

Maintainability Scripting and Coding Requirements

The contractor shall comply with the maintainability instructions and the code/script conventions presented below.

Maintainability. To allow easy editing and enhance readability by the government, all source code created and delivered for this IMI shall be designed and programmed using “open standards” as presented in the open source initiative web site, <http://www.opensource.org>. The contractor shall create all source material using programming techniques consistent with open standards requirements for software:

- **No Intentional Secrets:** The Contractor **MUST NOT** withhold any detail necessary for interoperable implementation.
- **Availability:** The contractor must deliver the complete set of content source files to the government under royalty-free terms with no distribution restrictions.
- **Patents:** All patents involving any source objects, scripts, tagging or code in this courseware’s implementation **MUST**:
 - be licensed under royalty-free terms for unrestricted use, or
 - be covered by a promise of non-assertion when practiced by open source software.
- **No Agreements:** There **MUST NOT** be any requirement for execution of a license agreement, NDA, grant, click-through, or any other form of paperwork to deploy conforming implementations of any code in the courseware.
- **No Open Source Requirement-Incompatible Dependencies:** Implementation of the content **MUST NOT** require any other technology that fails to meet the criteria of this Requirement.

The contractor shall ensure the addition of other scenarios is accomplished using tools approved for use on government computers. The contractor shall provide documentation that describes how to edit scenarios, how to add additional scenarios, and how each aggregated objective, lesson, module, etc. is constructed.

Code/Script Naming Convention Guidelines. Identifier names should be clear and informative. The name of any identifier should succinctly describe the purpose of that identifier. Avoid abstract names (in a global context) that are likely to be reused by other parts of the system. Names should be formed from composite words, delimited by upper case letters (Camel Casing), with no underscores allowed (except for appending pointer identifiers etc.)

1. *Function Names*

Function names must identify as far as possible the action performed or the information provided by the function. “Obvious” names (e.g. “util”) for modules should be avoided. Module names should be kept short. Function names should normally be formed from two parts: an action (verb) and an object (noun) of the action. Examples of acceptable function names are *chatCloseSession*, *gpsSetTime* and *driveIsActive*.

2. *Variable Names*

The type and purpose of each variable should be evident within the source code in which it is used (e.g. the reader will expect counter to be an *int*, *motorSet* might be a BOOLEAN or an array representing a set – context will usually clarify this). Variable names should be short, but meaningful. The contractor should create a variable or data definition comment section if they have any concerns the type and purpose of each variable is not clearly evident for context alone.

3. *Commenting Guidelines*

Always comment the scripts, xml, text and code in your content. Comments should document every decision that was made while building the instructional material. At each point where a choice was made about how to implement decision logic, place a comment describing that choice and why it was made. Additionally, comments should provide information that is not otherwise available from reading the content source files. Write comments at a higher level of abstraction than the scripts and code. Comments that only restate what is already obvious add nothing to the source files and should be avoided. Comments should not speak to how the scripts and code work, but should describe what it does. A blank line should precede all comments, and each comment should be aligned closely with the scripts and code to which they apply. Insert comments before the subject of the comment whether commenting a single line of script or a block of code, never put the comment on the same line with script or code.