command Line

URL command line

When the CMI launches a CBT Assignable Unit, it sends a URL command line" to the HTTP Client (e.g. Web Browser). This command line shall be URL encoded. It contains three basic values.

- URL of the CBT assignable unit
- AICC-required parameters
- Web Launch parameters

Required AICC Parameters

AICC Parameter Name	Format	Value
AICC_SID	Alpha-numeric Example: *AICC_SID=1234WE9*	Session ID. This is a string that uniquely identifies this Assignable Unit session among all other active CMI/CBT Assignable Unit sessions. The CMI system generates and passes this value on Assignable Unit launch. The CBT Assignable Unit will use this value to identify its session when making requests to the CMI system
AICC_URL	URL Location Example: *AICC_URL=http%3A%2F%2Fcompany.com%2Fcgi-bin%2Fcmi.cgi*	The full URL (including the protocol) to which the Assignable Unit will send requests for the CMI. This will likely be a cgi or servlet program on a server. This program acts as a CGI "catcher" for the CMI system.

Example   AICC_sid=123&AICC_URL=http%3A%2F%2Fcmi.net
%2FCGI-Bin%2FCMI.cgi&vendorparam=this

(Example without the URL encoding:
AICC_sid=123&AICC_URL=http://cmi.net/CGI-
Bin/CMI.cgi&vendorparam=this)

**Command line
format**

The Launch line has the form of:

**<URL of CBT Assignable Unit>?<AICC-required Parameter
1>&<AICC-required Paramter 2>& <Web Launch Parameters >**

Where

<URL of CBT Assignable Unit> is the HTTP location of the
primary assignable unit file needed for web launch.

<AICC-required Parameter 1> represents either the AICC_SID or
the AICC_URL parameter as needed by the specific assignable unit
being launched. Parameter requirements and rules are described in
Section A.3.2.

<AICC-required Parameter 2> represents either the AICC_SID or
the AICC_URL whichever is not parameter 1.

<Web Launch Parameters> are those lesson-specific parameters
found in the WEB_LAUNCH field of the course interchange
Assignable Unit File.

The command line is a concatenation including two fields in the
Assignable Unit file -- File_Name and Web_Launch -- and the CMI-
generated required parameters. The file name is separated from the
parameters with a question mark (?). Each of the parameter fields is
separated by an ampersand (&). All values in the name-value pairs,
must be URL encoded.

Data format   Alphanumeric. The values may be case sensitive. All characters to the right of the Question Mark (?) are limited to 255.

Examples   http://www.cbtx.com/CBT/Assignable Unit1.html?aicc_sid=3898i& aicc_url=www.aicc.org%2Fcgi-bin%2Fcmi&vendorparam=plato

(http://www.cbtx.com/CBT/Assignable Unit1.html?aicc_sid=3898i& aicc_url=www.aicc.org/cgi-bin/cmi&vendorparam=plato)

A.5   HTTP Communication

HTTP message format

This section describes the format of the HTTP Messages used in HACP (HTTP AICC CMI Protocol) .

The HACP messages are described in terms of HTTP/1.0 messages. A detailed description of HTTP messages is beyond the scope of this document. For a more definitive description of HTTP messages see RFC1945, Hypertext Transfer Protocol -- HTTP/1.0

Communication between HTTP Clients (Web Browsers) and HTTP servers (Web Servers) is accomplished by messages. There are two kinds of HTTP messages,
- Request (sent from the client) and,
- Response (a reply sent from the server).

HTTP Request messages in HACP use "POST" method with a "content-type" of "application/x-www-form-urlencoded". HTTP Response messages in HACP have a "content-type" of "text/plain".

The HACP request/response message data are contained in the "entity-body" of the request/response messages (respectively).

Name/Value pairs

The message data format follows a convention called "name/value pairs." Name/value pairs are defined as follows:

<name>=<value>

Where the name represents a field title and the value represents the contents of the field.

The following sections describe the format of the entity-body.

A.5.1	**Request HTTP Message Format**

The content-type: application/x-www-form-urlencoded

Format

The format of the entity-body is as follows:

command=<*AICC Command* >**&**
version=<*AICC Spec Version*>**&**
session_id=<*Unique Session Identifier*>**&**
AU_password=<*Assignable Unit specific password(optional)*>**&**
AICC_Data=<*(URL encoded) AICC Data*>
<*end of buffer*>

Where:
 <*AICC Command* > = Any valid AICC HTTP command (See Section A.6.2)
 <*Unique Session Identifier*> = Unique Session Identifier (See Section A.4)
 <*Assignable Unit specific password(optional)*>= Assignable Unit Specific Password (See Section A.3.2)
 < *AICC Spec Version* > = AICC Spec Version (e.g. 1.9)
 <*AICC Data*>=Data specific to the command

Usage rules

Usage Rules:
* All of the above values are URL-encoded
* The Name/value pairs can appear in any order
* If an optional value is to be omitted, the name must also be omitted.
* The name of each parameter is not case sensitive.

Examples

Command=GetParam&version=2.0&session_id=AX36&AICC_Data=

Command=PutParam&Version=2.0&Session_id=DAX36
&AICC_Data=[core]%0D%0A
lesson_location+%3D+end%0D%0Alesson_status%3D
pass%0D%0Ascore%3D87%0D%0Atime%3D00:23:15

Note that %0D is the hex value for carriage return, %0A the hex value for line feed, %3D the hex value for =, and %26 the hex value for &

A.5.2   **HTTP Response Message Format**

The content-type: text/plain
Request Method: POST

Format   The format of the entity-body is as follows**:**

<Name>	<Value>
error=	*<AICC error Number ><CR>*
error_text=	*<AICC error description (optional) ><CR>*
version=	*<AICC Spec Version (optional) ><CR>*
aicc_data=	*< AICC Data>*
	<end of buffer>

Where:
<CR> =	Carriage Return and Line feed (ASCII 13 10)
< AICC error description > =	AICC HTTP error message text (See below)
< AICC error Number > =	AICC HTTP error message Number (See below)
<AICC Data>=	PARAM.CMI data (if GetParam command was issued in the request)

AICC Error messages

Error Number	Error Text
0	Successful
1	Invalid Command
2	Invalid AU-password
3	Invalid Session ID

Usage rules   Usage Rules:

- Leading and trailing white space (Tab, space) is allowed before and after the <**name**>, "=", and <**value**>.
- The <**value**> data in **aicc_data** begins as the first non-white space character after the "=" and continues until the end of the entity-body buffer.
- The <**value**> data for all other <**name**> variables begins as the first non-white space character after the "=" and continues until the last non-white character before the CR (or CRLF).
- The <**value**> data is plain text (and NOT URL-encoded)
- **aicc_data** is only included in response to a GetParam (see A.6.2).request
- If **aicc_data** is returned it must be the last name/value pair in the entity-body.
- 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.

Example   Error=0
error_text = successful
version= 2.0
aicc_data=[core]
Student_ID=B1781
Student_Name=Doe, John
Output_file=
Credit=C
Lesson_Location=
Lesson_Mode=Sequential
Lesson_Status = Not Attempted
path =
Score=
Time = 00:00:00
[evaluation]
Course_ID=B17
[Student_data]
max_time_allowed=00:45:00
time_limit_action=Exit

A.6   **HTTP**

This section references the HTTP standard, and describes the AICC
protocol for the HTTP commands necessary for the AICC CMI
implementation.

A.6.1   **HTTP Standard**

The HTTP 1.0 specification is described in RFC 1945. RFC 2068 was
created from draft 07 of the HTTP 1.1 specification and is not yet a
standard.

Other related RFC's:
- RFC 822 Standard for the Format of ARPA Internet Text Messages
- RFC 1738 Uniform Resource Locators (URL)
- RFC 1808 Relative Uniform Resource Locators
- RFC 1521 MIME (Multipurpose Internet Mail Extensions) Part
  One: Mechanisms for Specifying and Describing the Format of
  Internet Message Bodies

A.6.2   **AICC CMI Protocol (HACP) Commands**

Command	Function	Usage Rules
GetParam	Get input data from the CMI. (Data format Specified in section 5.1)	*Required.* This command can be issued to the CMI multiple times. If the command follows one or more **PutParam**'s, the values received will reflect any changes caused by the **PutParam**'s.

PutParam	Sends output parameter file data to the CMI (Data format Specified in section 5.2)	*Required*. This command can be used multiple times. Each time it is used the output parameter data is replaced. The CMI must only use the data from the final PutParam in a CBT Assignable Unit session. (i.e. this is an "overwrite" operation) Multiple **PutParam**'s may be used to ensure against data loss caused by a dropped connection or abnormal session termination. If a **GetParam** is performed after multiple **PutParam'**s, the values in the **GetParam** will reflect the last **PutParam** values.
PutComments	Comments data (CMI001 7.1) Record	*Optional*. This command can be called multiple times. The CMI system must collect and store the new data content each time this command is called in an Assignable Unit session. (i.e. this an "append" operation)
PutPath	Path data (CMI001 7.4)	(Same rules as **PutComments**)
PutInteractions	Interactions data (CMI001 7.2)	(Same rules as **PutComments**)
PutObjectives	Objectives data (CMI001 7.3)	(Same rules as **PutComments**)
PutPerformance	Performance data (CMI001 5.1.5)	*Optional* Like **PutParam**, this command can be used multiple times, replacing the data (i.e. complete overwrite).
ExitAU	Ends an Assignable Unit session	*Required*. This command can only be issued at the end of a Assignable Unit session.

Appendix B: API-based CMI Communication

B.1　　　　　Introduction

API

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

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

B.1.1　　　　　HTTP Implementation

Appendix A of this document, defines data exchange in terms of HACP (HTTP AICC CMI Protocol), an HTTP-based protocol. HACP has proven successful in commercial products and in large-scale LMS applications. However, the average content developer finds HACP difficult to understand and some LMS applications require protocols other than HTTP.
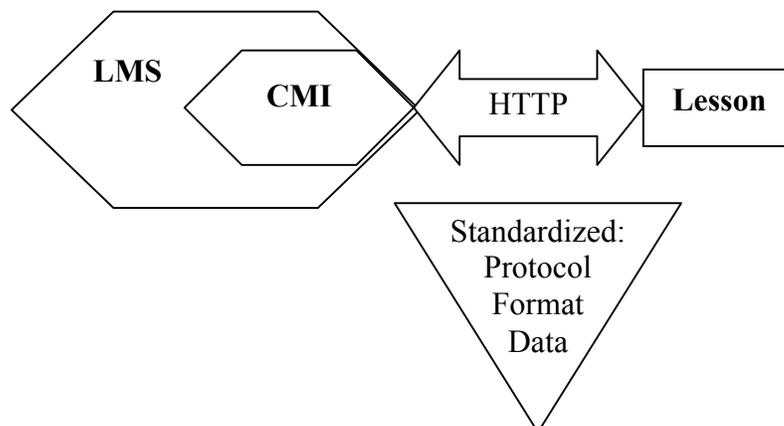
B.1.2   API Implementation

Description   This API standardizes the way content sends and receives information. It assumes that content will communicate using the widely supported ECMAScript calling conventions [9]. ECMAScript was selected as the method for implementing this API since nearly all browser platforms natively support it. This standard defines several calls, the data in these calls, and the format of that data.

The figure below illustrates what is standardized. Note that the communication of the ECMAScript with the LMS is outside the scope of this standard. Implementations of the communications of the JavaScript object with the LMS may vary from product to product."
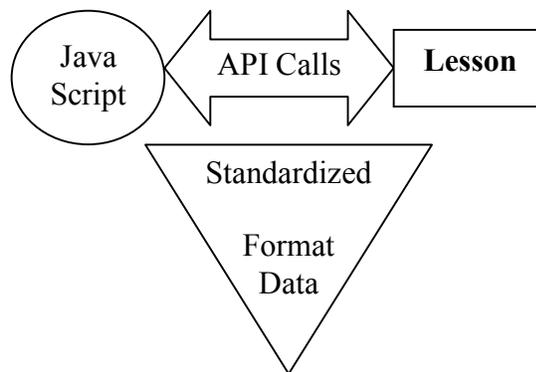
Advantages   There are several reasons for expanding the number of IEEE implementations to include an API standard in addition to an HTTP-based standard.

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

---

[9] ECMAScript is the ISO standard version of JavaScript. In this document the use of the term "JavaScript" is actually a reference to ECMAScript.

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

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

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

B.1.3   **Two Web Implementations**

The API standards defined here may be used to complement the HTTP standards already defined in Appendix A. HTTP may be thought of as one possible implementation for communication. In other words, an LMS can support either an API or HTTP implementation or both implementations simultaneously.
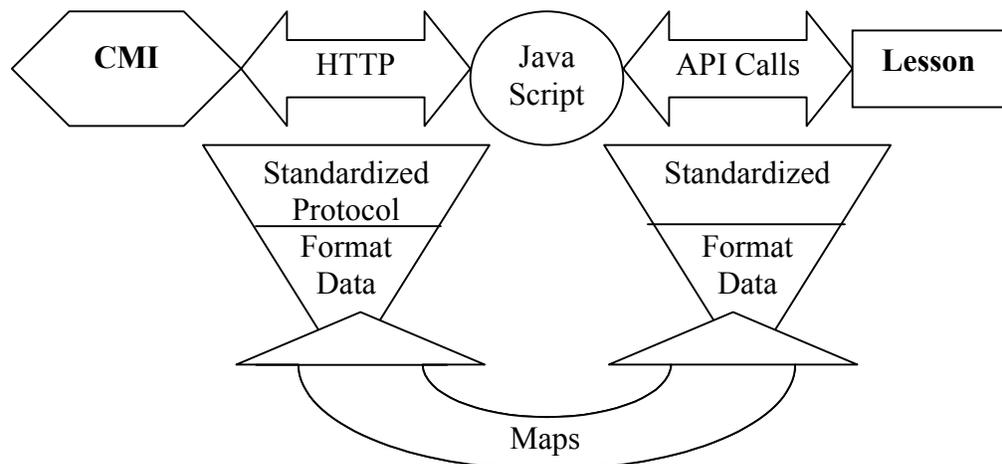
B.2   Conformance Rules

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

B.2.1   Obligation

Levels   There are three levels of obligation for the API's and the data elements described in this standard:
- Mandatory
- Optional
- Extension

Obligations for the content and LMS are different.

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

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

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

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

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

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

B.2.2   CMI Responsibilities

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

B.2.2.1   Launch

For browser and Web-based content, the LMS shall launch the content 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 a URL.

B.2.2.2   Communication

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

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

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

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

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

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

The table below summarizes the requirements for a conforming LMS.

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

B.2.2.3   Sequencing

Flow control is assumed to be the responsibility of the LMS 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 (content) will next be presented to the student is made by the LMS. (This recognizes that some AU's may make decisions—that is, branch – within itself, but that kind of internal flow is hidden from the LMS.

The determination of what the student experiences is determined solely by the LMS and is defined in large part by the Course Structure description (Chapter 6). Chapter 6 defines information about content that is context specific to the course (e.g., the default sequence of AU's, prerequisites that might alter the delivery path.)

Summary Points: A content assignable unit may only be launched by an LMS. 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.

B.2.3   Content Responsibilities

The content is responsible for discovering the API object.

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

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

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

The table below summarizes the requirements for conforming content.

Conformance Requirements for Content

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

B.2.3.1   Binding Mechanism

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

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


// obtain the LMS API
API = FindAPI(window);
if (API == null && window.opener != null)
        API = FindAPI(window.opener);
if (API == null)
        alert("Error: Could not find API");
```

B.2.3.2   Parameter Identification

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

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

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

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

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

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

B.3   API Set

The API's   The set of API function calls consists of the following:
          LMSInitialize("")
          LMSFinish("")
          LMSGetValue(cmi.group.element)
          LMSSetValue(cmi.group.element, value)
          LMSCommit("")
          LMSGetLastError("")
          LMSGetErrorString(errornumber)
          LMSGetDiagnostic(parameter)
          Security Request/Respond --- TBD

B.3.1   **API General Rules**

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

B.3.2   **Arrays -- Handling Lists**

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

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

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.

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

LMSGetValue("cmi.objective._count")

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

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

API Function Table

Function	Description	API Call	Return Value
Initialize	The content must call this function before calling any other API function. It indicates to the LMS system that the content is going to communicate. The LMS can take any initialization steps required in this function.	LMSInitialize("")	A string convertible to CMIBoolean
Finish	The content must call this function before it terminates, if it successfully called LMSInitialize at any point. It signals to the LMS that the content has finished communicating. The content may not call any API function except LMSGetLastError after it calls LMSFinish	LMSFinish("")	CMIBoolean
Get a value	This is used to determine values for various categories and elements in the CMI data model. Only one value is returned for each call. The category and/or element is named in the argument.	LMSGetValue(cmi.category) LMSGetValue(cmi.category.element)	A string convertible to appropriate data type
Set a value	This is how data categories and elements get values. The argument indicates which category or element is being set. Only one value may be set with a single function call.	LMSSetValue(cmi.category, value) LMSSetValue(cmi.category.element, value)	CMIBoolean
Send cache to LMS	If the ECMAScript is caching LMSSetValue values, this call requires that any values not yet sent to the LMS be sent.	LMSCommit(parameter)	CMIBoolean
Determine error code	The content must have a way of assessing whether or not any given API call was successful, and if it was not successful, what went wrong. This routine returns an error code from the previous API call. Each time an API function is called (with the exception of this one), the error code is reset in the API. The content may call this any number of times to retrieve the error code, and the code will not change until the next API call.	LMSGetLastError()	A string convertible to CMIInteger
Obtain text related to error	This function enables the content to obtain a textual description of the error represented by the error code number.	LMSGetErrorString(errornumber)	CMIString255
Determine vendor-specific diagnostics	This function enables vendor-specific error descriptions to be developed and accessed by the content. These would normally provide additional helpful detail regarding the error.	LMSGetDiagnostic(parameter)	CMIString255
Security functions	-TBD-		

B.3.3   Initialize

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

Syntax   LMSInitialize(parameter)

Parameter   "". An empty string must be passed for conformance to this standard. This parameter is reserved for future extensions.

Return value   Boolean.
A "true" result indicates that the initialization was successful and a "false" result indicates that it was not.

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

B.3.4   Finish

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

Syntax   LMSfinish(parameter)

Parameter   "". An empty string must be passed for conformance to this standard. This parameter is reserved for future extensions.

Return value   CMIBoolean.
A "true" result indicates that the function was successful and a "false" result indicates that it was not.

B.3.5   **Get a Value**

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

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

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

Syntax   LMSGetValue(parameter)

Parameters   cmi.element.element
                     Returns the value of the named sub-element

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

cmi.element._count
The _count keyword is used to determine the number of elements
currently in an array. This number is not changed by use of the
LMSCommit call. The count is the total number of elements in the
array, not the last index number used.

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

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

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

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

For LMSGetValue(cmi.group._children) the return value is a comma separated list of all the element names in the specified group or category that are supported by the LMS. If an element has no children, but is supported, an empty string is returned. An empty string is also returned if an element is not supported. A subsequent request for last error [LMSGetLastError()] can determine if the element is not supported. The error "401 Not implemented error" indicates the element is not supported.

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

Examples

LMSGetValue("cmi.core.student_name")
A typical return value might be "Jackson Hyde".

LMSGetValue("cmi.core.lesson_status")
A typical return value might be "incomplete".

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

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

LMSGetValue("cmi.comments._children")
The comments element has no children. A zero length string indicates that comments are supported. No return implies no support for comments.

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

B.3.6	**Set a Value**

Description

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

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

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

Syntax

LMSSetValue(parameter, value)

Parameter

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

The following represents some forms this parameter may take.

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

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

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

Value

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

Return value

CMIBoolean.
A "true" result indicates that the function was successful and a "false" result indicates that it was not.

B.3.7   **Send Cache to CMI**

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

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

Syntax   LMSCommit(parameter)

Parameter   "". An empty string must be passed for conformance to this standard.
                         This parameter is reserved for future extensions.

Return value   CMIBoolean.
                         A "true" result indicates that the function was successful and a "false"
                         result indicates that it was not.

B.3.8   **Determine Error Code**

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

Syntax   LMSGetLastError()

Parameter   "". An empty string must be passed for conformance to this standard. 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   LMS erors
      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 SetValue, element is a CMI keyword
    403. Element is read only
    404. Element is write only
    405. Incorrect data type
      Additional codes TBD

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

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

Syntax   LMSGetErrorString(errornumber)

Parameter   An integer number representing an error code.

Return value   A string that represents the verbal description of an error.

B.3.10   Determine Vendor-specific Diagnostics

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

Syntax   LMSGetDiagnostic(parameter)

Parameter   The parameter may take one of two forms.
- An integer number representing an error code. This requests additional information on the listed error code.
- "". 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.

B.3.11   Security Request/Respond -- TBD

B.4   LMS to Assignable Unit Communications

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

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

Comments
Comments appears 4 times in the following tables. An explanation of how these are to be interpreted is based on their usage in the file-based and HTTP communication implementations.

In the LMS to Assignable Unit Communication table, LMSGetValue("cmi.comments") is defined as a single string of 4096 bytes or less. This string represents annotations or instructor comments that the assignable unit is designed to make available to the student.

In the same table LMSGetValue("cmi.evaluation.comments") is used to determine if a the data collection of comments described in the Lesson Evaluation Data table is possible. The Lesson Evaluation Data table defines an array of comments that can be transmitted. Furthermore, supporting any element in that table requires that the LMS be able to create a separate file containing that element as defined in Chapter 7 Lesson Evaluation Files.

In the Assignable Unit to LMS Communication table, LMSSetValue("cmi.comments") enables the AU to send student comments on content to the LMS. In this case it is assumed that multiple comments may be captured, but all of the comments shall be concatenated into one string and that string shall not exceed 4096 bytes. The string may contain embedded location information.

In the Lesson Evaluation Data table, the LMSSetValue ("cmi.evaluation.comments.n.content") allows the AU to send an unlimited number of comments with a maximum length of 4096 bytes each. Additionally, time and location for each comment may be sent as a separate value.

Table of LMS to Assignable Unit Communications

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
core	Information required to be furnished by all CMI systems. What all assignable units may depend upon at start up.	SV	Man	LMSGetValue("cmi.core._children")	CMIString255
|--student_id	Unique alpha-numeric code/identifier that refers to a single user of the CMI system.	SV	Man	LMSGetValue("cmi.core.student_id")	CMIIdentifier
|--student_name	Normally, the official name used for the student on the course roster. A complete name, not just a first name.	SV	Man	LMSGetValue("cmi.core.student_name")	CMIString255
|--lesson_location	This corresponds to the assignable unit exit point passed to the CMI system the last time the student experienced the AU.	SV	Man	LMSGetValue("cmi.core.lesson_location")	CMIString255
|--credit	Indicates whether the student is being credited by the CMI system for his performance (pass/fail and score) in this assignable unit.	SV	Man	LMSGetValue("cmi.core.credit")	CMIVocabulary
|--lesson_status	This is the current student status as determined by the CMI system, and sent to the assignable unit when it is launched.	SV	Man	LMSGetValue("cmi.core.lesson_status")	CMIVocabulary
|--entry	Indication of whether the student has been in the assignable unit before.	SV	Man	LMSGetValue("cmi.core.entry")	CMIVocabulary

Table of LMS to Assignable unit (cont.)

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
|--score	Indication of the performance of the student during his last session in the assignable unit.	SV	Man	LMSGetValue("cmi.core.score._children")	CMIString255
|--|--raw	Numerical representation of student performance in assignable unit. May be unprocessed raw score.	SV	Man	LMSGetValue("cmi.core.score.raw")	CMIDecimal CMIBlank
|--|--max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSGetValue("cmi.core.score.max")	CMIDecimal CMIBlank
|--|--min	The minimum score that the student could have achieved.	SV	Opt	LMSGetValue("cmi.core.score.min")	CMIDecimal CMIBlank
|--total_time	Accumulated time of all the student sessions in the assignable unit. Normally this is the sum of the session_time values. If no session time is provided by the AU, then the total_time may be the accumulated values of LMS-determined session times.	SV	Man	LMSGetValue("cmi.core.total_time")	CMITimespan
|--lesson_mode	Identification of student-related information that may be used to change the behavior of the assignable unit.	SV	Opt	LSMGetValue("cmi.core.lesson_mode")	CMIVocabulary
suspend_data	Unique information generated by the assignable unit during previous uses, that is needed for the current use.	SV	Man	LMSGetValue("cmi.suspend_data")	CMIString4096
launch_data	Unique information generated at the assignable unit's creation that is needed for every use.	SV	Man	LMSGetValue("cmi.launch_data")	CMIString4096
comments_from_l ms	Instructor comments directed at the student that the assignable unit may present to the student when appropriate.	SV	Opt	LMSGetValue("cmi.comments")	CMIString4096

Table of LMS to Assignable unit (cont.)

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
evaluation	Assignable units may be able to generate detailed student-performance/AU-evaluation information. This category identifies if this functionality is supported by the LMS.	SV	Opt	LMSGetValue("cmi.evaluation._children")	CMIString255
|--course_id	Alpha numeric sequence that provides a unique label for a course.	SV	Opt	LMSGetValue("cmi.evaluation.course_id ")	CMIIdentifier
|--comments	Identifies if the student's comments on an AU can be collected and made available by the LMS in a separate file.	SV	Opt	LMSGetValue("cmi.evaluation.comments.)	CMIBoolean
|--interactions	Identifies what detailed information of a student's interactions in an AU can be collected.	SV	Opt	LMSGetValue("cmi.evaluation.interactions._children")	CMIString255
|--objectives_ status	Identifies what detailed information on assignable unit objectives can be collected.	SV	Opt	LMSGetValue("cmi.evaluation.objectives_status._children")	CMIString255
|--paths	Identifies what detailed information can be collected on the path through the assignable unit taken by the student.	SV	Opt	LMSGetValue("cmi.evaluation.paths._children")	CMIString255
objectives	Identifies how the student has performed on individual objectives covered in the AU.	Arr	Opt	LMSGetValue("cmi.objectives._count") LMSGetValue("cmi.objectives._children")	CMIInteger CMIString255
|--id	A developer defined, AU-specific identifier for an objective.	SV	Opt	LMSGetValue("cmi.objectives.n.id")	CMIIdentifier

Table of LMS to Assignable unit (cont.)

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
|--score	The score obtained by the student after the last attempt to master the objective.	SV	Opt		CMIString255 CMIInteger
|--|--raw	Numerical representation of student performance after each attempt on the objective. May be unprocessed raw score.	SV	Opt	LMSGetValue("cmi.objectives.n.score.raw")	CMIDecimal CMIBlank
|--|--max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSGetValue("cmi.objectives.n.score.max")	CMIDecimal, CMIBlank
|--|--min	The minimum score that the student could have achieved.	SV	Opt	LMSGetValue("cmi.objectives.n.score.min")	CMIDecimal CMIBlank
|--status	The status obtained by the student after the last attempt to master the objective.	SV	Opt	LMSGetValue("cmi.objectives.n.status")	CMIVocabulary
student_data	Information to support customization of an AU based on a student's performance.	SV	Opt	LMSGetValue("cmi.student_data._children")	CMIString255
|--attempt_number	Number of times the student has been in, or previously used the assignable unit.	SV	Opt	LMSGetValue("cmi.student_data.attempt_number")	CMIInteger
|--mastery_score	The passing score, as determined outside the assignable unit.	SV	Opt	LMSGetValue("cmi.student_data.mastery_score")	CMIDecimal
|--max_time_ allowed	The amount of time the student is allowed to have in the current attempt on the assignable unit.	SV	Opt	LMSGetValue("cmi.student_data.max_time_allowed")	CMITimespan
|--time_limit_action	What the assignable unit is to do when the max time allowed is exceeded.	SV	Opt	LMSGetValue("cmi.student_data.time_limit_action")	CMIVocabulary

Table of LMS to Assignable unit (cont.)

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
|--attempt_records	Student's performance after previous times in the assignable unit.	Arr	Opt	LMSGetValue("cmi.student_data.attempt_records._children") LMSGetValue("cmi.student_data.attempt_records._count")	CMIString255 CMIInteger
|--|--lesson_score	The final score obtained by the student after each use of the AU.	SV	Opt	LMSGetValue("cmi.student_data.attempt_records.n.lesson_score._children")	CMIString255
|--|--|--raw	Numerical representation of student performance after each use of the AU	SV	Opt	LMSGetValue("cmi.student_data.attempt_records.n.lesson_score.raw")	CMIDecimal CMIBlank
|--|--|--max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSGetValue("cmi.student_data.attempt_records.n.lesson_score.max")	CMIDecimal, CMIBlank
|--|--|--min	The minimum score that the student could have achieved.	SV	Opt	LMSGetValue("cmi.student_data.attempt_records.n.lesson_score.min")	CMIDecimal CMIBlank
|--|--lesson_status	Indication of the status of the assignable unit after each session.	Arr	Opt	LMSGetValue("cmi.student_data.attempt_records.n.lesson_status")	CMIVocabulary
student_demographics	Student attributes possessed before entering the course.	SV	Opt	LMSGetValue("cmi.student_demographics._children")	CMIString255
|--city	Portion of student's current address.	SV	Opt	LMSGetValue("cmi.student_demographics.city")	CMIString255
|--class	A predefined training group to which a student belongs.	SV	Opt	LMSGetValue("cmi.student_demographics.class")	CMIString255
|--company	Student's place of employment.	SV	Opt	LMSGetValue("cmi.student_demographics.company")	CMIString255
|--country	Portion of student's current address.	SV	Opt	LMSGetValue("cmi.student_demographics.country")	CMIString255
|--experience	Information on the student's past that might be required by an AU to determine what to present, or what presentation strategies to use.	SV	Opt	LMSGetValue("cmi.student_demographics.experience")	CMIString255
|--familiar_name	An informal title that may be used to address the student.	SV	Opt	LMSGetValue("cmi.student_demographics.familiar_name")	CMIString255
|--instructor_name	Name of the person responsible for the student's understanding of the material in the assignable unit.	SV	Opt	LMSGetValue("cmi.student_demographics.instructor_name")	CMIString255

Table of LMS to Assignable unit (cont.)

name	Contextualized Definition	Mult	LMS Obl	Typical API Calls	Return Value
|--title	Title of the position or the degree currently held by the student.	SV	Opt	LMSGetValue("cmi.student_demographics.title")	CMIString255
|--native_language	The language used in the student's country of origin.	SV	Opt	LMSGetValue("cmi.student_demographics.native_language")	CMIString255
|--state	Segment of a country, also called province, district, canton, etc.	SV	Opt	LMSGetValue("cmi.student_demographics.state")	CMIString255
|--street_address	Portion of student's current address.	SV	Opt	LMSGetValue("cmi.student_demographics.street_address")	CMIString255
|--telephone	Telephone number of a student.	SV	Opt	LMSGetValue("cmi.student_demographics.telephone")	CMIString255
|--years_ experience	Number of years the student has performed in current or similar position.	SV	Opt	LMSGetValue("cmi.student_demographics.years_experience")	CMIString255
student_ preference	Student selected options that are appropriate for subsequent AUs.	SV	Opt	LMSGetValue("cmi.student_preference._children")	CMIString255
|--audio	Sound on/off and volume control.	SV	Opt	LMSGetValue("cmi.student_preference.audio")	CMIInteger
|--language	Identifies in what language the information should be delivered.	SV	Opt	LMSGetValue("cmi.student_preference.language")	CMIString255
|--lesson_type	Indicates suitability of preferences to current assignable unit.	SV	Opt	LMSGetValue("cmi.student_preference.lesson_type")	CMIString255
|--speed	Pace of content delivery.	SV	Opt	LMSGetValue("cmi.student_preference.speed")	CMIInteger
|--text	Written content visibility control.	SV	Opt	LMSGetValue("cmi.student_preference.text")	CMIInteger
|--text_color	Written content foreground and background hue.	SV	Opt	LMSGetValue("cmi.student_preference.text_color")	CMIString255
|--text_location	Position of text window on the screen.	SV	Opt	LMSGetValue("cmi.student_preference.text_location")	CMIString255
|--text_size	Magnitude of the written content characters on screen.	SV	Opt	LMSGetValue("cmi.student_preference.text_size")	CMIString255
|--video	Motion picture tint and brightness on the screen.	SV	Opt	LMSGetValue("cmi.student_preference.video")	CMIString255
|--windows	Size and location of video, help, glossary, etc. windows.	SV	Opt	LMSGetValue("cmi.student_preference.windows.n")	CMIString255

B.5   Assignable unit to LMS Communication

Table of Assignable unit to LMS Communication

name	Contextualized Definition	Mult	LMS Obl	API Call	Value Data Type
core	Information required by the CMI system to function.	SV	Man		-
|--lesson_location	This identifies the point where the student leaves the assignable unit.	SV	Man	LMSSetValue("cmi.core.lesson_location", value)	CMIString255
|--lesson_status	This is the student status when he leaves the assignable unit.	SV	Man	LMSSetValue("cmi.core.lesson_status", value)	CMIVocabulary
|--exit	An indication of how or why the student left the assignable unit.	SV	Man	LMSSetValue("cmi.core.exit", value)	CMIVocabulary
|--score	Indication of the performance of the student during his time in the AU.	SV	Man		
|--|--raw	Numerical representation of student performance in the AU. May be unprocessed raw score.	SV	Man	LMSSetValue("cmi.core.score.raw", value)	CMIDecimal
|--|--max	The maximum score or total number that the student could have achieved.	SV	Opt	LMSGetValue("cmi.core.score.max")	CMIDecimal, CMIBlank
|--|--min	The minimum score that the student could have achieved.	SV	Opt	LMSGetValue("cmi.core.score.min")	CMIDecimal, CMIBlank
|--session_time	Time spent in the assignable unit during the session that is ending.	SV	Man	LMSSetValue("cmi.core.session_time", value)	CMITimespan
suspend_data	Unique information generated by the assignable unit, that is needed for future uses. Passed to the CMI system to hold and to return the next time the student starts this AU.	SV	Man	LMSSetValue("cmi.suspend_data", value)	CMIString4096
comments	Student's written remarks recorded during the current use of the assignable unit.	SV	Opt	LMSSetValue("cmi.comments", value)	CMIString4096

Table of Assignable unit to LMS Communication (cont.)

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

Table of Assignable unit to LMS Communication (cont.)

name	Contextualized Definition	Mult	LMS Obl	API Call	Value Data Type
|--|--status	The status of the assignable unit or segment after each attempt.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.status ", value)	CMIVocabulary
|--|--time	Length of time required for each attempt on an AU or segment.	SV	Opt	LMSSetValue("cmi.student_data.tries.n.time ", value)	CMITimespan
student_ preference	Student selected options that are appropriate for subsequent AUs.	SV	Opt	--	-
|--audio	Sound on/off and volume control.	SV	Opt	LMSSetValue("cmi.student_preference.audio ")	CMISInteger
|--language	Identifies in what language the information should be delivered.	SV	Opt	LMSSetValue("cmi.student_preference.language", value)	CMIString255
|--lesson_type	Indicates suitability of preferences to current assignable unit.	SV	Opt	LMSSetValue("cmi.student_preference.lesson_type", value)	CMIString255
|--speed	Pace of content delivery.	SV	Opt	LMSSetValue("cmi.student_preference.speed", value)	CMISInteger
|--text	Written content visibility control.	SV	Opt	LMSSetValue("cmi.student_preference.text", value)	CMISInteger
|--text_color	Written content foreground and background hue.	SV	Opt	LMSSetValue("cmi.student_preference.text_color", value)	CMIString255
|--text_location	Position of text window on the screen.	SV	Opt	LMSSetValue("cmi.student_preference.text_location", value)	CMIString255
|--text_size	Magnitude of the written content characters on screen.	SV	Opt	LMSSetValue("cmi.student_preference.text_size", value)	CMIString255
|--video	Motion picture tint and brightness on the screen.	SV	Opt	LMSSetValue("cmi.student_preference.video ", value)	CMIString255
|--windows	Size and location of video, help, glossary, etc. windows.	Arr	Opt	LMSSetValue("cmi.student_preference.windows.n", value)	CMIString255

B.6   Lesson Evaluation Data

The Lesson Evaluation Data table represents data that is defined in Chapter 7 Lesson Evaluation Data, and is not redundant with other API calls.

This information is all optional. However, if any of this data is supported, it shall be made available by the CMI system in the file formats specified in Chapter 7 Lesson Evaluation Data.

Although this data is used by the CMI system to create the files described in Chapter 7, it is just more data to be communicated by API calls. As such this table should be thought of as an extension of table B.5, the Assignable Unit to LMS communication table.

Lesson Evaluation Data Table

name	Contextualized Definition	Mult	API Call	Value Data Type
lesson_id	Alphanumeric label supplied by the developer.	SV	LMSSetValue("cmi.evaluation.lesson_id", value)	CMIString255
date	The calendar day on which the data is created.	SV	LMSSetValue("cmi.evaluation.date", value)	CMIDate
comments	Freeform feedback from the student. More structured representation than the comments in the Assignable unit to LMS table.	Arr		-
|--time	Indication of when the comment is made.	SV	LMSSetValue("cmi.evaluation.comments.n.time ", value)	CMITime
|--location	Indication of where in the assignable unit the comment is made.	SV	LMSSetValue("cmi.evaluation.comments.n.location ", value)	CMIString255
|--content	The recorded statement of a student.	SV	LMSSetValue("cmi.evaluation.comments.n.content ", value)	CMIString4096
interactions	A recognized and recordable input or group of inputs from the student to the computer	Arr		-
|--id	Unique alphanumeric label created by the assignable unit developer.	SV	LMSSetValue("cmi.interactions.n.id ", value)	CMIIdentifier
|--objectives	Indication of any objectives associated with the interaction.	Arr		
|--|--id	Developer created identifier for an objective.	SV	LMSSetValue("cmi.interactions.n.objectives.n.id", value)	CMIIdentifier

Lesson Evaluation Data Table

name	Contextualized Definition	Obl	API Call	Value Data Type
|--time	Indication of when the interaction is available to the student.	SV	LMSSetValue("cmi.interactions.n.time ", value)	CMITime
|--type	Indication of which category of interaction is recorded.	SV	LMSSetValue("cmi.interactions.n.type ", value)	CMIVocabulary
|--correct_ responses	Expected student feedback in the interaction.	Arr		
|--|--pattern	Definition of possible student response.	SV	LMSSetValue("cmi.interactions.n.correct_responses .n.pattern", value)	CMIFeedback
|--weighting	Factor that is used to identify the relative importance of one interaction compared to another.	SV	LMSSetValue("cmi.interactions.n.weighting ", value)	CMIDecimal
|--student_ response	Description of the computer-measurable action of a student in an interaction.	SV	LMSSetValue("cmi.interactions.n.student_response ", value)	CMIFeedback
|--result	Judgment of the of the student's response.	SV	LMSSetValue("cmi.interactions.n.result ", value)	CMIVocabulary
|--latency	The time from the presentation of the stimulus to the completion of the measurable response.	SV	LMSSetValue("cmi.interactions.n.latency ", value)	CMITimespan
objectives_ status	Information about a student's performance on AU objectives.	Arr	Note: The only objectives data element that is not described in table B.5 Assignable Unit to LMS Communication, is mastery_time.	-
|--mastery_time	Chronological period spent in the objective.	SV	LMSSetValue("cmi.objectives_status.n.mastery_tim e ", value)	CMITimespan
paths	Description of the sequence of events the student experienced in the assignable unit.	Arr		-
|--location_id	Identification of where the student is in the assignable unit.	SV	LMSSetValue("cmi.paths.n.location_id", value)	CMIIdentifier
|--time	Indication of when the student entered the assignable unit segment.	SV	LMSSetValue("cmi.paths.n.time", value)	CMITime
|--status	A record of the student's performance in a segment each time he leaves that element	SV	LMSSetValue("cmi.paths.n.status", value)	CMIVocabulary
|--why_left	The reason a student departed an element in the assignable unit.	SV	LMSSetValue("cmi.paths.n.why_left", value)	CMIVocabulary
|--time_in_ element	How long the student spent in the element.	SV	LMSSetValue("cmi.paths.n.time_in_element", value)	CMITimespan

---

B.7   Data Types

Description   These definitions are for the data types used to describe the format of each data element.   All of the data types have the first three characters of "CMI" to clearly indicate they are data types that may be unique to the CMI data model.

CMIBlank   An empty string.

CMIBoolean   A vocabulary of two words. true or false

CMIDate   A period in time of one day, defined by year, month, and day in the following numerical format YYYY/MM/DD.

CMIFeedback   Structured description of student response in an interaction. The structure and contents of the feedback depends upon the type of interaction. The currently defined interactions are:
True/False:
Feedback is one of the following single characters: 0,1,t, or f.
Choice:
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: { }
Fill-in:
Any alpha-numeric string up to 255 characters. After the first letter spaces are significant.
Numeric:
CMIDecimal
Likert:
Single character. Legal characters are 0 to 9 and a to z.
Matching:
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 all pairs must be matched correctly to consider the interaction correct, then the pairs are surrounded by curly brackets: { }
Performance:
Very flexible format. Essentially an alphanumeric string of 255 characters or less.

Sequencing:
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.

CMIDecimal   Number which may have a decimal point. If not preceded by a minus sign, the number is presumed to be positive. Examples are "2" and "2.2".

CMIIdentifier   Alphanumeric group of characters with no white space or unprintable characters in it. Maximum of 255 characters.

CMIInteger   An integer number from 0 to 65536.

CMISIdentifier   CMI System Identifier: Alphanumeric group of characters that begins with a single letter: A, B, or J and ends with an integer number. One to five numerals may follow the letter. See *System Identifier*.

CMISInteger   A signed integer number from –32768 to +32768.

CMIString255   A set of ASCII characters with a maximum length of 255 characters.

CMIString4096   A set of ASCII characters with a maximum length of 4096 characters.

CMITime   A chronological point in a 24 hour clock. Identified in hours, minutes and seconds in the format: HH:MM:SS.SS Hours and seconds shall contain two digits. Seconds shall contain 2 digits with an optional decimal point and up to two additional digits.

CMITimespan   A length of time in hours, minutes, and seconds shown in the following numerical format: HHHH:MM:SS.S Hours and seconds shall contain two or more digits. Hours has a maximum of 4 digits. Minutes shall consist of 2 digits. Seconds shall contain 2 digits with an optional decimal point and additional digits.

CMIVocabulary   Used to attach specific vocabularies within contexts in a schema. Vocabulary words must be complete and exact matches to those below. Single letters and abbreviations may not be used in API communication. The following are vocabularies included in the CMI Data Model:

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

B.8   Data Comparison

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

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

Table of CMI to Assignable Unit Communication

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

Table of CMI to Assignable Unit Communication (cont.)

Data Model Name	Group/Keyword Title	Why Different
objectives	[Objectives_Status]	Status is only one element in this group.
|--id	J_ID.1	Objectives relationship established by parent name. i.e. objectives.n.id
|--score	J_Score.1	Relationship of score to objective established by parent name and array index.
|--|--raw		Need separate variable for every value
|--|--max		Need separate variable for every value
|--|--min		Need separate variable for every value
|--status	J_Status.1	Objectives relationship established by parent name. i.e. objectives.n.status
student_data	[Student_Data]	
|--attempt_number	Attempt_Number	
|--mastery_score	Mastery_Score	
|--max_time_ allowed	Max_Time_ Allowed	
|--time_limit_action	Time_Limit_Action	
|--attempt_records		Must have a name with no value to enable children.
|--|--lesson_score	Score.1	Added lesson for consistency with status.
|--|--|--raw		Need separate variable for every value
|--|--|--max		Need separate variable for every value
|--|--|--min		Need separate variable for every value
|--|--lesson_status	Lesson_Status.1	Plural convention for possible array.
student_ demographics	[Student_ Demographics]	
|--city	City	
|--class	Class	
|--company	Company	
|--country	Country	
|--experience	Experience	
|--familiar_name	Familiar_Name	
|--instructor_name	Instructor_Name	
|--title	Job_Title	Generalize person's title.
|--native_language	Native_Language	
|--state	State	
|--street_address	Street_Address	
|--telephone	Telephone	
|--years_ experience	Years_Experience	

Table of CMI to Assignable Unit Communication (cont.)

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

Table of Assignable Unit to CMI Communication

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

Table of Assignable Unit to CMI Communication (cont.)

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

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

Data Model Name	Field Title	Why Different
	Course_ID	CMI already has this information. Do not need to return it to the CMI.
	Student_ID	CMI already has this information. Do not need to return it to the CMI.
lesson_id	Lesson_ID	
date	Date	
comments	Comments File	Identifies existence of detailed/extensive comments, not a file.
|--time	Time	
|--location	Location	
|--content	Comment	Content of comment. Keeping the word comment would be redundant.
interactions	Interactions File	
	Course_ID	Superflous (see above).
	Student_ID	Superflous (see above).
	Lesson_ID	Already in data model. (See above)
|--id	Interaction_ID	Full name superflous in data model. (Appears as interactions.id)
|--objectives		Plural for array names.
|--|--id	Objective_ID	
	Date	Already in data model. (See above)
|--time	Time	
|--type	Type_Interaction	
|--correct_responses		Need ability to describe multiple responses.
|--|--pattern	Correct_Response	More accurate term for describing correct (and incorrect) responses.
|--weighting	Weighting	
|--student_ response	Student_Response	
|--result	Result	
|--latency	Latency	

Table of Lesson Evaluation Data (cont.)

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

B.9   Combined Table

Description   The following table merges the "B.6 Lesson Evaluation Data" table with the "B.5 Assignable Unit to LMS Communication" table. It shows how the LMSSetValue API can be used to report data in both tables.

The "Orig" column identifies the original table from which the data element came. Additionally, the data elements that originated in the "B.5 Assignable Unit to LMS Communication" table, are on a gray backgound.

Combined Table

Name	Orig	Mult	LMS Obl	API Call	Value Data Type
core	B.5	SV	Man		-
|--lesson_location	B.5	SV	Man	LMSSetValue("cmi.core.lesson_location", value)	CMIString255
|--lesson_status	B.5	SV	Man	LMSSetValue("cmi.core.lesson_status", value)	CMIVocabulary
|--exit	B.5	SV	Man	LMSSetValue("cmi.core.exit", value)	CMIVocabulary
|--score	B.5	SV	Man		
|--|--raw	B.5	SV	Man	LMSSetValue("cmi.core.score.raw", value)	CMIDecimal
|--|--max	B.5	SV	Opt	LMSGetValue("cmi.core.score.max")	CMIDecimal, CMIBlank
|--|--min	B.5	SV	Opt	LMSGetValue("cmi.core.score.min")	CMIDecimal, CMIBlank
|--session_time	B.5	SV	Man	LMSSetValue("cmi.core.session_time", value)	CMITimespan
suspend_data	B.5	SV	Man	LMSSetValue("cmi.suspend_data", value)	CMIString4096
comments	B.5	SV	Opt	LMSSetValue("cmi.comments", value)	CMIString4096
evaluation	B.6	SV	Opt		
|--lesson_id	B.6.	SV	Opt	LMSSetValue("cmi.evaluation.lesson_id", value)	CMIString255
|--date	B.6	SV	Opt	LMSSetValue("cmi.evaluation.date", value)	CMIDate
|--comments	B.6	Arr	Opt	-	
|--|--time	B.6	SV	Opt	LMSSetValue("cmi.evaluation.comments.n.time ", value)	CMITime
|--|--location	B.6	SV	Opt	LMSSetValue("cmi.evaluation.comments.n.location ", value)	CMIString255
|--|--content	B.6	SV	Opt	LMSSetValue("cmi.evaluation.comments.n.content ", value)	CMIString4096

Combined Table (cont)

Name	Orig	Mult	LMS Obl	API Call	Value Data Type
interactions	B.6	Arr	Opt	-	
|--id	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.id ", value)	CMIIdentifier
|--objectives	B.6	Arr	Opt		
|--|--id	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.objectives.n.id", value)	CMIIdentifier
|--time	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.time ", value)	CMITime
|--type	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.type ", value)	CMIVocabulary
|--correct_ responses	B.6	Arr	Opt		
|--|--pattern	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.correct _responses.n.pattern", value)	CMIFeedback
|--weighting	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.weighting ", value)	CMIDecimal
|--student_ response	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.student _response ", value)	CMIFeedback
|--result	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.result ", value)	CMIVocabulary
|--latency	B.6	SV	Opt	LMSSetValue("cmi.interactions.n.latency ", value)	CMITimespan
objectives	B.5	Arr	Opt		-
|--id	B.5	SV	Opt	LMSSetValue("cmi.objectives.n.id", value)	CMIIdentifier
|--score	B.5	SV	Opt		
|--|--raw	B.5	SV	Opt	LMSSetValue("cmi.objectives.n.score.raw". value)	CMIDecimal
|--|--max	B.5	SV	Opt	LMSSetValue("cmi.objectives.n.score.max", value)	CMIDecimal
|--|--min	B.5	SV	Opt	LMSSetValue("cmi.objectives.n.score.min", value)	CMIDecimal
|--status	B.5	SV	Opt	LMSSetValue("cmi.objectives.n.status", value)	CMIVocabulary
objectives_ status	B.6	Arr	Opt	Note: The only data element that is not described in table B.5 Assignable Unit to LMS Communication, is mastery_time.	
|--mastery_time	B.6	SV	Opt	LMSSetValue("cmi.objectives_status.n.mastery_time ", value)	CMITimespan

Combined Table (cont)

Name	Orig	Mult	LMS Obl	API Call	Value Data Type
student_data	B.5	SV	Opt	--	-
|--tries_during_ lesson	B.5	SV	Opt	LMSSetValue("cmi.student_data.tries_du ring_lesson", value)	CMIInteger
|--tries	B.5	Arr	Opt		
|--|--score	B.5	SV	Opt		
|--|--|--raw	B.5	SV	Opt	LMSSetValue("cmi.student_data.tries.n.s core.raw". value)	CMIDecimal
|--|--|--max	B.5	SV	Opt	LMSSetValue("cmi.student_data.tries.n.s core.max". value)	CMIDecimal
|--|--|--min	B.5	SV	Opt	LMSSetValue("cmi.student_data.tries.n.s core.min". value)	CMIDecimal
|--|--status	B.5	SV	Opt	LMSSetValue("cmi.student_data.tries.n.s tatus ", value)	CMIVocabulary
|--|--time	B.5	SV	Opt	LMSSetValue("cmi.student_data.tries.n.ti me ", value)	CMITimespan
paths	B.6	Arr	Opt	-	
|--location_id	B.6	SV	Opt	LMSSetValue("cmi.paths.n.location_id", value)	CMIIdentifier
|--time	B.6	SV	Opt	LMSSetValue("cmi.paths.n.time", value)	CMITime
|--status	B.6	SV	Opt	LMSSetValue("cmi.paths.n.status", value)	CMIVocabulary
|--why_left	B.6	SV	Opt	LMSSetValue("cmi.paths.n.why_left", value)	CMIVocabulary
|--time_in_ element	B.6	SV	Opt	LMSSetValue("cmi.paths.n.time_in_elem ent", value)	CMITimespan
student_ preference	B.5	SV	Opt	--	-
|--audio	B.5	SV	Opt	LMSSetValue("cmi.student_preference.a udio")	CMISInteger
|--language	B.5	SV	Opt	LMSSetValue("cmi.student_preference.l anguage", value)	CMIString255
|--lesson_type	B.5	SV	Opt	LMSSetValue("cmi.student_preference.l esson_type", value)	CMIString255
|--speed	B.5	SV	Opt	LMSSetValue("cmi.student_preference.s peed", value)	CMISInteger
|--text	B.5	SV	Opt	LMSSetValue("cmi.student_preference.t ext", value)	CMISInteger
|--text_color	B.5	SV	Opt	LMSSetValue("cmi.student_preference.t ext_color", value)	CMIString255
|--text_location	B.5	SV	Opt	LMSSetValue("cmi.student_preference.t ext_location", value)	CMIString255
|--text_size	B.5	SV	Opt	LMSSetValue("cmi.student_preference.t ext_size", value)	CMIString255
|--video	B.5	SV	Opt	LMSSetValue("cmi.student_preference.vi deo", value)	CMIString255
|--windows	B.5	Arr	Opt	LMSSetValue("cmi.student_preference.w indows.n", value)	CMIString255

Glossary

application programming interface	API. A set of standard software interrupts, calls, functions, and data formats that can be used by a computer program to access network services, devices, or operating systems.
API	Abbreviation for "application programming interface."
argument	Keyword argument. The information relating to a keyword that appears to the right of the equal sign. Also called keyword value or keyword data.
assignable unit	The smallest element of instruction or testing to which a student may be routed by a CMI system. It is the smallest unit the CMI system assigns and tracks. A program or lesson launched by the CMI system.
attitude survey	A series of questions used to determine how well a student likes the courseware; or how well the student feels the courseware is working. This aids in measuring customer satisfaction. See *Likert test.*
AU	Abbreviation for "assignable unit."
block	An arbitrarily defined grouping of course components. Blocks are composed of related assignable units or other blocks. This is a term used in the AICC document *CMI Guidelines for Interoperability.* A block may correspond to any level of the AICC instructional hierarchy above lesson, up to and including course.
bookmark	Identification of a location in a lesson to which a student plans to return. Bookmarks are placed by the student for his own reference and review purposes.

CAI	Computer-aided Instruction. Sometimes Computer-assisted Instruction. Normally used as a synonym for CBT. However some make the following distinction between CAI and CBT.

CAI: The computer as an aid to learning. Supports instruction, but is not the prime medium for delivery of instruction. Uses include presentation or practice but not both. (ref. (j))

CBT: Computer as the primary mode of instruction. (ref. (j))

See *CBT* |
| **CBT** | Computer-Based Training. The use of computers to provide an interactive instructional experience. Also referred to as CAI (Computer Assisted Instruction), CAL (Computer-aided Learning), CBE (Computer Based Education), CBI (Computer-based Instruction), etc. |
| **CGI** | Abbreviation of Common Gateway Interface, a specification for transferring information between a World Wide Web server and a CGI program.

A CGI program is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any programming language, including C, Perl, or Visual Basic.

CGI programs are the most common way for Web servers to interact dynamically with users. Many HTML pages that contain forms, for example, use a CGI program to process the form's data once it's submitted. Another increasingly common way to provide dynamic feedback for Web users is to include scripts or programs that run on the user's machine rather than the Web server. These programs can be Java applets, Java scripts, or ActiveX controls. These technologies are known collectively as client-side solutions, while the use of CGI is a server-side solution because the processing occurs on the Web server. (from Webopedia: www.pcwebopedia.com) |

CMI	Computer-Managed Instruction has several definitions. In its broadest sense, it includes the following:

1) Rostering and storing student information.

2) Scheduling students and resources.

3) Computer acquisition and storage of student performance data. This is frequently referred to as Lesson Evaluation Data instead of CMI.

4) Data presentation. After the data has been collected, it can be massaged by the computer, providing meaningful summaries for human interpretation. This is frequently referred to as data analysis instead of CMI.

5) And finally, the computer can make decisions based on its analysis of the student's performance. It can manage the student's learning. It makes decisions as to what material the student should cover next, what material is not necessary, and what remedial actions if any, should be taken.

In some contexts, the term CMI excludes data collection and data analysis. The strictest definition of CMI includes only the fifth aspect, the computer management of the student.

The combination of items 3) and 4) above, is frequently referred to as "Student Evaluation."

The focus of a CMI system is the management of a course.

complex objective	An objective whose mastery requires

- Mastery of two or more objectives.
- Mastery of two or more other complex objectives.
- Completion of two or more lessons (assignable units).
- Completion of two or blocks.
- One or more objectives (complex or simple) and/or one or more lessons and/or one or more blocks.

core item	A value in one of the three "core" groups for CMI/Lesson communication. Most core items are mandatory. The three "core" groups that contain mandatory data are Core, Core_Lesson (suspend_data in the API data model), and Core_Vendor (launch_data in the API data model.) Mandatory items are those, which a lesson may always depend upon being available. The lesson may or may not use the mandatory items, but they are available if required.

course   A complete unit of training. A course generally represents what a student needs to know in order to perform a set of related skills or master a related body of knowledge.

Level 2 in the AICC Hierarchy of CBT Components:
1. Curriculum
2. Course
3. Chapter
4. Subchapter
5. Module
6. Lesson
7. Topic
8. Sequence
9. Frame
10. Object

course elements   Three items which constitute the building blocks for a course description. Each of these building blocks has its own title and attributes.
- Assignable Unit (lesson)
- Block, and
- Objective.

curriculum	A grouping of related courses.

Level 1 in the AICC Hierarchy of CBT Components:
1. Curriculum
2. Course
3. Chapter
4. Subchapter
5. Module
6. Lesson
7. Topic
8. Sequence
9. Frame
10. Object

default A value, argument, or option that is assumed when another is not explicitly specified.

demographics Information associated with a student prior to entering a course. Student attributes. Typical demographic data includes the student's name, job title, years of experience, and native language.

extension A feature that is not defined, but is permitted in this Specification. Extensions are specified elsewhere and agreed upon outside the Specification.

group A unit of information in a standardized file for storing CMI information. Groups are large data items, generally several lines in length. A group extends from the group identifier to the next group identifier, and may include multiple lines. All carriage returns and symbols between group identifiers may be significant, depending on the definition of the specific group. Although groups may contain keywords, they may not contain other groups.

hierarchy The structure of lessons and/or courses determining what will be the student's next assignment

HACP	Initials standing for **HTTP-based AICC CMI Protocol.**

HTTP

Short for HyperText Transfer Protocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed.

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input. This shortcoming of HTTP is being addressed in a number of new technologies, including ActiveX, Java, JavaScript and cookies.

Currently, most Web browsers and servers support HTTP 1.0, but a new version called 1.1 should be implemented soon. One of the main features of HTTP 1.1 is that it supports persistent connections. This means that once a browser connects to a Web server, it can receive multiple files through the same connection. This should improve performance by as much as 20%. (from Webopedia: www.pcwebopedia.com)

IIOP

Short for Internet Inter-ORB Protocol, a protocol developed by the Object Management Group (OMG) to implement CORBA solutions over the World Wide Web. IIOP enables browsers and servers to exchange integers, arrays, and more complex objects, unlike HTTP, which only supports transmission of text. (from Webopedia: www.pcwebopedia.com)

item analysis.

This can indicate how well an element of instruction trains; or how well a test question measures student performance. This enables quality control of the testing and instruction.

interaction	An exchange between a student and a program, beginning with a screen touch, a mouse click, a keyboard, or other input by a student, followed by an on-screen reaction of the program.

In the context of this guideline for storing student performance data: A recognized and recordable input or group of inputs from the student to the computer. |
| **learning management system** | All the functionality of a CMI system, plus the ability to schedule learning resources and schedule and track learners over multiple courses. |
| **lesson** | A meaningful division of learning that is accomplished by a student in a continuous effort -- that is at one sitting. That part of the learning that is between designed breaks. Frequently requires approximately 20 minutes to an hour.
OR
A grouping of instruction that is controlled by a single executable computer program.
Or
A unit of training that is a logical division of a subchapter, chapter, or course.

Level 6 in the AICC Hierarchy of CBT Components:
    1. Curriculum
    2. Course
    3. Chapter
    4. Subchapter
    5. Module
    6. Lesson
    7. Topic
    8. Sequence
    9. Frame
    10. Object |
| **lesson element** | An arbitrary division of an assignable unit that has been uniquely named (has its own ID). An assignable unit may have from two to hundreds of lesson elements. |

LMS	Abbreviation for Learning Management System.

keyword	A unit of information in a standardized file for storing CMI information. Keywords are names of data items that are limited in size to a single line. This generally limits the data to 60 or 70 characters.

Likert test	A Likert test is made up of a series of Likert questions. Each 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 FROM

Strongly agree	TO	Strongly disagree
Should be more	TO	Should be less
Understand completely	TO	Do not understand at all

One way in which the Likert test differs from a multiple choice test is that the Likert test has no correct answer for each question.

See *attitude survey.*

mandatory	Pertaining to features that are defined and required by this specification.

optional	Pertaining to features that are defined but not required by this Specification.

part-task trainer (PTT)	A device that simulates a part of some sophisticated hardware, such as an airplane. It permits selected aspects of a task to be practiced independently of other elements of the task. Its purpose is to provide economical training on certain elements requiring special practice but that are not dependent upon the total equipment.

performance analysis.	Determination of a student's capabilities, based upon data collection of the student's interactions within one or more lessons. This helps to determine what the student knows, and what he learns. Comparing individual student progress with his peers gives a measurement of individual rate of learning.

RFC	Short for Request for Comments, a series of notes about the Internet, started in 1969 (when the Internet was the ARPANET). An RFC can be submitted by anyone. Eventually, if it gains enough interest, it may evolve into an Internet standard.
	Each RFC is designated by an RFC number. Once published, an RFC never changes. Modifications to an original RFC are assigned a new RFC number. (from Webopedia: www.pcwebopedia.com)

router	Software which sequences a series of lessons, tests, and other assignable units in a course. The router determines the order in which the student experiences segments of his computer-based training.

score	A result of a learner's assessment expressed as a numerical value or a point on a descriptive scale.

session	The period of time, during which a user of a terminal can communicate with an interactive system, usually equal to elapsed time between logon and logoff. (610.10 – *1994 IEEE Dictionary*)

structure elements	The parts of a course which can be uniquely assigned by a CMI system. These are units that can be rearranged to determine the order in which a student experiences a course of instruction. There are two structure elements in the AICC view of a course description: • Assignable unit (the lesson) • Block

TCP/IP	Initials for Transmission Control Protocol/Internet Protocol, the suite of communications protocols used to connect hosts on the Internet.

TCP/IP uses several protocols, the two main ones being TCP and IP. TCP/IP is built into the UNIX operating system and is used by the Internet, making it the de facto standard for transmitting data over networks. Even network operating systems that have their own protocols, such as Netware, also support TCP/IP. |
| **URI** | Short for Uniform Resource Identifier, the generic term for all types of names and addresses that refer to objects on the World Wide Web. A URL is one kind of URI. (from Webopedia: www.pcwebopedia.com) |
| **URL** | Abbreviation of Uniform Resource Locator, the global address of documents and other resources on the World Wide Web.

The first part of the address indicates what protocol to use, and the second part specifies the IP address or the domain name where the resource is located.

For example, the two URLs below point to two different files at the domain sandybay.com. The first specifies an executable file that should be fetched using the FTP protocol; the second specifies a Web page that should be fetched using the HTTP protocol:

       ftp://www.sandybay.com/stuff.exe
       http://www.sandybay.com/index.html

(from Webopedia: www.pcwebopedia.com) |

URL encoding   HTML form data is usually URL encoded. For the AICC interoperability, ALL AICC data that is part of the Request-Body must be URL encoded. Here are the rules for URL encoding:

- Convert all "unsafe" characters in the names and values to "%xx", where "xx" is the ASCII value of the character, in hex.
- The only safe characters are alphanumerics and the following
  $ - _ . ! * " ( ) ,
  All other characters are unsafe.
- Examples of unsafe characters are
  = & % +
- Change each space to "+" (plus) or "%20".

value   Keyword data. The information relating to a keyword that appears to the right of the equal sign. Also called keyword argument.

Index