Simple SCORM LMS Adapter

Full Documentation

Version 5.0

Table of Contents

Introduction

The *Simple SCORM LMS Adapter (SSLA) Full Documentation* is designed to provide an in depth explanation of the Simple SCORM LMS Adapter. This documentation covers the purpose, use, and technical details of the SSLA. For a quick overview to get started using the SSLA see the *Simple SCORM LMS Adapter (SSLA) Quick Start Guide*.

What is the Simple SCORM LMS Adapter?

The Simple SCORM LMS Adapter (SSLA) is a tool designed to quickly make a course SCORM conformant. SSLA collects all SCORM runtime data and provides a mechanism for serializing it to a server. Essentially SSLA gives the user an easy way to handle getting and setting course data with minimal setup.

There are no specific language or database requirements that need to be met in order to use the Simple SCORM LMS Adapter. Most customizations needed to handle a unique Learning Management System can be set in a configuration override file. The most important custom configurations will involve setting up files that determine where on the user's server the JSON data will be sent to their database via Ajax. All of the public properties available for customization are listed under *Configuration Properties*. For a quick overview on how to get started see the *Simple SCORM LMS Adapter (SSLA) Quick Start Guide*. The *How Simple SCORM LMS Adapter* section provides an in depth look at how the SSLA works.

How the Simple SCORM LMS Adapter Works

This section covers the Simple SCORM LMS Adapter technical details and SSLA file workflow. The technical details shown below are the same regardless of whether or not the user setup is for SCORM 1.2 or SCORM 2004. The file workflow takes the SCORM version into account.

Technical Details

This section covers the data transfer between the server side and the client side. To begin with the API is loaded, data from the LMS is sent through the Custom Data Parser (PHP, ASP, ColdFusion) to the local Session Storage on the student's computer. This initializes the data that will be needed to begin the course. If the course has been entered before, this data will be used to return the course data to the state of the previous session.

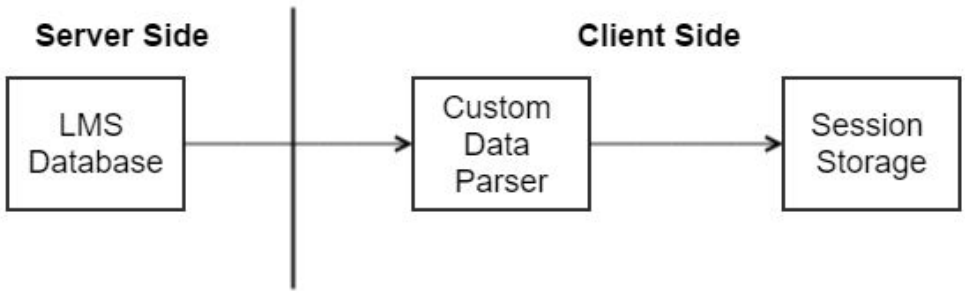

Figure A. On Load Diagram

Once the Session Storage has been populated, an event is triggered to load the course into a window or iframe. This ensures that the course is not launched until the data is available to support it. Once the course is launched, the course issues an

initialization request and makes the connection with the API Adapter. The course can then get and set data to and from the API Adapter.

When data is set from the course to the API Adapter that data is first saved in a local Session Storage on the Client Side then the data is also transferred to the Custom Data Parser and from there to the LMS via form post or form get action. The Custom Data Parser will save the entire contents of the Session Storage to the LMS Database and can also parse the Session Storage data and forward specific pieces of the Session Storage data to the proper fields in the LMS Database. As an example, administrators may want to track grades or status in a separate field. That data can be parsed from the Session Storage and copied into a separate field within the LMS Database.

The course may issue a get request to the API Adapter. The resulting action would be to pull the requested data from the Session Storage on the client side. A get request does not require a connection to the database.

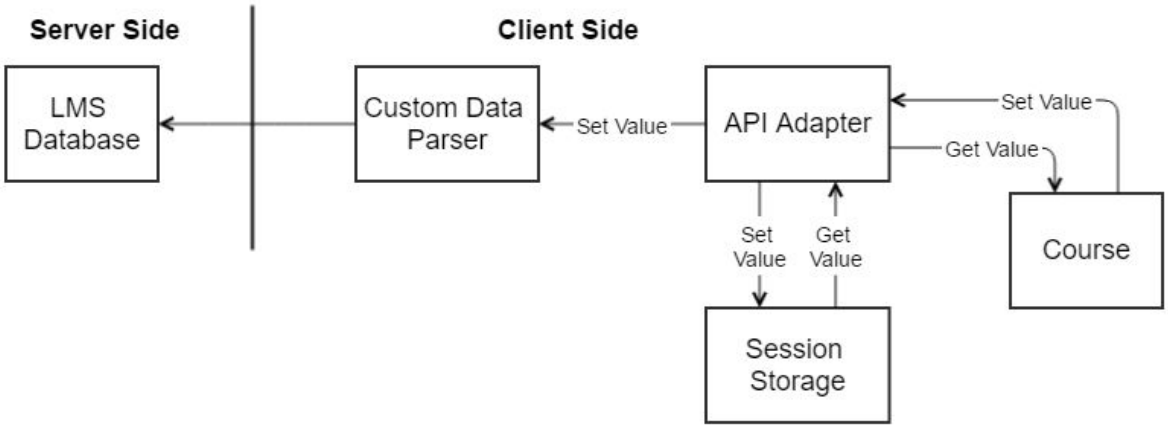

Figure B. Run-Time Diagram

When the course is unloaded the window or frameset that contained the course is closed and the Session Storage data is saved to the LMS database through the Custom Data Parser.

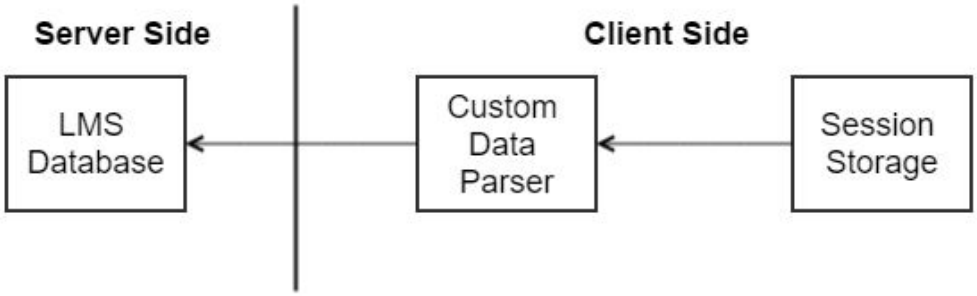

Figure C. On Unload Diagram

SSLA Installation and Course Setup

These instructions show how to setup a basic course with a single SCO. Minimal customization is included to allow debugging, posting data and retrieving data. See the Configuration Properties for a complete list of customization options.

1. Extract the SSLA zip file to a location where you are allowed to make changes. Note: For Windows users make sure you have administrative rights on the directory you choose.

2. Go to the top *ssla* folder. Open the secondary *ssla* folder.

3. Create a new JavaScript file called *config_override.js* in the *ssla* folder. Add the following text to the new file.

```
var sslaConfig = {
   openContentIn: “inline”,
   autoLaunchFirstSco: false,
   singleScoView: ""
};
```

   This custom configuration has basic settings to the SCO from automatically launching, displaying the course within the browser, and display the default content window. For an actual course set and get properties would be added to this configuration file.

   The last two properties that need to be added to the custom configuration are get and set files. Below ***singleScoView*** add ***getDataUrl*** and ***setDataUrl****.* The values for these properties need to be your get and set files. Your get and set files must be created in the language of your server. The example code below shows ***getDataUrl*** and ***setDataUrl*** with values *get.php and set.php* respectively.

```
var sslaConfig = {
   openContentIn: “inline”,
   autoLaunchFirstSco: false,
```

```
    singleScoView: "",
    setDataUrl: "../install/examples/php/set.php",
    getDataUrl: "../install/examples/php/get.php"
};
```

4. This step is optional. If you want your SCOs located in a specific place add a ***courseDirectory*** value to *config_override.js*. For example:

```
var sslaConfig = {
    ...
    courseDirectory: "/my_courses"
}
```

   Keep in mind that omitting the "/" from the *courseDirectory* value will automatically append the *courseBaseHtmlPath* value to the beginning of the path.

5. Save the changes to *config_override.js*.

6. Open the directory specified in *courseDirectory* in *config_override.js* or the directory where you unzip your SCORM packages. See the section on the Courses Directory for more information. Add your SCO folder.

7. Now that you have modified the configuration file to point to your server (***getDataUrl***, ***setDataUrl***) you will need to create pages that can accept web service calls from SSLA. In the above example SSLA would send the data to a page in the local directory called "set.php" and when SSLA needs data it will request it from "get.php". Your files will probably have different names. Example get.php and set.php files are provided under *Code Examples*. An example of the data sent by SSLA can be found under *POST Data Block*.

8. Your ***setDataUrl*** end point will need to be able to accomplish the following minimal tasks:
   a. Select a record from the database
      i. If that record exists then update it. If it does not exist then create it.
   b. Write the SSLA JSON data for that record into the table

9. Your ***getDataUrl*** end point will need to be able to accomplish the following minimal tasks:
   a. Select a record from the database
   b. Read the SSLA JSON data for that record
   c. Return that data back to SSLA

10. Create a URL to launch your particular piece of content. This step explains the basics of a query string creation. The following example shows a breakdown of the values in the string.

    player.htm?courseId=1&studentName=Caudill,Brian&studentId=1&courseDirectory=courses/SSLA_tryout) The query string breaks down as follows

    a. ***player.htm*** – the launch file for SSLA, do not modify
    b. ***courseId*** – your LMS's identifier for the course you are launching
    c. ***studentName*** – the name of the student that is launching this course (Last Name, First Name)
    d. ***studentId*** – the identifier for the student that is launching this course
    e. ***courseDirectory*** – a directory on the server that is accessible by SSLA that contains the unzipped SCORM package so that the SSLA can read the imsmanifest and launch the course

About the Application File Structure

This section covers the folder structure and different application files.

Courses Directory

For the purposes of this documentation the *courses directory* refers to the ***courseDirectory*** value. The ***courseDirectory*** value is set up either in the URL or in config_override.js. This value points to the directory you unzip your SCORM packages to. The *courses directory* contains any course directories the user adds there. Figure E below demonstrates that multiple courses can be placed in the *courses directory*.

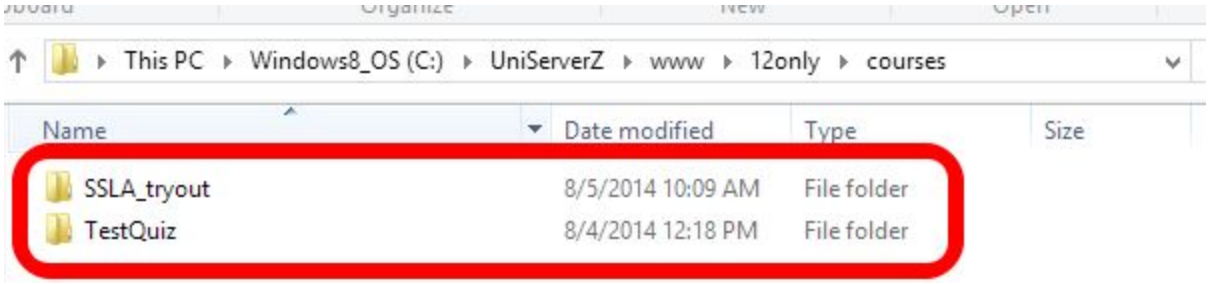

Figure E. Courses directory example

Manifest File

The top level user course folder must contain the file *imsmanifest.xml.* This file is supplied by the user and varies based on the individual course. Figure F shows the location of the *imsmanifest.xml* file for the example SSLA course.

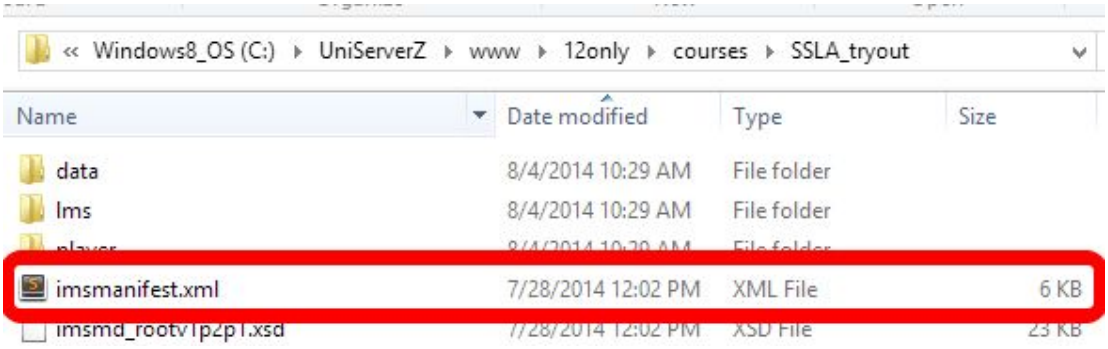

Figure F. Manifest file location example

Customizable Files

A few of the SSLA files can be customized to fit a particular look and feel for the SSLA window. The customizable files will not interfere with the look of a particular course. These files are *config_override.js*, *player.htm* and *interface.css*. The *config_override.js* file allows the user to set public configuration properties. Details about this file are covered in *Course Setup* and *Configuration Properties*. The *player.htm* file can be edited to change the html or add new CSS style changes. The *interface.css* file can be edited to change the style for your SSLA window. Figure G below shows the location for each of these customizable files.

The only restrictions apply to player.htm. There are a few items that must exist even if you choose to hide those items. The required items are listed below:

1) id="frame-here"

2) id="toc"

3) The navigation buttons

4) The body onload/onbeforeunload functions.

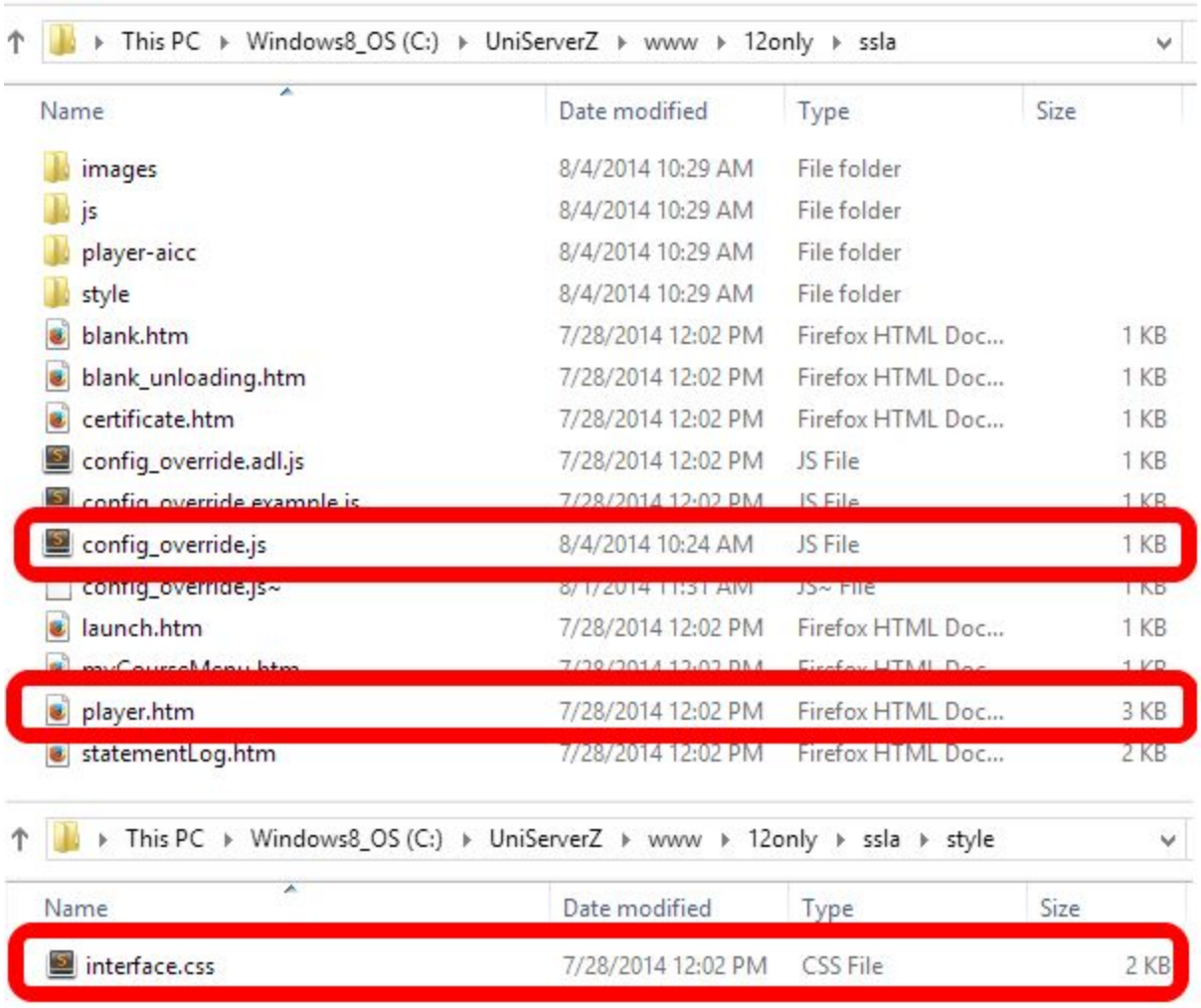

Figure G. Customizable file locations

Placing Your Content on a Different Domain

By default, SSLA must live on the same domain as the content itself. There are two options which allow the content to exist on a domain separate from the LMS. The first solution requires that SSLA be placed on the content domain while the configurable endpoints point to the LMS. The second solution requires the use of another JCA software solution, *SCORM 4 Cross Domain*. Each solution is furthered explained below.

To implement the first solution place SSLA on the content domain. Configure the endpoints ***SetDataUrl*** and ***GetDataUrl*** so that they point to the LMS. To ensure that cross-domain AJAX POSTs can be made successfully CORS needs to be implemented correctly on the LMS domain. While his solution is only guaranteed to work in IE9 and up, it will most likely work for other browsers that are reasonably aged.

The second solution requires the use of *SCORM 4 Cross Domain*. Use *SCORM 4 Cross Domain* to place a stub package on SSLA on the LMS. The stub package can communicate with the cross-domain adapter located on the content domain and can proxy the SCORM commands. *SCORM 4 Cross Domain* is the better option if CORS support cannot be added to the LMS or CORS cannot support the browsers for this particular setup.

Code Examples

The following code examples are provided to elaborate on certain topics. They are not intended for greater user as production code. It is important that you, as the user, ensure any customized code that you use with SSLA meets your own security standards.

This example shows a PHP application writing to a MySQL database, with a table named scorm_data that has a two column primary key on (student_id, activity_id).

get.php

```
<?PHP
    mysql_connect("localhost", "root", "<your_password>") or die("Could not
                    connect: " . mysql_error());
    mysql_select_db("ssla_source");

    $activityId = mysql_real_escape_string($_POST['activityId']);
    $studentId = mysql_real_escape_string($_POST['studentId']);
    $sql = "SELECT data_block FROM scorm_data WHERE student_id =
'".$studentId."' AND activity_id = '".$activityId."'";
    $result = mysql_query($sql);
    $row = mysql_fetch_row($result);
    echo($row[0]);
?>
```

set.php

```
<?PHP
    mysql_connect("localhost", "root", "<your_password>") or die("Could not
                    connect: " . mysql_error());
    mysql_select_db("ssla_source");

    $data = mysql_real_escape_string($_POST['data']);
    $activityId = mysql_real_escape_string($_POST['activityId']);
    $studentId = mysql_real_escape_string($_POST['studentId']);
    $query = "";

    $sql = "REPLACE INTO scorm_data (
                    student_id, activity_id, data_block, created, updated)
                    VALUES ('".$studentId."','".$activityId."',
                    '".$data."',NOW(),NOW())";
?>
```

POST Data Block

In order for SSLA to work properly you must store and return JSON statements. Below is an example POST data block produced by SSLA. The block below contains information about the SCO Simple_SCORM_Packager_Tutorial_SCO and marks the status of the SCO as incomplete.


```
{
	"status":"incomplete",
	"sessionTime":"",
	"currentSco":"Simple_SCORM_Packager_Tutorial_SCO",
	"scos":{
		"Simple_SCORM_Packager_Tutorial_SCO":{
			"status":"incomplete",
			"score":"",
			"sessionTime":"-1",
			"data":{
				"cmi.core.lesson_status":"incomplete",
				"cmi.suspend_data":"",
				"cmi.core.lesson_location":"",
				"cmi.core.total_time":"0000:00:00",
				"cmi.core.entry":"ab-initio",
				"cmi.core.score.raw":"",
				"cmi.core.score.min":"",
				"cmi.core.score.max":"",
```

```
                "cmi.comments":"",
                "cmi.comments_from_lms":"No comment",
                "cmi.student_preference.audio":"",
                "cmi.student_preference.language":"",
                "cmi.student_preference.speed":"",
                "cmi.student_preference.text":"",
                "cmi.core.student_name":"Student",
                "cmi.core.student_id":"1",
                "cmi.launch_data":"",
                "cmi.student_data.mastery_score":"",
                "cmi.student_data.max_time_allowed":"",
                "cmi.student_data.time_limit_action":"",
                "cmi.core.credit":"credit",
                "cmi.core.lesson_mode":"normal",
                "cmi.interactions._count":0,
                "cmi.objectives._count":0,
                "cmi.core.mode":"normal",
                "cmi.core.exit":"suspend",
                "cmi.core.session_time":"0000:00:00.23"
            }
        }
    }
}
```

Configuration Properties

The configuration properties allow for further customization of the SSLA. This document only includes public properties that are available to the end user. These properties can be added to config_override.js. For an explanation of how to create and use config_override.js see *Course Setup*.

Property	Description
autoLaunchFirstSco	Determines whether or not the first SCO begins automatically when the user starts the course. Accepts a boolean:: true: Default value, SCO automatically launches false: First SCO will not start until selected
courseBaseHtmlPath	Changes the base HTML path for the course from the default string "../". Accepts a string value.
closePopupSingleScoBehavior	Set the behavior a popup with a single SCO has when the user closes the popup. Accepts a string and defaults to "exit": "exit": exits the course "custom": runs the behavior set in ***closePopupSingleScoCustomFunction()*** "": empty does nothing
closePopupMultiScoBehavior	Set the behavior a popup with multiple SCOs has when the user closes the popup. Accepts a string and defaults to "" (empty): "exit": exits the course "custom": runs the behavior set in ***closePopupMultiScoCustomFunction()*** "": empty does nothing
closePopupSingleScoCustomFunction	Defines the custom behavior for closing a popup with a single SCO.
closePopupMultiScoCustomFunction	Defines the custom behavior for closing a popup with multiple SCOs.
courseId	Forces the courseId for a specific course to be run. Accepts a string with the courseId and defaults to the value pulled from the URL.
courseDirectory	Changes the default course directory from "". If the new course directory does not begin with the character "/" then the new course directory value will be amended to become "courseBaseHtmlPath + courseDirectory". Accepts a string value.
exitAction	Defines the exit behavior for the course. Defaults to 'referrer'. "none": does nothing

	“close”: close any popups that are available “referrer”: redirects the window to the HTTP referrer “url”: redirects the window to the config property specified in ***exitUrl()*** “custom”: runs the funciton specified in e***xitFn()***
exitFn	If *exitAction* is set to “custom”, calls the function pointed to by this configuration option on exit.
exitUrl	Defines the URL to redirect the window to after exiting a course.
getCredit	Set whether or not the course provides credit. Accepts a string and defaults to “credit”: “credit”: The course provides credit “no-credit”: The course does not provide credit
getDataAjaxMethod	Accepts a string with any valid HTTP verb. Default value is “POST”.
getDataHeaders	JavaScript Object containing key-value pairs that could be used in any HTTP request. Custom headers that the server may use. Pairs are written like this: {"X-Auth-Token": "your auth token here"}
getDataUrl	Path to your custom script that returns JSON statements to the SSLA. Accepts a string. Defaults to “"/trackAllInOne/get".
getMode	Changes the mode that the user views the course in. Accepts a string and defaults to the value pulled from the URL: “browse”: sets course to browse mode “normal”: course is viewed as normal “review”: sets course for review
layoutUpdater	Provide a custom function to perform layout updates based on the system state, including information about the course itself (single SCO vs. multi SCO, for example). Expected function signature is single argument that can take an SSLA object.
navigationType	Set the type of navigation for the SCOs. Accepts a string: “”: Defaults to the manifest’s behavior “lockstep”: SCOs must be taken in order “examaftercontent”: All content must be taken before entering the exam “onlyonce”: Every SCO is only accessible once
openContentIn	Determines where the content displays. Accepts a string: “inline”: Opens inside the main player window (default) “popup”: Opens in a popup window
popupMainContentMessageAfterOpen	Sets a custom message displayed when the popup opens. Accepts a string.
popupMainContentMessageFailed	Sets a custom message displayed when the popup fails to open. Accepts a string.
popupWindowParams	Adds custom properties to the window.open() call. Accepts a string with valid values for the the third

	argument in window.open(). For example, “width=200,height=200”.
returnToLastScoOnLaunch	Set whether or not the user returns to the first SCO when reopening the course or the last SCO they viewed. Accepts a boolean and defaults to false: false: Returns to the first SCO true: Returns to the last SCO viewed
saveDataOnCommit	Set whether or not data is saved when the commit call is made. Accepts a boolean and defaults to true: false: Does not save data on commit true: Saves data on commit
saveDataOnSetValue	Set whether or not the data is saved every time the set call is made. This increases the number of times the data is saved. Accepts a boolean and defaults to false: false: Does not save data on set true: Saves data on set
scoreAllowChangeAfterCompleted	Allows the score to be changed after the lesson has been marked as completed. Defaults to true. true: allows the score to be changed false: blocks the score from being changed
scoreAllowChangeAfterFailed	Allows the score to be changed after the lesson has been marked as failed. Defaults to true. true: allows the score to be changed false: blocks the score from being changed
scoreAllowChangeAfterPassed	Allows the score to be changed after the lesson has been marked as passed. Defaults to true. true: allows the score to be changed false: blocks the score from being changed
scoreAllowReduce	Allows the score to be reduced. Setting this to false blocks the score from being set to a lower value even if the score is allowed to be changed. Defaults to true. true: allows the score to be set lower than its current value false: blocks the score from being set lower than its current value
scoreRollupIncludeScosWithNoScore	(boolean) Should SCOs that have no specified score contribute to the score rollup that’s sent to the server? Only applies to SCORM 1.2 or SCORM 2004 that doesn’t have a manifest with specified rollup rules. Default: false
scoreRollupCustomFunction	Provides a custom function for doing custom score rollup logic.
scormManifestFilename	Changes the manifest filename. Accepts a string and defaults to “imsmanifest.xml”.
scormManifestRawXml	Manifest to use in place of the existing SCORM manifest file. Accepts a string.
scormManifestUseRawXml	Determines whether or not to use scormManifestRawXML. Defaults to false. true: use scormManifestRawXML for the manifest XML body

	false: load the XML from the file referenced by scormManifestFilename
setDataAjaxMethod	Accepts a string with any valid HTTP verb. Default value “POST”.
setDataHeaders	JavaScript Object containing key-value pairs that could be used in any HTTP request. Custom headers that the server may use. Pairs are written like this: {"X-Auth-Token": "your auth token here"}
setDataUrl	Path to your custom script that stores JSON statements. Accepts a string. Defaults to “"/trackAllInOne/set".
singleScoView	Change what the user sees when there is only one SCO: "": Leave it however it's currently displaying. "HIDE_ALL": Hide all non-course view components. "HIDE_NAV_BUTTONS": Hide the navigation menu. "HIDE_TREE": Hide the navigation tree.
statusAllowChangeAfterCompleted	Determines whether or not the course status is allowed to change after the course already has a completed status. Defaults to true. true: the course status can be changed even if it already has a completed status false: the course status cannot be changed if it already has a completed status
statusAllowChangeAfterFailed	Determines whether or not the course status is allowed to change after the course already has a failed status. Defaults to true. true: the course status can be changed even if it already has a failed status false: the course status cannot be changed if it already has a failed status
statusAllowChangeAfterPassed	Determines whether or not the course status is allowed to change after the course already has a passed status. Defaults to true. true: the course status can be changed even if it already has a passed status false: the course status cannot be changed if it already has a passed status
statusRollupCustomFunction	Defines a custom status rollup function.
storageAdapter	Replaces the current AllInOneStorage with a completely separate storage class, if needed for customization. Defaults to the built-in storage solution.
storageAllowCORS	(boolean) Adds CORS attributes to the AJAX calls for storage requests.
storageUseWithCredentials	(boolean) Adds “withCredentials” to AJAX calls for storage requests.
studentId	Forces the studentId for a specific student to be run. Accepts a string with the studentId and defaults to the value pulled from the URL.

studentName	Forces the studentName for a specific student's name to be run. Accepts a string with the studentName and defaults to the value pulled from the URL.

FAQ

Q: Is SSLA a client-side JavaScript run-time environment for SCORM packages?

A: Yes.

Q: Do I embed the provided SSLA artifacts (ssla.min.js, 3rd.js, ...) in the page that starts the SCORM session (e.g. Launch.htm)?

A: Yes. You'll need to preserve the HTML artifacts as-named from player.htm distributed with the SSLA package, but you can add and change whatever you want beyond that.

Q: The adapter requires JQuery and I have to resolve that dependency, right?

A: The adapter does not require JQuery. Everything it requires is already bundled in 3rd.js.

Q: Does SSLA send Ajax calls to the server?

A: Yes.

Q: Does GET fetch the user's entries out of an existing SCORM session?

A: Yes.

Q: Does SET post the user's current SCORM session? How often (per user session) does this happen?

A: Yes. Whenever the SCORM content calls the API function "LMSCommit()". We have no control of this, it happens based on the way the content is constructed. It also calls when the course is being exited and/or the browser window is being closed.

Q: Are the URLs for GET/SET customizable in the config_override.js properties ***setDataURL*** and ***getDataUrl***?

A: Yes.