

DRAFT AICC White Paper CBT Design and Development Recommendations for Users of AICC CMI Systems

AICC CMI Systems implicitly involve a wide audience—instructional designers, subject matter experts, CBT developers, students, instructors, CBT/CMI administrators and others. As such a CMI system can have a tremendous impact on course design, development, and delivery. *CMI Guidelines for Interoperability* (CMI001) addresses technical and functional specifications for AICC compliant CMI systems. This White Paper addresses CBT design and development considerations affected by the CMI Guidelines, based on experience and best known methods of AICC members. These recommendations are not requirements, nor are they the only approaches likely to succeed. Rather these recommendations are intended to facilitate CBT design and development for experienced courseware personnel using AICC CMI systems for the first time.

CBT Design and Development Recommendations

1. Use a highly modular and granular instructional design approach with small, short learning activities and clear separation of each type of activity (e.g., evaluation, drill-and-practice, tutorial, simulation, game).
 - 1.1. Rationale: Allows end-user designers to more easily reconfigure courses, provides flexibility, facilitates searching and indexing to allow “just-in-time” and “just-enough” training. This also allows the same evaluation instrument to be used as a quiz, test or pre-test.
 - 1.2. Example: Instead of building one 15 minute courseware file including a tutorial and test, plan on a 10 minute tutorial and a 5 minute test. The same test could potentially be a quiz or pre-test for other audiences.
2. Review AICC CMI data collection, description and generation capabilities at the *beginning* of your courseware design phase (*well before* lesson development).
 - 2.1. Rationale: Failure to anticipate data needs can result in unstable or cumbersome work-arounds during development. Likewise, though it may be wise to err on the side of caution, excessive data collection may needlessly burden lesson design *and* development, despite no clear need for specific historical or analytical data.
 - 2.2. Example: Similar to comparing objectives against test items, consider comparing reporting and analysis needs against data collection requirements.
3. Consider AICC CMI item types and data while designing evaluation instruments and scored interactions.
 - 3.1. Rationale: Failure to anticipate data needs can result in cumbersome or confusing work-arounds which misuse data fields or may not work under other CMI systems.
 - 3.2. Example: AICC interaction records are suitable for collecting data for item analysis. The interaction type field supports 7 specific interaction types and a unique or unanticipated type. By failing to review these types you may unnecessarily constrain lesson design to only a few types of interactions.
4. Review other relevant AICC documents during lesson design—there may be additional synergies and benefits.
 - 4.1. Rationale: AICC subcommittees coordinate guidelines and recommendations to assure consistency and facilitate courseware development and delivery. Documents not directly related to CMI-CBT communication may still provide benefits during course design and development.
 - 4.2. Example: AICC platform recommendations may open up additional media capabilities or identify incompatibility issues that impact design. Additionally, recommendations on user interface elements

DRAFT Development Recommendations for CMI Users 7/24/00

such as icons, may expedite development by providing alternatives to select from rather than a blank slate from which you must create alternatives.

5. Avoid “chaining” lesson files together and passing data between them.
 - 5.1. Rationale: Use of chaining defeats the CMI representation of a single “activity” and reduces the value of CMI interoperability. It makes lessons less portable across CMI systems by introducing another level of menuing and implicitly another level of communication. It requires extra diligence on the part of the CMI to determine if the “lesson” has truly finished (e.g., 1st exe passes control to the 2nd, is the lesson done?).
 - 5.2. Example: Avoid chaining together two simulation files and using the first to establish the “state” of the second. Instead, allow either simulation to be launched from the CMI and include logic to initialize the simulation within the second file.
6. If functionality can be achieved at the lesson or CMI level, attempt to design and implement at the CMI level *first*.
 - 6.1. Rationale: Use of sub-routers reduces the value of CMI interoperability. It makes lessons less portable across CMI's by introducing another level of menuing and implicitly another level of communication.
 - 6.2. Example: Avoid hard-coding mastery level into an evaluation, since the CMI can pass this data. This allows other designers to change the mastery level without modifying courseware, thus expanding the potential uses and life cycle of the evaluation.
7. Avoid “sub-routers”— Let the CMI handle AU-level menuing.
 - 7.1. Rationale: Use of sub-routers reduces the value of CMI interoperability. It makes lessons less portable across CMI's by introducing another level of menuing and implicitly another level of communication.
 - 7.2. Example: This is a specific instance of R6. Rather than having a courseware-based (lesson) file that offers a menu of a logical group of lessons, organize lessons into logical groups using CMI. Be sure to provide a quick and easy way to return directly to the CMI menu from within the lessons.
8. Set PARAM\$CMI in AUTOEXEC.BAT file.
 - 8.1. Rationale: Many CMI systems do not properly set this variable (more likely to be used by DOS courses). Even if a CMI sets this variable, setting it in AUTOEXEC.BAT should have no ill effect. Also, be sure to use an ALL CAPS representation of the variable name since Window 95/NT are case-sensitive for environment variables, but DOS/Windows 3.x is not.
 - 8.2. Example: Consider providing written instruction to CBT/CMI administrators, or including code in your CBT setup/installer that adds a line to the system AUTOEXEC.BAT similar to:

```
set PARAM$CMI=C:\WINDOWS
```

Be sure to use installers features or Windows API calls to identify the windows directory, since it is not always “c:\windows”.
9. Centralize Read/Write Logic.
 - 9.1. Rationale: In the event you have misinterpreted standards or later require additional functionality, maintenance and revision will be simplified by having fewer places within the courseware that require changes.
 - 9.2. Example: Rather than including logic to update parameters to CMI upon lesson entry, topic switch and lesson exit, create a subroutine or code library and put calls to that logic in those choke points.
10. Externalize Read/Write Logic.
 - 10.1. Rationale: In the event you have misinterpreted standards or later require additional functionality, maintenance and revision will be simplified by being able to make change without any modification to individual courseware files.
 - 10.2. Example: Rather than including logic to update parameters to CMI in courseware code, consider writing or using a DLL. If alternate CMI API arises or AICC standards are enhanced you will be able

DRAFT Development Recommendations for CMI Users 7/24/00

to extend or enhance courseware by changing the actions of the DLL, without changing calling parameters or courseware.

11. Don't make work for yourself—Read/Write *all* required data and *only* the optional fields your courseware or reporting requirements need.
 - 11.1. Rationale: AICC CMI standards are intended to encompass the widest possible range of courseware and interactions. Implementing such broad functionality in courses with a limited focus may unnecessarily burden them and create superfluous maintenance requirements.
 - 11.2. Example: A brief introductory lesson with a sample simulation or judged interaction may not warrant the extra overhead of an interactions file, particularly if the analysis phase does not identify a reporting need for that data.
12. Call system “close” function or delete param.cmi after reading all data.
 - 12.1. Rationale: This offers a minor improvement in security by reducing the window of opportunity for users to view this file. This also prevents lessons which determine the presence of AICC CMI by checking for “param.cmi”, from inadvertently “assuming” CMI is present if an old file is still lurking.
 - 12.2. Example: If combined with the recommendation to externalize read/write functions (R10), a “CMIReadComplete()” function could delete a param.cmi file or eventually close a TCP/IP socket or pipe. Either or both functions could be done without changing courseware.
13. Write lesson data incrementally and ASAP, not just upon exit.
 - 13.1. Rationale: This provides more fault tolerance in case of a power outage or other abnormal termination of the program. Note this does potentially offer a minor reduction in security by expanding the window of opportunity for end-users to view this file.
 - 13.2. Example: Rather than including logic to write data only upon exit, create output data immediately upon opening the lesson and update parameters at key choke points such as completing scored interactions, leaving bookmarks, completing or switching topics, lesson completion and exiting the lesson.
14. Write interaction data incrementally, *immediately* after judging a response, rather than all-at-once.
 - 14.1. Rationale: Interaction data is written separately from general CBT-to-CMI data. The same rationale is applicable to this data as to general parameters. Incremental writes provide more fault tolerance in case of a power outage or other abnormal program termination. This too potentially reduces system security by expanding the window of opportunity for end-users to view or manipulate this file.
 - 14.2. Example: Rather than including logic to write data only upon completion of a bank of test items, write interaction data immediately after or concurrently to providing feedback (or performing judgment).
15. Assign default values to all required data keys—whether read or written.
16. Be mindful of data types and formats (e.g., integer vs string, time/date formats).
17. Assure graceful exit through data output routines.
 - 17.1. Rationale: Make sure the lesson passes through the “Write” code to update any final CMI date. Students may quit by using Control-Q, Alt+F4 or other methods, which might otherwise bypass your data output routines and lose student data.
18. Test for an AICC CMI by looking for a Non-NULL value of Lesson_Status.
 - 18.1. Rationale: Some lessons may be written to function with or without an AICC CMI system (or with alternate CMI's). Future versions of AICC may pass data by a means other than an INI file mechanism. Used in conjunction with the recommendations for centralizing and externalizing read/write code, this is the safest means of assessing if an AICC CMI system is controlling the lesson.

DRAFT Development Recommendations for CMI Users 7/24/00

19. Perform one-time lesson initializations contingent on a Non-NULL value of Lesson_Status **and** a Null value for Time.
 - 19.1. Rationale: Some lessons may be written to function with or without an AICC CMI system (or with alternate CMI's). Future versions of AICC may pass data by a means other than an INI file mechanism. Used in conjunction with the recommendations for centralizing and externalizing read/write code, this is the safest means of assessing if an AICC CMI system is controlling the lesson.
20. Avoid relying on Core_Vendor data if possible.
21. Keep Lesson and Vendor data (Core_Lesson and Core_Vendor) in INI "Key" form.
 - 21.1. Rationale: This will facilitate use/reuse of functions used to externalize and centralize read/write functions. Also, future versions of AICC may pass data by a means other than text files. Since the INI structure (file, section, key) is key to current AICC CMI communications, future mechanisms are more likely to provide a smoother transition for data in this format. It will also make data more readable during lesson/CMI debugging.
22. Keep Core_Lesson data less than 254 bytes and Core_Vendor data less than 254 bytes—incl. key size and <CR>.
 - 22.1. Rationale: Though Microsoft limits INI files to 32K total, AICC specifications limit these sections to 254 bytes. Though a particular CMI *may* support more data, other equally AICC compliant systems may not support additional data..
23. Provide external documentation explaining vendor-specific or lesson-specific data stores, types, and formats.
24. Avoid using Core_Lesson to store data which is already represented by optional data keys (e.g., student bookmarks).
25. Keep all path names in short file name form (i.e., "8.3" or "progra~1" format).
 - 25.1. Rationale: Though the CMI system may be 32 bit and support long file names, 1 or more lessons or even courses may be 16 bit. Many 16 bit programs will interpret the 1st space in a long file name as the separator between arguments.
26. Do not assume path names are in short file name format (or even are DOS paths).
 - 26.1. Rationale: Though your CMI system and lessons may be "16 bit friendly" others may not. Also, at some future point URLs may be valid paths under AICC Standards
27. Do not misuse data stores (or combinations thereof) to store data they are not intended to represent.