Rustici SCORM Adapter

Full Documentation

Version 5.3.0

Table of Contents

Introduction

Rustici SCORM Adapter Full Documentation is designed to provide an in-depth explanation of our client-side tool that helps make any LMS conformant with SCORM 1.2 and 2004 3rd and 4th editions. This documentation covers the purpose, use, and technical details of Rustici SCORM Adapter.

What is Rustici SCORM Adapter?

Rustici SCORM Adapter is a tool designed to enable a web application to play SCORM courses and track learner results. It collects all SCORM runtime data and provides a mechanism for serializing it to a server. Essentially, Rustici SCORM Adapter gives the user an easy way to handle getting and setting course data with minimal setup.

Rustici SCORM Adapter is delivered as a JavaScript library that will parse the SCORM XML files, create a table of contents for the course, launch the course, and track all the SCORM user data that is sent from the course to the LMS. Adding Rustici SCORM Adapter to your LMS will make your LMS is SCORM 1.2 and 2004 (3rd and 4th editions) runtime conformant.

There are no specific language or database requirements that need to be met in order to use Rustici SCORM Adapter. Most customizations needed to handle a unique LMS 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*.

How Rustici SCORM Adapter Works

This section covers Rustici SCORM Adapter technical details and Rustici SCORM Adapter 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 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.

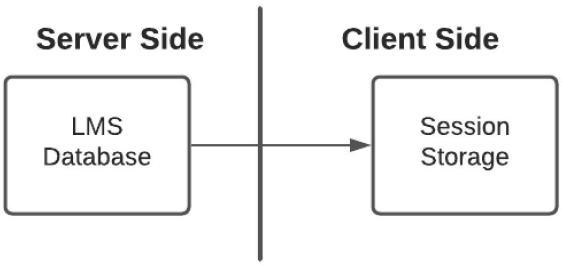

Figure A. On Load Diagram

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 transferred to the

LMS via form post or form get action. The entire contents of the Session Storage are saved to the LMS Database. It is also possible to 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.

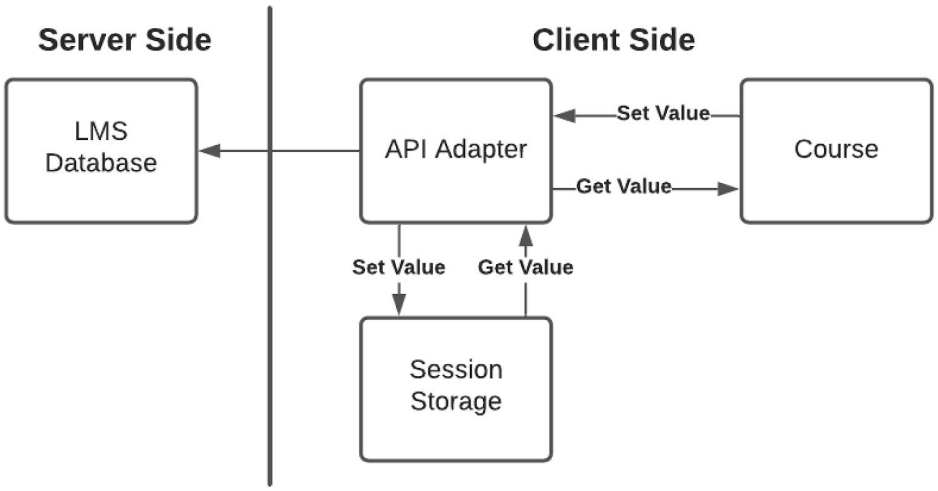

Figure B. Run-Time Diagram

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

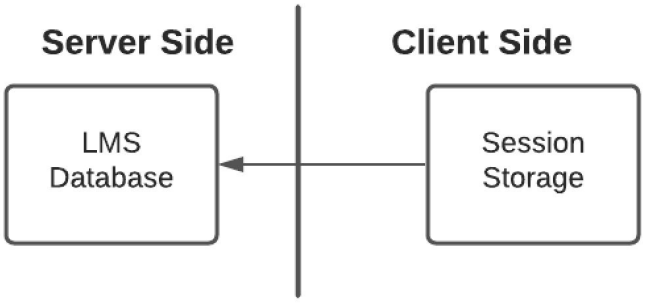

Figure C. On Unload Diagram

Installation and Course Setup

These instructions show how to set up 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.

Please Note Name Change: Rustici SCORM Adapter was previously called Simple SCORM LMS Adapter (SSLA). There are places in the code that reference the old name. SSLA appears as the name of folders, in code snippets, etc. Some references of the name SSLA will remain in the code and documentation directions beyond this point.

1. Extract your Rustici SCORM Adapter 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 displaying 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. This step is optional. Create a database table for storing your records. The exact fields depend on your system. See *Code Examples* for a basic data table setup.

7. 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.

8. 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 Rustici SCORM Adapter. In the above example Rustici SCORM Adapter would send the data to a page in the local directory called “set.php” and when Rustici SCORM Adapter 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 Rustici SCORM Adapter can be found under *POST Data Block*.

9. 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 JSON data for that record into the table

10. Your *getDataUrl* endpoint will need to be able to accomplish the following minimal tasks:
    a. Select a record from the database
    b. Read the JSON data for the record
    c. Return that data back to Rustici SCORM Adapter

11. 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, 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 Rustici SCORM Adapter that contains the unzipped SCORM package so that the software 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 D below demonstrates that multiple courses can be placed in the *courses directory*.

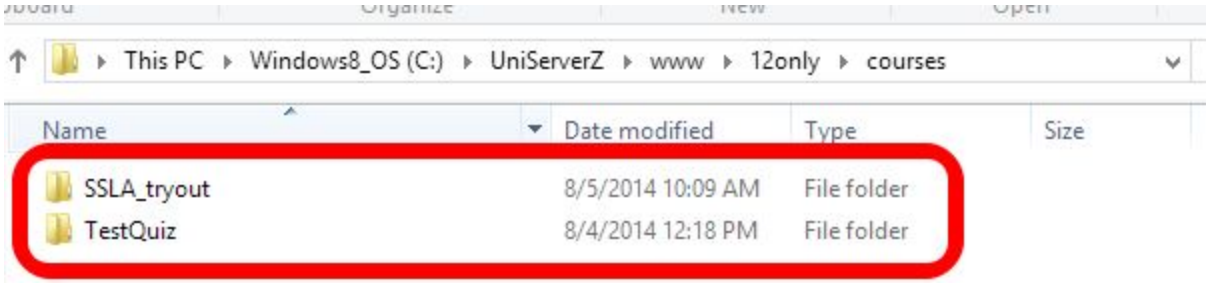

Figure D. Courses directory example

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 E. shows the location of the *imsmanifest.xml* file for the example course.

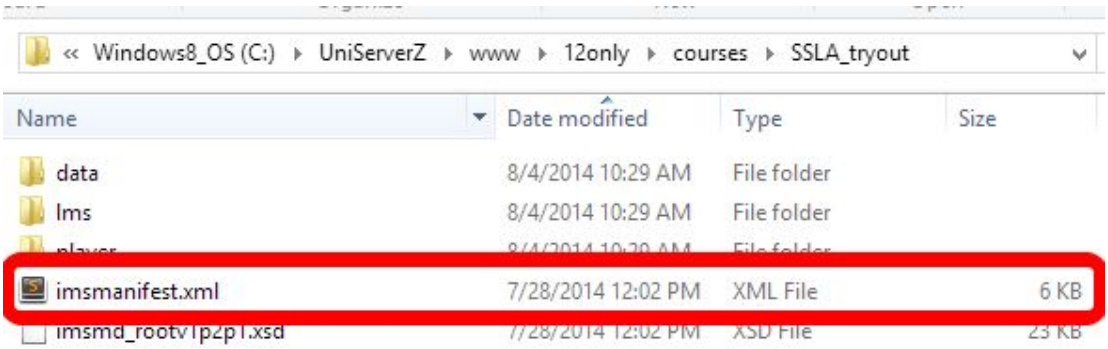

Figure E. Manifest file location example

Customizable Files

A few of the files can be customized to fit a particular look and feel for the Rustici SCORM Adapter 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 application window. Figure F. 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.

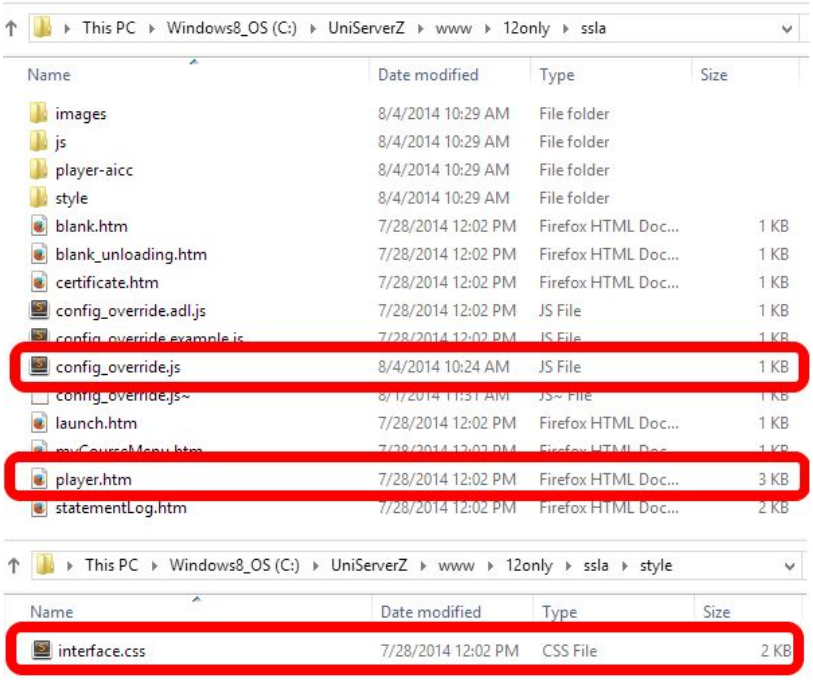

Figure F. Customizable file locations

Placing Your Content on a Different Domain

By default, Rustici SCORM Adapter must live on the same domain as the content itself. There is one option that allows the content to exist on a domain separate from the LMS. That solution requires that Rustici SCORM Adapter be placed on the content domain while the configurable endpoints point to the LMS.

To implement this solution, place Rustici SCORM Adapter 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. This solution will work for all modern browsers.

Code Examples

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

A minimal table for storing SCORM data using MySQL is shown here. This table will work with the PHP application example.

```
create table scorm_data (id int not null auto_increment primary key, student_id
varchar(100) NOT NULL, course_id varchar(100) NOT NULL, data longtext, created
datetime NOT NULL, updated datetime NOT NULL, UNIQUE KEY(course_id,
student_id));
```

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
    $db = mysqli_connect("127.0.0.1", "root", "mypass", "ssla");
    if (!$db) {
        echo "Error - Failed to connect to MySQL: " . mysqli_connect_errno() .
": " . mysqli_connect_error();
        die();
    }
    $course_id = $_POST['activityId'];
    $student_id = $_POST['studentId'];
    $sql = "SELECT data FROM scorm_data WHERE student_id = '".$student_id."'
AND course_id = '".$course_id."'";
    if ($result = mysqli_query($db, $sql)) {
        $row = $result->fetch_row();
        echo($row[0]);
    }
?>
```

set.php

```
<?PHP
    $db = mysqli_connect("127.0.0.1", "root", "mypass", "ssla");
    if (!$db) {
        echo "Error - Failed to connect to MySQL: " . mysqli_connect_errno() .
": " . mysqli_connect_error();
        die();
    }
    $course_id = $_POST['activityId'];
    $student_id = $_POST['studentId'];
    $data = $_POST['data'];
    print_r($course_id);
    print_r($student_id);
    print_r($data);
    $stmt = $db->prepare("REPLACE INTO scorm_data (student_id, course_id, data,
updated) VALUES (?, ?, ?, NOW())");
    $stmt->bind_param("sss", $student_id, $course_id, $data);
    $stmt->execute();
    $stmt->close();
?>
```

POST Data Block

In order for Rustici SCORM Adapter to work properly you must store and return JSON statements. Below is an example POST data block produced by the application. 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 Rustici SCORM Adapter. 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	(boolean) Determines whether or not the first SCO begins automatically when the user starts the course. true: Default value, SCO automatically launches false: First SCO will not start until selected
closePopupSingleScoBehavior	When using popup mode on single SCO packages, set the behavior 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	When using popup mode on multiple SCO packages, set the behavior 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	When using popup mode on single SCO packages, defines the custom behavior upon closing a popup.
closePopupMultiScoCustomFunction	When using popup mode on multiple SCO packages, defines the custom behavior upon closing a popup.
cmi5ManifestFilename	Sets the manifest filename for a cmi5 course. The default value is "cmi5.xml". Accepts a string.
courseBaseHtmlPath	Changes the base HTML path for the course from the default string "../". Accepts a string value.
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.
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.
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 function 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.
frameDefaultUrl	When the frame is launched for the first time, the asset that is loaded within the url. Added in 5.0.
frameId	The DOM element ID for the iframe where SCOs are loaded. Added in 5.2.0.
frameInterstitialUrl	The domain-relative path from the player that provides the interstitial page loaded between SCOs. Added in 5.0.
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 Rustici SCORM Adapter. 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

initializeScoAtOpen	If true, will call the appropriate SCORM version initialization call as soon as the SCO has started opening. Added in 5.1
launchDataAllowOverride	(function) A custom function that accepts a string (the SCO identifier) and returns a boolean to allow/disallow using the launchDataOverride value. Defaults to false. Format: launchDataAllowOverride(string) {return boolean} Added in 5.0
launchDataOverride	(function) A custom function that accepts a string (the SCO identifier) and returns a string with override data based on a given SCO ID. Specifically, the cmi.launch_data runtime property will be overwritten with the string provided for the specific SCO. Defaults to an empty string. Format: launchDataOverride(string) {return string} Added in 5.0
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 a Rustici SCORM Adapter object.
learningSpecification	Controls which learning specification will be attempted to initialize the course. Not all specifications may be included in your build, please consult your purchase documents. See the section on *Extension Modules* for more information. Accepts a string and defaults to "" (empty): "": Attempts to detect all available specifications "scorm": Attempts load of only SCORM 1.2 / 2004. "xapi": Attempts load of only xAPI. "cmi5": Attempts load of only cmi5. "aicc": Attempts load of only AICC.
logMessageBehavior	(string) Controls how Rustici SCORM Adapter informational/error messages are presented to learners. Defaults to "alertErrorOnly". "console": Only shows messages in the browser console. "alert": Only shows messages as browser alerts. "alertErrorOnly": Show error-level messages as browser alerts, all other messages in the browser console. "custom": If logMessageCustomFunction is specified, process messages through it.

	Added in 5.2.0.
logMessageCustomFunction	(function) Accepts a custom function for Rustici SCORM Adapter informational/error messages to be piped through for processing and display. Defaults to null.
looseIdentifiers	Allows a loose identifier validation instead of the more precise and stricter CMIIdentifier definition (255 characters, only alphanumeric, underscore and dashes). Accepts a boolean and defaults to false: false: Strict identification true: Loose identification
masteryScoreMode	Decides the behavior of the mastery score if there is one. Accepts a string and defaults to "after_completion": "none" or "" (empty): ignore the mastery score altogether "always": apply mastery score on finish if it exists, in all cases where a cmi.core.score.raw is set to an actual value. "after_completion": apply mastery score on finish only if cmi.core.lesson_status is "completed" (or the equivalent version of this in SCORM 2004)
modeBrowseAllowSave	When mode="browse", determines whether a storage call is made on Commit/Finish/Terminate. Accepts a boolean and defaults to false. true: Storage calls will be sent to backend. false: Storage calls will not be sent. Added in 5.1.
modeReviewAllowSave	When mode="review", determines whether a storage call is made on Commit/Finish/Terminate. Accepts a boolean and defaults to true. true: Storage calls will be sent to backend. false: Storage calls will not be sent. Added in 5.1.
multiScoView	Change what the user sees when there are multiple SCOs. Defaults to "" (empty). Accepts a string: "": 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.
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.
popupName	Provides the name used for the content popup window. Override this if you intend on having multiple Rustici SCORM Adapter instances open at once, and need the popups to be uniquely named. Added in 5.2.0.
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".
reenterActivity	Allows re-entrance to an activity after it has been initialized and before it has been terminated. Accepts a boolean. Defaults to true: true: Allow re-entry false: Block re-entry
returnToLastActivityOnLaunch	Returns the student to the last activity they were in the next time they launch the course. Accepts a boolean. Defaults to false: true: Returns the student to the last activity false: Sends the student to the start of the course
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
scoreConvertNormalizedToRawMethod	(string) When a course only provides a normalized value this option determines a raw value for storage output. The raw value is based on the range set by cmi.score.max and cmi.score.min. The option set in the configuration changes the conversion mode. Defaults to 'same_sign'. Note: A min and max must be set to provide a range for conversion. same_sign: Returns a raw score that has the same sign as the normalized score. This means the min/max range is forced to stay within the positive or negative range. For example, given a normalized value of -0.5 with a minimum of -1 and a maximum of 1, the raw value is -50. This is because the range is changed to only negative values. So the maximum becomes 0. A normalized score of 0 always converts to a raw score of 0. full_avg: The normalized score is converted to the raw value within the set min/max range. Added in 5.0
scoreConvertRawToNormalizedMethod	(string) When a course only provides a raw value this option determines a normalized value for storage output. The normalized value is based on the range set by cmi.score.max and cmi.score.min. The option set in the configuration changes the conversion mode. Defaults to 'same_sign'. Note: A min and max must be set to provide a range for conversion. same_sign: Returns a normalized score that has the same sign as the raw score. This means the

	min/max range is forced to stay within the positive or negative range. For example, given a raw value of 50 with a minimum of -30 and a maximum of 100, the normalized value is .5. This is because "same_sign" only uses the positive range. So the minimum becomes zero and the maximum stays the same. A raw score of zero is always converted to a normalized value of 0. full_avg: The raw score is converted to the normalized value within the set min/max range. Added in 5.0
scoreRollupAction	(string) Determines which rolled up SCO score(s) to save. first: Sets the first SCO's score as the rolled up score data. Defaults to an empty string. last: Sets the last SCO's score as the rolled up score data. "": Takes the average of all the SCO scores and sets that as the rolled up score data. Added in 5.0
scoreRollupCustomFunction	(function) Provides a custom function for doing custom score rollup logic. Defaults to null. Format: scoreRollupCustomFunction() { return any } Added in 5.0
scoreRollupIncludeScosWithNoScore	(boolean) When performing score rollup for "all_equal" mode, if false, only average SCO scores that exist and are not empty. If true then empty scores will be treated as zero values. This only applies to SCORM 1.2 or SCORM 2004 courses that don't have a manifest with specified rollup rules. Default: false
scorm2004Mode	(choice: "simple", "full") Determines whether SCORM 2004 content should be played with a full sequencing and navigation experience ("full") or with a simplified experience that matches the SCORM 1.2 behaviors and provides full runtime support for the data model ("simple"). Only enable "full" when your content is assuredly completely conformant with all portions of the SCORM 2004 standard. Default: "simple" Added in 5.0
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
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.
startIncomplete	(boolean) When true, forces the lesson status to incomplete if the status hasn’t already been initialized, otherwise defaults to ‘not_attempted’. true: lesson status set to incomplete when it hasn’t been initialized. false: leaves the lesson status as not_attempted Added in 5.0
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

statusAllowCompletionOnUnsetFinish	(boolean) Determines the behavior when a course is initialized but the status is never set before leaving the course. Defaults to true. true: the course will be marked as completed false: Rustici SCORM Adapter allows for the status to remain unchanged Added in 5.0
statusRollupAction	(string) Sets the rollup action for the SCO statuses. Defaults to an empty string with the rollup including all SCOs. Defaults to an empty string. "": Rustici SCORM Adapter determines the appropriate status to save given the status of each SCO first: uses the status of the first SCO last: uses the status of the last SCO custom: expects statusRollupCustomFunction to define the rollup behavior Added in 5.0
statusRollupCustomFunction	(function) Defines a custom status rollup function. Defaults to null. Format: statusRollupCustomFunction() {return any } Added in 5.0
statusVisualOverride	Sets which function is used to override the visual status shown. The default function's priority from greatest to least is failed, passed, completed, browsed, incomplete, and not attempted. Accepts a string. Defaults to "default": "default": uses the built in default function "custom": calls statusVisualOverrideFn
statusVisualOverrideFn	Allows a custom function to be created for the visual override priorities. Accepts a function. Defaults to null.
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.
storageCallbackAfterLoad	(function) Provides a callback that allows you to receive an object after the load request is completed, but before Rustici SCORM Adapter uses the data to launch the learner experience. If the function returns a value, it will be used to replace the original value from the callback. Added in 5.2.0.

storageCallbackAfterSave	(function) Provides a callback that allows you to receive an object after the save request has completed (successfully or errored). This fires after all Rustici SCORM Adapter save storage operations are completed. Added in 5.2.0.
storageGzipRequestBody	(boolean) When true, apply gzip encoding to the request body of any save call and an appropriate matching Content-Encoding header. It will be necessary for the server to decode this, but post-decoding is the same storage format as defined elsewhere in the document. Default: false Added in 5.3.0.
storageIncludeScoDetails	(boolean) Provides some structural details, including name and ID, for each SCO. Defaults to false. true: includes the SCO structural details false: leaves out the SCO structural details Added in 5.0
storageQueueSaves	(boolean) If true, if a save request is already active, queue any subsequent save requests and flatten them into a single request of the newest changes. Otherwise, fire save requests as soon as they are sent to the adapter. Defaults to false. Added in 5.2.0.
storageSaveDuplicateCommits	(boolean) If false, the storage adapter will attempt to identify requests where no “substantive” data has changed (things not relative to timestamps), and will not send a request to the backend if so. Defaults to true.
storageSaveFrequency	(number) Polling interval in milliseconds where any delta in generated learner data before a save call is issued (whether by SetValue or other SCORM activity), whether or not a SCORM Commit is referenced. If set to a value <= 0, saves will only occur on an explicit commit. Default: 10000 Added in 5.3.0.
storageTimeout	(number) Number of milliseconds before a save call times out. Defaults to 30000. Added in 5.2.0.

storageUseFetch	(boolean) When true, will use the Fetch/Beacon API provided by some browsers for storage calls. Otherwise, this will fall back to using XMLHTTPRequest-based storage. Provides support for Chrome version 80 onward, deprecations surrounding synchronous AJAX on unload. Defaults to true for Chrome, else false. true: use Fetch/Beacon API false: use request based storage Added in 5.0
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.
validateDataModelLength	(boolean) Allows Rustici SCORM Adapter to ignore the SCORM data length limitations. Defaults to true. false: Rustici SCORM Adapter ignores the SCORM data length limitations. For example, cmi.suspend_data would not be limited to 4K characters in SCORM 1.2 or 64K characters in SCORM 2004. true: Rustici SCORM Adapter enforces the SCORM data length limitations. Added in 5.0
xapiActor (available in xAPI plugin)	(string) Sets the xAPI actor. Defaults to the value passed in as query parameter "actor".
xapiAuth (available in xAPI plugin)	(string) Sets the xAPI authentication credentials. Defaults to the value passed in as query parameter "auth".
xapiEndpoint (available in xAPI plugin)	(string) Sets the xAPI endpoint. Defaults to the value passed in as query parameter "endpoint".
xapiManifestFilename (available in xAPI plugin)	(string) Sets the xAPI manifest filename. Defaults to "tincan.xml".
xapiRegistration (available in xAPI plugin)	(string) Sets the xAPI registration. Defaults to the value passed in as query parameter "registration".

Frequently asked questions

Will my LMS need a great deal of refactoring?

This answer is always “it depends”. Usually, the LMS does not require much refactoring because Rustici SCORM Adapter is a JavaScript wrapper that works externally to your existing LMS. All that you will need to do is provide a mechanism to save data, usually a database and some server side page that AJAX requests can be sent to.

What server languages does Rustici SCORM Adapter work with?

Any of them. Rustici SCORM Adapter is written in Javascript and uses AJAX to send data to a file on your server written specifically to accept data from Rustici SCORM Adapter. With this set up the Rustici SCORM Adapter code resides on your server and it does not matter what programming language(s) you use on your server. All output from the environment is in JSON format which all server side languages have a JSON to string and JSON to object method for reading key value pairs so you can use that to extract any scorm data from the JSON file that you would like to report in other tables in your database, things like score, status, and other relevant data.

Is Rustici SCORM Adapter SCORM 1.2 and SCORM 2004 complaint or just SCORM 1.2?

Rustici SCORM Adapter supports SCORM 1.2 and SCORM 2004 (3rd and 4th editions)

Does Rustici SCORM Adapter support assessment interactions?

Yes, Rustici SCORM Adapter is fully SCORM 1.2 conformant so it supports all the mandatory and optional elements. This is also true for the version support in SCORM 2004 (3rd and 4th editions).

Will it run all SCORM packages? Custom built in HTML or from any authoring tool like Articulate, Lectora and Captivate?

Yes, running SCORM packages is what it was made for. As long as the package is SCORM conformant and Rustici SCORM Adapter can be pointed to the imsmanifest it will run.

Will the end users of our system (learner side) require any specific plugins to run courses on PC/Mac? Like JVM etc?

No, the system is written in JavaScript. There is no plugin required to run JavaScript code.

Will Rustici SCORM Adapter work fine for tracking data on an iPad/Android tablet when launched in a mobile browser?

Yes.

Is there functionality to create courses in the API?

No Rustici SCORM Adapter only "plays" courses. It does not create them.

Support

Have questions about Rustici SCORM Adapter or need technical support? Please contact us at support@rusticisoftware.com.