Mosaic Learning SCORM Documentation

May 2017 – SCORM 1.2

# Software Stack

When designing SCORM content this is the structure of the software that will be employed. The LMS is at the base layer, which will talk to the SCORM layer, the SCORM layer implements an API for interactions. It is possible to interact with the API directly, but for simplicities sake the use of the Pipwerks wrapper will significantly ease development.

None of the previous layers will ever need to be modified by multimedia developers. The layers that will require modification sit above the Pipwerks layer. The ML\_SCORM layer is a Javascript file that is a further wrapper of the Pipwerks API with many convenience functions, simplicity shortcuts, and local constants available for consistency. This is a file that is currently in development, so while not complete, once it is more stable should need relatively little editing for day to day programming tasks.

Any custom functionality (bookmarking, objective tracking, SCO completion, exercise scoring, etc.) should be placed in a custom Javascript file that will make use of the ML\_SCORM API, and occasionally the Pipwerks API. Ideally all Pipwerks functions will be wrapped in the ML\_SCORM file for simplicities sake.

You should include the Pipwerks file, ML\_SCORM, and your custom Javascript in each HTML file that represents your SCO, and they should be loaded in that order. If the SCO is self-contained all communication between the SCO and the LMS will be done through the custom Javascript file.

If the SCO is a wrapper for smaller lessons/objectives (as in DC Theory) the SCO HTML can also contain an iframe to load more granular content. It is important to note that the iframe will not be responsible for communicating with the LMS. It will communicate with the parent HTML which will in turn communicate with the LMS. It should not have access to the ML\_SCORM or Pipwerks files. It will have its own custom Javascript file however for its internal housekeeping and to communicate with its parent layer.

# Ml\_scorm.js

*This file is in active development so the following is subject to change. Best attempts will be made to update the documentation as the framework changes. The most recent version of this project can be found at:* <https://github.com/EVMosaic/ml_scorm>

The ml\_scorm.js file is the heart of the Mosaic Learning SCORM development process. It is the middle man between the SCORM API and your custom Javascript which will drive your page. In the following section I will briefly describe the most important parts of it. Once you understand the core principles the rest should be fairly straight forward to understand.

The main engine behind ml\_scorm.js is the Pipwerks SCORM wrapper (SCORM\_API\_wrapper.js). It is included in the git repository and also available for download at <https://pipwerks.com/laboratory/scorm/> This needs to be loaded before ml\_scorm.js

let scorm = pipwerks.SCORM;

Pipwerks gives you access to the pipwerks.SCORM object. For convenience it is provided under the scorm shortcut in ml\_scorm.js. Any Pipwerks functions can be accessed through dot notation on the scorm variable.

let lmsConnected = false;

Another important global variable is the lmsConnected variable. This will be initialized with the rest of the SCORM initialization. If the LMS cannot be connected to this will be set to false and prevent any interactions with the LMS. This is not something that needs to be actively managed.

## Constants

Several constants have been provided to ensure compatibility with the SCORM 1.2 storage options. When setting SCORM variables that fall into one of these categories be sure to use the constants to maintain consistency.

const STATUS = {

PASSED : "passed",

FAILED : "failed",

COMPLETED : "completed",

INCOMPLETE : "incomplete",

BROWSED : "browsed",

NOT\_ATTEMPTED : "not attempted"

}

The STATUS constant allows access to the status conditions for objectives and lessons.

const EXIT = {

TIMEOUT : "time-out",

SUSPENED : "suspend",

LOGOUT : "logout",

NORMAL : ""

}

The EXIT constant allows access to the exit conditions for SCORM termination.

const INTERACTION = {

TF : "true-false}",

CHOICE : "choice",

FILL : "fill-in",

MATCH : "matching",

PERFORMANCE : "performance",

LIKERT : "likert",

SEQUENCE : "sequencing",

NUMERIC : "numeric"

}

The INTERACTION constant allows access to available interaction types.

## Debug functions

Several debug functions have been included to help with debugging code. These are wrappers of the various console functions and included under the DEBUG constant. The main difference is the inclusion of the DEBUG\_ENABLED flag which when set to false will bypass all debug logging. You can toggle this to quickly strip all debug output from your code. Any output that should remain regardless of being in debug or production should be implemented using the standard console functions.

const DEBUG\_ENABLED = true;

const DEBUG = {

LOG : function(msg) {

if (DEBUG\_ENABLED) {

console.log(msg);

}

},

ERROR : function(msg) {

if (DEBUG\_ENABLED) {

console.error(msg);

}

},

WARN : function(msg) {

if (DEBUG\_ENABLED) {

console.warn(msg);

}

},

INFO : function(msg) {

if (DEBUG\_ENABLED) {

console.info(msg);

}

}

}

## Core Functions

The following four functions make up the bulk of the ml\_scorm.js framework. The first two are necessary for opening and closing the connection to the LMS and the last two are the main workhorses for interacting with the LMS. Nearly all remaining functions are convenience functions that wrap these two in order to make it easier to interact with the LMS.

function initSCO() {

lmsConnected = scorm.init();

}

initSCO() **must be called before any other SCORM functions can be called.** It should be done at the beginning of your custom Javascript file on every SCO that you create. If it is able to communicate with the LMS it will set lmsConnected to true and allow you to interact with the LMS with the remaining functions.

Under the hood this calls pipwerks.SCORM.connection.initialize() which in turn is responsible for calling LMSInitalize()through the LMS API implementation.

function closeSCO() {

scorm.quit();

lmsConnected = false;

}

closeSCO() must be called before closing the SCO or the window containing it. It should be called from the custom Javascript file when the SCO is no longer needed. Either closeSCO() or a function containing it and any other cleanup tasks should be bound to both the unload and beforeunload events to ensure it is reliably called. It will only ever be called once. Once closeSCO() is called the connection with the LMS is terminated and cannot be reopened again until the page has been reloaded.

Under the hood this calls pipwerks.SCORM.connection.terminate() which in turn is responsible for calling LMSFinish through the LMS API implementation;

function getValue(param) {

DEBUG.INFO(`retrieving value for ${param}`);

if (lmsConnected) {

let value = scorm.get(param);

DEBUG.LOG(`found value of ${value}`);

return value;

} else {

DEBUG.WARN('LMS NOT CONNECTED');

}

}

After the LMS has been initialized you have the ability to interact with it from your custom javascript file. The first way you can do that is by retrieving values from the LMS using the getValue(param) function. If the LMS is connected to it will look up the requested parameter and return the value found.

This is a wrapper of the pipwerks.SCORM.get(param) function. The only advantage of calling it this way is that it automatically checks if the LMS is connected before retrieving the value, and allows for custom debug messaging.

function setValue(param, value) {

if (lmsConnected) {

DEBUG.LOG(`setting ${param} to ${value}`);

scorm.set(param, value);

scorm.save();

} else {

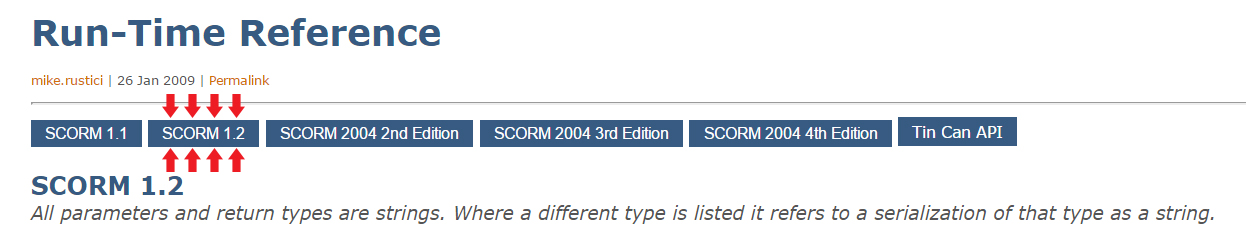
DEBUG.WARN('LMS NOT CONNECTED');

}

}

The other side of the lookup is setting values which you can do with setValue(param, value). Again the function will check if the LMS is connected before updating the value in the LMS. The setValue function will handle saving for you automatically so you do not need to call scorm.save() after any values being set in your custom Javascript.

This function is a wrapper of the pipwerks.SCORM.set(param, value) function. The added benefits of using this one are the auto checking for the LMS connection, and the auto save feature. It also allows for custom debug messaging. If you are going to call the Pipwerks function independently you need to call scorm.save() in order to save your data ta the LMS.



The values available for lookup and setting are listed at <http://scorm.com/scorm-explained/technical-scorm/run-time/run-time-reference/> Make sure to click SCORM 1.2 to get correct names. You can also read much more in depth about the individual variables in the SCORM\_1\_2\_pdf zip file included in the documentation folder. This file contains 4 official pdfs from which the bulk of the knowledge for this documentation came from. If there is ever any doubt about a behavior consult those first and take their word over anything written here.

## Convenience Functions

Several convenience functions have been added to save the hassle of repeated lookups of exact SCORM compliant names. These are all wrappers of getValue and setValue primed with the correct SCORM variable names. These will most likely be the most frequently added functions as they become needed in the process of development.

An example is provided below, but anywhere they are included in the code there are comments detailing their intended purposes, and the simplicity of their function should make it apparent what their intended outcome is.

function completeSCO() {

setValue('cmi.core.lesson\_status', STATUS.COMPLETED);

}

This function will set the entire SCO to complete in the LMS. It can be called immediately before closeSCO or anywhere else it makes sense to mark the course complete. Note the verbose SCORM name which is avoided as well as the use of the STATUS constant.

## Imsmanifest.xml

Coming Soon!

## Objectives

Coming Soon!

Probably going to rework this to use OOP.

## Bookmarks

SCORM allows for the storage of a lesson location bookmark which can be updated as a student progresses through a course. Upon returning to the course this bookmark can be retrieved and used to initialize the SCO to the point the student left off at. Two convenience functions have been added for ease of use of this feature.

function getBookmark() {

return getValue('cmi.core.lesson\_location');

}

function setBookmark(location) {

setValue('cmi.core.lesson\_location', location);

}

You can store the location as a number or a string, but it is important to note that the values retrieved from the LMS will be returned as strings. If you need to manipulate the returned value as a number or another type you will need to coerce it manually.

## Interactions

See page 56 in SCORM\_1\_2\_RunTimeEnv.pdf for more detailed information

Interactions are stored in an array on cmi.interactions. You have the option to add id, objectives, time, type, correct\_responses, weighting, student\_responses, result, and latency to each interaction. You can access individual interactions through cmi.interactions.n where n is the index of the interaction and the previous options are accessible through dot notation on the n. You are responsible for managing this number manually as the interactions will otherwise be added on to the end of the array which can cause unintended ordering if students are allowed to navigate non-linearly through the course.

cmi.interactions.n.type requires one of the variables made available in the INTERACTION constant.