TEACH ON MARS

WEB ACTIVITY / SCORM
TEACH ON MARS - MOBILE LEARNING APPS

© 2019 Teach on Mars

Documentation 3

Documentation

Introduction

Teach on Mars offers a wide variety of mobile learning activities including courses, quizzes and games, and we regularly create new ones… but you may want more, right now! Good news, it is possible! With Teach on Mars, you can import a web content in a Teach on Mars training course just like any other activity.

This can be useful for various purposes:

- Reuse a web content that you already own (from a previous LMS, for instance)
- Create a brand new type of activity (e.g. including VR or AR or a new game)

How can you create such a *web activity*? Simply use one of the various authoring tools available on the market. If you don't know any, try to look for "mobile learning authoring tools" on your favorite search engine.

Usage data like progress, score, success and time spent in your web activity can be sent to the Mission Center and added to the dashboards and exports. There are two ways to do this:
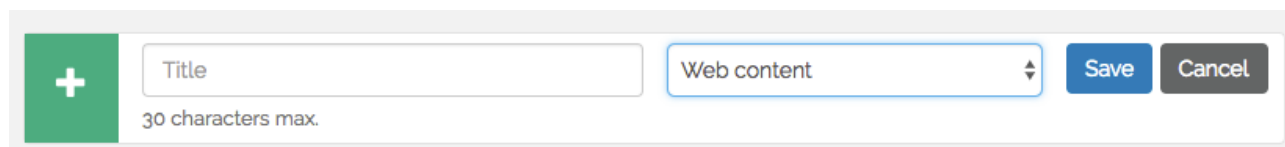
- Use the ToM JavaScript library in your web content
- Use a SCORM compatible content (1.2 and 2004 are supported)

This document is intended to content creators and curators who wish to integrate web contents in the Teach on Mars solution.
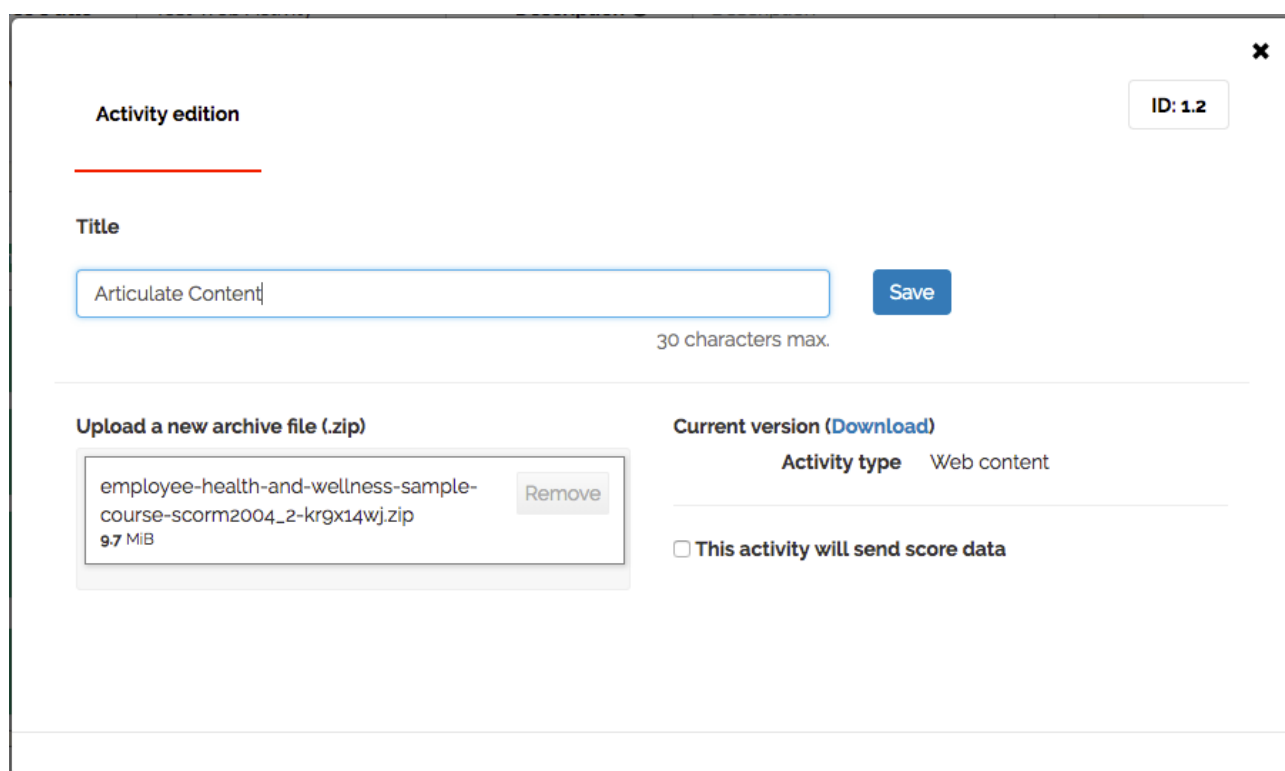
Web activity

How to import a web activity

To import a web activity, just go to the authoring section of a Training Course. Then create a *Web content* activity.

In the activity edition popup, drop your web content archive (.zip) in the drop zone. Be careful, the whole training course cannot weight more than 200Mo. So if your training course only has one web activity, it cannot weight more than 200Mo.

If your web content is supposed to send score and success data*, check the option "*This activity will send score data*". This activity will then appear in the results dashboard in your Mission Center. (*for more information, please refer to the "Web activity creation guide" section below)

That's it! As for any other activity, your web activity will be available in the application as soon as you publish your training course on your Mission Center.

Web activities have a special way of handling translations which differs from other activities. The uploaded web content is common to all languages in the training course. It was way too heavy to upload pretty much the same content for each language, especially when there are a lot of media (video, audio or images). So it has been decided to let the content creator deal with the translations inside the content.

General behavior of a web activity

A web activity behaves pretty much like any standard Teach on Mars activity. So here are some key concepts that are noticeable in Teach on Mars activities, which also apply to web activities:

- A web activity is part of a module, which is itself part of a training course.
- A web activity is downloaded to the application with the whole training course.
- A web activity works offline (unless some elements are not in the content but rather linked or referenced from the Internet, like a streamed video).
- Any data generated by the content (time, progress, score, etc.) will be stored in the device until an Internet connection is available to send the data to the Mission Center.

But there are some rules that apply only to web activities:

- If the device goes to sleep mode while an activity is opened, the activity keeps running but the application will limit the time that the content sends to the time during which the activity was actually active.
- A web activity is displayed in an in-app browser in fullscreen. No button is provided on the in-app browser to close the activity, so the content must be responsible to close the window (we will see later how to do it).

Beforehand, things to consider

1. Make sure your content is mobile compatible and check:

- The navigation (responsive design and buttons adapted to the users)
- The content must provide a way out for the user
- The user experience (simple and user-friendly)
- The content must weigh less than 200 Mo

2. International deployment

To prevent duplication of heavy files like media, the content is shared between languages in the web activity. As a result, the content must deal with its translations (more details later).

3. Recommended steps to integrating a SCORM content

1. Forward this document to the content provider
2. Invite the provider to join the Teach on Mars Integration Community (contact your project manager for more information)
3. Test the content on SCORM cloud: **http://cloud.scorm.com**
4. Test the content in your own Mission Center through a web activity in a published training course. In your application, check the usability and back in the Mission Center check the consistency of the statistics with SCORM cloud.
5. If the data is inconsistent, feel free to contact the Teach on Mars support team.

4. Identify what data you need to track in the Mission Center

Time spent? Progress? Content launches count? Score? Success ?
Fine progress measure (34%, 72%) or a simpler one (0%, 50% or 100%) ?

We activity creation guide

To SCORM or not to SCORM?

If you don't use a SCORM content creation tool and don't need a SCORM compatible content, the Teach on Mars (ToM) JavaScript library may be the best choice. You are free to hire any web oriented agency to create a custom content and have them use the ToM JavaScript library, which is very simple and adapted to the Mission Center dashboards.

If you already use a SCORM content creation tool or already have SCORM contents, they will work as they are as long as they are compatible with a mobile screen resolution.

Just make sure that your don't use both options at the same time.

The table below shows a difference between using the ToM JS Library and SCORM:

Feature	ToM JS library	SCORM (in Teach on Mars)
Content works offline	Yes	Yes
Basic data is tracked (progress, score, success, time)	Yes	Yes
Ranking with points is enabled (in the app and in the Mission Center)	Yes	No*
Sync data between devices	Yes	Yes
Access to learner basic information (ID, name)	Yes	Yes
Access to learner advanced information (custom fields, email, etc.)	Yes	No**
Compatibility with other LMS	No	Yes

The SCORM model does not provide a way to gather points that will be cumulated and used for a ranking feature. So any SCORM activity will not participate to the learners' ranking whereas web activities using the ToM JS Library might.

*** The SCORM model only provides a way to read the learner's name and ID. Any other information about the learner (like custom field values) will require the usage of ToM JS Library.*

Teach on Mars web activity creation guide

This section is intended to web content creators. It will cover the simple rules that must be applied to the content structure and the usage of the ToM JavaScript library. So brace yourselves, it's going to be a little technical.

Basic rules

- The content must be in a *.zip* archive when uploaded to the Mission Center
- The entry point of the web activity must be a *index.htm* file located at the root directory of the content.
- The content must be responsive (it might be displayed on both mobile applications and desktop web applications)
- Try not to rely on resources on the Internet. The whole web activity is embedded in the training courses content so that it can work offline. Distant resources might be inaccessible if the cellular network is not working

ToM JavaScript library usage

The ToM JS library is already embedded and available in the mobile web view. So you need not worry about including the library in your content.

In the webapp

Things are different in the web app. The web activity is displayed in an iframe, so before you can start calling the following methods in the webapp, you need to get a link to the library object called *ToM* in your environment. To do that, you need to reference the *ToM* object from the parent window. In order to do that, you need to put these three lines in your code:

```
window.addEventListener('load', function () {
    window.ToM = window.ToM || (window.top && window.top.ToM) || {};
});
```

Namespaces related to the web activity environment

Namespace	Purpose
ToM.data	Provides methods to get and send learning related data from and to the Mission Center
ToM.env	Provides methods to get context related values like: the learner's name, login or language, or the training ID, or activity ID.
ToM.utils	Provides various useful methods

So let's dive in.

Content display rules

In the mobile application, the content is displayed in a web view. This web view may or may not provide a toolbar with a *Close* action depending on some rules:
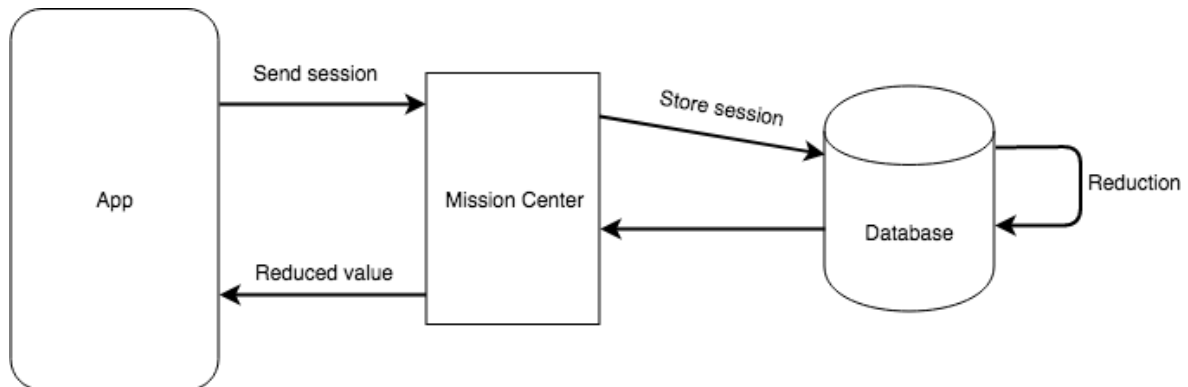
- It is a SCORM content
  - the toolbar is not displayed
  - the content will be closed when it calls SCORM's *Terminate* method
- The content uses the ToM JS Library
  - The toolbar is displayed
  - Unless the content uses the method *ToM.utils.close().* The Mission Center scans the files for this method.

ToM.data.init()

Initializes the learning related data of the learner from the application database.

About the data flow in Teach on Mars

Learning related data is sent in *sessions*. The Mission Center collects them and stores them in the Database. Then the database operates a *reduction* of the data which means it will consolidate the data according to some rules that are described below.

Data	Reduced value
time	Sum of **time** values sent in all previous sessions
progress	Maximum value of **progress** sent in all previous sessions
success	True if at least one session contains a **success: true** value, false otherwise
score	Maximum value of **score** sent in all previous sessions
points	Sum of **points** values sent in all previous sessions
suspendData	The value sent in the most recent session. This acts like a way to save any kind of data.

ToM.data.get(key)

Returns the data recorded at the given key. Just after a *ToM.data.init()*, the value is the reduced value from previous sessions, but after a *ToM.data.set()* is called, the value is the one that has been set for this session.
There is an exception with *time*. After a ToM.data.init(), the *time* value will always be 0. To get the reduced value of the time spent, use the *totalTime* key.

ToM.data.set(key, value)

Sets the session value at a given key. Make sure you set all the data you need to send to the Mission Center using this method before calling the ToM.data.send() method.

Key	Type	Validation	Description
time	integer	>= 0	The time spent on the activity by the learner (in seconds) *NB: If time is not set during a session, the application will automatically insert the actual time spent on the activity.*
progress	integer	0 .. 100	The progress the learner has made in the activity. For a course, the percentage of progress For a scored activity, just send 100 to say the learner has done it.
success	boolean	true / false	Only for a scored activity. Has the learner passed the test.
score	integer	0 .. 100	The score of the learner on this attempt (scaled from 0 to 100)
points	integer	>= 0	The number of points earned by the learner doing this activity. Points participate to the learner's ranking.
suspendData	string		Any data to save after this session. The data can be fetched again on any device on the next attempt.

ToM.data.send()

Commits the session data and asks the application to send the current session to the Mission Center. If network is not available, the application will store the session until it can be sent to the Mission Center.

A quick word about time
As mentioned before, there is a subtle trick when sending time from a web content that is read on a mobile device. Whenever the app is in background more (during a call, after the phone has gone to sleep mode, after opening another app, etc.) the activity keeps running. So if the content counts the time spent from the moment the activity was opened to the moment it is closed, it might send irrelevant data like 76 hours, while the user only spent 27 minutes on the content.

To prevent this from happening, when *ToM.data.send()* is called (or the SCORM equivalent), the application will automatically cap the time that has been recorded by the content to the actual time the content was in foreground.

ToM.env.getAll()

Returns all environment variables in a single object

Variable	Description
APPLICATION_SERVER_ID	Application server ID
APPLICATION_VERSION	Application version
COACHING_ID	Coaching ID
EMAIL	Email for current logged in user
FIRST_NAME	First name for current logged in user
LANGUAGE	Language set in the application by the user
CONTENT_LANGUAGE	Language selected in the training course (can be different from the application language)

Variable	Description
LAST_NAME	Last name for current logged in user
LEARNER_ID	The unique identifier for current logged in user
LOGIN	Login of current logged in user
SEQUENCE_ID	Activity ID
TRAINING_ID	Training ID
USER_OS	OS on witch the application is running
metadata.CODE_META	Display the value of a metadata (CODE_META is case sensitive

ToM.env.get(variable)

Returns the value of the given variable.

ToM.utils.enableWakeLock()

Enables the wake lock status on the device. This means that the device will not go to sleep mode. This might be helpful if the activity contains a media that takes time to read. A call to this function will allow the user to keep reading or playing the content without being kicked out of the activity when the phone goes to sleep mode.

ToM.utils.disableWakeLock()

Disables the wake lock status on the device. This means that the device may again go back to sleep mode after a while according to its general settings. If the phone goes to sleep mode while the activity is running, the activity will be closed and no data will be sent to the Mission Center.

ToM.utils.close()

This method closes the web activity, and makes the app return to the training course screen.

How to use sessions properly

The way your web content initiates and sends sessions will have an effect on the reporting in the Mission Center. So here's everything you need to know so that you get the reporting you need from your web content.

The **ToM.data.init()** function initiates a new session which will gather data with **ToM.data.set()** calls and will be sent to the Mission Center using **ToM.data.send()**.
Every session that is sent to the Mission Center counts as *launch* for the web activity. You can see the *launches count* in the export by activities.

Each session recorded in the Mission Center will bring new data that will be mixed (or reduced) with the previous data.

Key	Type	Reduced value (in reporting)
time	integer	All the *time* values are added together
progress	integer	The maximum *progress* value is kept
success	boolean	The reduced value is *true* if there was at least one *true* value sent.
score	integer	The maximum *score* value is kept
points	integer	All the *points* values are added together

Why is it important?

Well, depending on what kind of content you are designing, you may want to count launches and gather data differently.

Let's examine some example cases:

1. Your content is a simple course that launches straight when the content is opened

So you need to track time and progress as soon as the content is launched. And if the user leaves the content before the end, the time he spent and his progress should me kept.

To do that:
- the **ToM.data.init()** should be called at the initialization of the content (i.e. when the web page is loaded)
- the progress data should be updated on the fly when the user is browsing
- the session should be sent when the user leaves the content using a navigation button bound to **ToM.utils.close()**
- the session should be sent when the user leaves the content by killing the app or putting it in the background. To do this, you much catch the window event **beforeunload**.

2. Your content is a game that have an introduction screen and in which the user can launch consecutive game sessions.

To do that:
- it's better to keep count of every game launches, so the session initialization should be done when the user launches a game from the introduction screen
- the session should be sent at the end of a game
- if the user leaves the content or the application before the game session is over, no session should be sent (that is what is implemented in every Teach on Mars game activity)

Content translation

Web activities have a special way of handling translations which differs from other activities. The uploaded web content is common to all languages in the training course. It was way too heavy to upload pretty much the same content for each language, especially when there are a lot of media

(video, audio or images). So it has been decided to let the content creator deal with the translations inside the content.

This pattern lets you free to implement translations the way needed, whether it is a string to string translation, or a complete switch of content for each language. What you really need is to know what language the learner is reading the content in.

To do that, just call **ToM.env.get("CONTENT_LANGUAGE")** or its SCORM equivalent and you will be able to select the right language for the content when launching the activity.

SCORM implementation

The application implements both SCORM 1.2 and SCORM 2004. Not all SCORM values are implemented, but here's a list of the values that are recorded and how they are transformed to the Teach on Mars model.

Data written by the SCORM content

SCORM 1.2	SCORM 2004	ToM data
cmi.core.session_time	cmi.session_time	Time (in seconds), should appear in the export by training and by activity and on the dashboards.
cmi.core.lesson_status	cmi.success_status cmi.completion_status	This status or the combination of both statuses will affect both the progress value and the success status on the activity. if **completed** then progress = 100 if **passed** then success = true and progress = 100
cmi.core.lesson_location	cmi.location	Saved and then returned as it is when the content needs it.
cmi.suspend_data	cmi.suspend_data	Saved and then returned as it is when the content needs it.
cmi.core.score_min cmi.core.score_max cmi.core.score_raw	cmi.score_min cmi.score_max cmi.score_raw cmi.score_scaled	The score is transformed to a value between 0 and 100 and returned in the export by activity.
	cmi.progress_measure	The progress measure is stored as a value from 0 to 100 in the **progress** value.

Data read by the SCORM content

SCORM 1.2	SCORM 2004	ToM data
cmi.core.total_time	cmi.total_time	Time (in seconds), should appear in the export by training and by activity and on the dashboards.
cmi.core.lesson_mode	cmi.mode	Currently, the application will always return **normal**
cmi.core.student_id	cmi.learner_id	The ID of the logged in learner
cmi.core.student_name	cmi.learner_name	The firstname and lastname of the logged in learner
cmi.core.lesson_location	cmi.location	Saved and then returned as it is when the content needs it.
cmi.suspend_data	cmi.suspend_data	Saved and then returned as it is when the content needs it.
	cmi.progress_measure	The progress measure the max of the recorded progress measures (from 0 to 100).

SCORM 1.2	SCORM 2004	ToM data
cmi.student_preference.language	cmi.learner_preference.language	The language selected by the learner in the application. Equivalent to LANGUAGE in the ToM environment variables. Be careful, this is different than the language in which the training is displayed. In order to get this information, please use the ToM environment variable CONTENT_LANGUAGE.

Other details on SCORM contents

Content entry point

The Mission Center analyzes any *imsmanifest.xml* file and looks for the right file to launch. It looks for this kind of value:

```
<resource
    identifier="RES-0"
    adlcp:scormtype="sco"
    href="SCORM/index_scorm.html"  <-- This file will be launched
    type="webcontent">
```

Closing the content

When using the SCORM standard, the content must allow the user to close the window. A simple call to *Terminate()* or *LMSFinish()* will save the data and close the in-app browser.

A quick word about time

As mentioned before, there is a subtle trick when sending time from a web content that is read on a mobile device. Whenever the app is in background more (during a call, after the phone has gone to sleep mode, after opening another app, etc.) the activity keeps running. So if the content counts the time spent from the moment the activity was opened to the moment it is closed, it might send irrelevant data like 76 hours, while the user only spent 27 minutes on the content.

To prevent this from happening, when the data is sent (Ex: with Terminate()), the application will automatically cap the time that has been recorded by the content to the actual time the content was in foreground.