To learn more about Project Tin Can, visit http://www.adlnet.gov

---

Tin Can API (REST binding)

Revisions

Date	Updates
2011-10-25	● Corrected state API to use 'registrationID' filter instead of 'registered' verb. ● Added caution to registration definition on the use of unassigned registrations. ● clarified that the 'since' parameter excludes the specified date (up to but not including).

Definitions

Tin Can API (TCAPI): The API defined in this document, the product of "Project Tin Can". A simple, lightweight way for any permitted actor to store and retrieve extensible learning records, learner and learning experience profiles, regardless of the platform used.

Learning Activity Provider (AP): Like a SCORM package, the software object that is communicating with the LRS to record information about a learning experience.

Learning Activity (activity): Like a SCORM activity, a unit of instruction, experience, or performance that is to be tracked.

Statement: A simple statement consisting of <Actor (learner)> <verb> <object>, with <result>, in <context> to track an aspect of a learning experience. A set of several statements may be used to track complete details about a learning experience.

Learning Record Store (LRS): A system that stores learning information. Currently, most LRSs are Learning Management Systems (LMSs), however this document uses the term LRS to be clear that a full LMS is not necessary to implement the TCAPI.

Learning Management System (LMS): Provides the tracking functionality of an LRS, but provides additional administrative, and reporting functionality. In this document the term will be used when talking about existing systems that implement learning standards.

Registration: If the LRS is an LMS, it likely has a concept of registration, an instance of a learner signing up for a particular learning activity. The LMS may also close the registration at some point when it considers the learning experience complete. For Tin Can purposes, a registration may be applied more broadly; an LMS could assign a group of students to a group of activities and track all related statements in one registration. **Note:** an activity providers are cautioned against reporting registration other than when assigned by an LRS. An LRS that assigns registrations is likely to reject statements containing unassigned registration IDs.

State: Similar to SCORM suspend data, but allows storage of arbitrary key/document pairs. The LRS does not have to retain state once the learning experience is considered done (LRS has closed its "registration").

Profile: Learners and activities can both have arbitrary key/document pairs of profile data stored about them. This could be used for leaderboards, to note learner preferences, learner strengths & weaknesses, etc.

Launch

Tin Can APs do not need to be launched from an LRS, however it is still an option. When an LRS launches a Tin Can experience, it will provide the necessary information for that experience to track back to the LRS (endpoint, learner information, credentials, and optionally registration, activity ID, version, and platform). The format of the launch URL will be as follows:

<AP URL>/?endpoint=<lrs endpoint>&auth=<token>&actor=<learner>[&registration=<registration>] [&activity_id=<activity ID>&activity_version=<version>&activity_platform=<platform>]

Note that that some of the parameter values include reserved characters, and even other URLS, and therefore must be URL encoded.

Example launch link (shown without URL encoding and with line breaks for readability):

http://example.scorm.com/TCActivityProvider/
?endpoint=http://example.scorm.com/lrs/
&auth=OjFjMGY4NTYxNzUwOGI4YWY0NjFkNzU5MWUxMzE1ZGQ1
&actor={ "name" : "Project Tin Can", "mbox" : "mailto:tincan@scorm.com" }
&registration=760e3480-ba55-4991-94b0-01820dbd23a2
&activity_id=http://example.scorm.com/tincan/example/simplestatement

Partial launch information may also be provided by an LRS in the form of a launch link,

which may consist of only endpoint information, or may include learner information but not credentials. In this case, the AP would have to have been configured with or prompt for the necessary information.

If no launch information is provided, then the AP must minimally be configured with the LRS endpoint it should track to. The AP may also be configured with credentials from the LRS, in which case credentials need not be obtained for each learner.

The process of getting launch information from an LRS to an AP in a manner other than a launch link (URL) is not defined. Although it is a goal of the TCAPI to support out of browser scenarios, this is supported by allowing the AP to pass information to a LRS about learners and activities that have not been previously defined in the LRS. That is, out of browser scenarios are supported by removing the requirement for the LRS to launch the activity. Minimally, the AP must be configured with the LRS endpoint, and usually will also need authentication credentials.

Objects

Statement

The statement is the core of the TCAPI. All learning events are stored as a statement: "I did this". Bolded properties are required.

Property	Type	Default	Description
id	UUID		may be assigned by statement creator or LRS.
actor	JSON/XML object		Learner or Team object the statement is about. "I". If not specified, LRS will infer based on authentication.
verb	String		String. See table below.
inprogress	Boolean	false	Should the LRS wait for further information about this statement, is this statement just a mention of a learning experience in progress, but not yet to be submitted.
object			Activity or person object that is the object of the statement, "this"
result			Result object, further details relevant to the specified verb.
context	JSON/XML object		Context that gives the statement more meaning. Examples: Team actor is working with, altitude in a flight simulator.
timestamp			Timestamp of when what this statement describes happened.

stored			Timestamp of when this statement was recorded. Set by LRS.
authority			Actor who is asserting this statement is true. Verified by LRS based on authentication, and set by LRS if left blank.

Aside from the possible initial assignment of "ID" and "Authority" by the LRS, and the assignment of "Stored" whenever a statement is passed from system to system, statements are immutable.

Example of a simple statement:

```
{
    "id" : "fd41c918-b88b-4b20-a0a5-a4c32391aaa0",
    "actor" : {
        "name" : "Project Tin Can",
        "mbox" : "mailto:tincan@scorm.com"
    },
    "verb" : "created",
    "object" : {
        "id" : "http://example.scorm.com/tincan/example/
simplestatement",
        "definition" : {
            "name" : "simple statement",
            "description" : "A simple Tin Can API
statement. Note that the LRS does not need to have any
prior information about the actor (learner), the verb, or
the activity/object."
        }
    }
}
```

Simplest possible statement:

```
{
   "verb" : "created",
   "object" : { "id" : "http://example.scorm.com/tincan/
example/simplestatement"}
}
```

Typical simple completion with score:

```
{
    "actor" : {
        "name" : "Example Learner",
        "mbox" : "mailto:learner@example.scorm.com"
    },
```

```
    "verb" : "attempted",
    "object" : {
        "id" : "http://example.scorm.com/tincan/example/
simpleCBT",
        "definition" : {
            "name" : "simple CBT course",
            "description" : "A fictitious example CBT
course."
        }
    },
    "result" : {
          "score" : { "scaled" : .95},
          "success" : true,
          "completion" : true
    }
}
```

Statement Verbs:

There are two major competing goals in choosing a set of defined verbs for use in statements: provide sufficient verbs to clearly express any foreseeable learning event, and to ensure that different verbs are not used to express the same concept. The first goal suggests a large set of verbs, but a small set is better for the second goal.

The list of verbs below should go a long way towards meeting the first goal, while also meeting the second. There is still a need for more verbs, which can be added once an appropriate governance model is put in place.

The table below shows valid object types and results for use with defined verbs. The results listed are valid, not required results for that verb – so a statement with the verb "read" may report completion, but doesn't have to, and it should not report a score.

The attempt column indicates if a statement with the listed verb indicates the start of a new attempt. In other words, if prior statements exist for this learner and object, does the new statement represent a distinct interaction with that object (new attempt), or does it add detail to existing statements (no new attempt)? No specific behavior is triggered by attempt vs. non-attempt verbs, however clients should choose the appropriate verb type, and reporting tools may use this information to logically group results.

The list below is sorted into groups of related verbs. Related verbs have similar meanings, the same applicable object types, results, context, and attempt behavior, and can be handled as a group for reporting purposes. The specific verb used should always be displayed on reports except when aggregating statements.

Verb	Object Types	Results	Attempt?

experienced, read, watched, witnessed, studied, reviewed, learned, attended, heard	Content (video, book, article, blog), Event	Completion	Yes
attempted, performed, played, simulated	Any	Completion, Success, Score, (interaction details)	Yes
completed, passed,mastered, failed	Any	Completion, Success, Score, (interaction details)	Yes
answered	question	Completion, Success, Score, interaction details	Yes
interacted, drove, piloted, used	control, thing, interaction	(interaction details)	No
achieved	Any activity	Completion, Success, Score, (interaction details)	No
participated	Event		No
mentored (by)	Person	Completion	
commented	Any	Comment	No
asked	Any	Question	No
created, authored, wrote, edited, blogged	Content (video, book, article, blog)	Completion, Success, Score	Yes
shared, posted	Content (video, book, article, blog)		No
taught	course, class, session, lecture, topic	Completion, Success, Score	Yes
imported	Any object		No

Completed, passed, mastered, and failed are special in that they indicate a specific result value. All statements using these verbs shall be read by an LRS as having completion = true, and passed, mastered, and failed will have success set to true,true, or false respectively. They are provided as a convenience to allow these common concepts to be compactly and clearly expressed. It is OK to explicitly set these values as described above in a statement with one of these verbs, or leave the results blank. A statement using the completed, passed, or failed verb and containing contradictory results is invalid.

When queried, an LRS must expand statements using completed, passed, or failed to include the appropriate results block as implied by the verb used.

The achieved verb is intended for scenarios where results are stored or adjusted after the tracking of the initial attempt, such as when a student responds to an essay question (attempted), and then later an instructor grades it (achieved). For example: Instructor asserts that I achieved assessment with result “passed”. Graded was not chosen because in the “I did this” model, using a graded verb we would be talking about the instructor, not the student. If one does want to report about the instructor grading something, “graded” may be used, as can any unlisted verb, but always keep in mind

that the actor is the subject of the statement, including any associated information such as score. So if I make the statement: "I graded Ben's test with result: passed, 80%", that score information means I successfully graded the test, and someone assigned me a score of 80% in doing so. It should not be taken to mean the score I assigned while grading. That could be expressed: "Ben achieved test with result: passed, 80%" The important point there is "I" has to be "Ben" in order to assign Ben a score.

Statements with the registered verb will usually be added by the LRS or an agent acting on behalf of the LRS, and cause all prior statements on that activity (or it's children)/ actor combination to be considered obsolete. Obsolete statements will not be returned by default when reading statements from the API, and should not be displayed on reports unless specifically requested.

The imported verb exists as a way to get activity or actor definitions into an LRS, without requiring a separate import API or misuse of another verb. Since activities or actors can be used in statements without previously importing them, and an LRS is required to save the information provided in such statements, it is possible to import an activity definition just by using it in a statement.

Result

If the result of a statement is logically a simple string, eg: I commented *"This question is a little vague"* on "question1", then that string may be used as the result object. Otherwise, the result object is as follows:

Property	**Description**
score	Score object (or not specified)
success	true, false, or not specified
completion	Completed, or not specified
response	As in cmi.interactions.n.learner_response. Only valid if activity is of type "Interaction"
<extension>	Other properties as needed.

Context

Context information relevant to the current statement.

Property	**Description**
registration	UUID of the registration statement is associated with.
Instructor	Instructor that the statement relates to, if not included as the actor or object of the statement.
Team	Team that this statement relates to, if not included as the actor or object of the statement.

Activity	A learning activity this statement is related to. For example, if I am studying a textbook, for a test, the textbook is the activity the statement is about, but the test is the context activity. This activity could also be a session, like a section of a specific course, or a particular run through a scenario. So the statement could be about "Algebra I", but in the context of "Section 1 of Algebra I". This is particularly useful with the object of the statement is an actor, not an activity. "I mentored Ben with context Algebra I".
statement	Another statement (either existing or new), which should be considered as context for this statement. This could be used to add context to a comment, or when grading.
<extension>	Any other domain-specific context relevant to this statement. For example, in a flight simulator altitude, airspeed, wind, attitude, GPS coordinates might all be relevant.

Score

Property	**Description**
Scaled	cmi.score.scaled (recommended)
Raw	cmi.score.raw
min	cmi.score.min
max	cmi.score.max

State

Property	**Description**
id	String, set by AP, unique within state scope (learner, activity)
updated	Timestamp
contents	Free form.

Note that in the REST binding, State is a document not an object. ID is stored in the URL, updated is HTTP header information, and contents is the HTTP document itself.

Agent (learner or team)

These will be FOAF agent objects http://xmlns.com/foaf/spec/#term_Agent

A key design goal for the TCAPI is to enable an LRS to receive learning records about a learner that has not yet been defined in the LRS, or from a LP that does not have access

to the identifier used by the LRS for that learner. FOAF, or "Friend of a Friend" agent objects provide some capabilities that will help in achieving those goals.

1. FOAF provides a vocabulary with a variety of options for uniquely defining an agent (or person), such as email address, weblog address, or account on a system (the LRS for example).
2. OWL (Web Ontology Language), which FOAF uses, provides a way to declare what properties can be used to uniquely identify an entity, they have the "Inverse Functional Property". Using this information, it is possible to merge two different sets of statements about the same entity, provided they have a match in one of these properties. For example, email is an inverse functional property for Person, if we have two sets of statements (FOAF objects) about someone who has the email address person1@example.com then we know that those statements are about the same person.

When defining a Learner, Team, or Agent, at least one field that has the Inverse Functional Property (http://www.w3.org/wiki/InverseFunctionalProperty) must be defined. Note that the "account" field in FOAF is not defined as having the Inverse Functional Property, but in the context of the TCAPI it will be considered to have this property.

The LRS should consider FOAF agent with matching fields having the inverse functional property (such as email) to be the same agent. This equivalence should be applied both when filtering statements based on agent, and when reporting. However, the LRS should still be able to report on the original FOAF agent that was associated with any statement, before applying any merge operation.

Activity

Property	**Description**
id	URI , may be a URL. If a URL, the URL should refer to the AP for this activity, or a description or metadata for this activity. It should not refer to something unrelated to the activity. This URI is unique, any reference to it always refers to the same activity, the AP must ensure this is true and the LRS may not attempt to treat multiple reference to the same URI as references to different activities.
revision	String: Different minor revisions of an activity that is logically still the same can share an ID and be differentiated by revision
platform	String: activities meant to run on different platforms that are logically still the same can share an ID and be differentiated by platform
definition	Metadata, See below

ActivityDefinition

name	String, what the activity is called
description	String, a description of the activity (question text if a question)
type	Course, Module, Meeting, Media, Performance, Simulation, Assessment, Interaction, Objective
children	If this activity permanently has sub-activities (questions within an assessment for example), they may be listed here.
interaction_type	As in CMI interactions, only valid for "Question"
correct_responses	As in CMI interactions, only valid for "Question"
extensions	Other properties as needed

Activity/Learnerprofile

Property	**Description**
id	String, set by AP, unique within activity/learner scope (learner, activity)
updated	Timestamp
contents	Free form.

Note that in the REST binding, activity and learner profiles are documents not objects. ID is stored in the URL, updated is HTTP header information, and contents is the HTTP document itself.

RuntimeCommunication

The TCAPI consists of 4 sub-APIs: statement, state, learner, and activity profile. The statement API can be used by itself to track learning records.

Security

API calls may be authenticated using HTTP Basic authentication, using one of the following credentials:

- Username/password of a LRS user.
- Token in Username field, no password

- Token received in launch information
- Token received via OAuth negotiation with LRS

An LRS is free to accept unauthenticated API calls, but rejecting them is recommended.

Protection from tampering and eavesdropping can be achieved using transport layer security.

Statements

Store (Statement)

POSThttp://example.com/TCAPI/Statements/

Stores a statement, or a set of statements. Returns: 200 OK, statement ID(s) (UUID). Since the PUT method targets a specific statement ID, POST must be used rather than PUT to save multiple statements, or to save one statement without first generating a statement ID. An alternative for systems that generate a large amount of statements is to provide the LRS side of the API on the AP, and have the LRS query that API for the list of updated (or new) statements periodically. This will likely only be a realistic option for systems that provide a lot of data to the LRS.

GEThttp://example.com/TCAPI/Statements/<StatementID>

Returns: 200 OK, statement

PUThttp://example.com/TCAPI/Statements/<StatementID>

Returns: 204 No Content
Errors: 409 Conflict

Stores statement with the given ID. This MUST NOT modify an existing statement. If the statement ID already exists, the receiving system SHOULD verify the received statement matches the existing one and return 409 Conflict if they do not match.

GEThttp://example.com/TCAPI/Statements/

Returns: List of statements in reverse chronological order based on "stored" time, subject to permissions and maximum list length

Parameters:

Parameter	Type	Default	Description
verb	String		Filter, only return statements matching the specified verb.

object	Actor / Activity Object (JSON/ XML)		Filter, only return statements matching the specified object (activity or actor). Object is an activity: All populated fields in the filter activity, out of the list: {ID, platform, revision} must match corresponding fields in statements to be returned. Object is an actor: same behavior as “actor” filter, except match against object property of statements.
registration	UUID		Filter, only return statements matching the specified registration ID. Note that although frequently a unique registration ID will be used for one actor assigned to one activity, this should not be assumed. If only statements for a certain actor or activity should be returned, those parameters should also be specified.
descendants	Boolean	True	When filtering on activities, consider a statement a match if the object or context:activity is the specified activity, or one of its descendants.
actor	Actor Object (JSON/XML)		Filter, only return statements about the specified agent. Note: at minimum agent objects where every property is identical are considered identical. Additionally, if the LRS can determine that two actor objects refer to the same agent, they should be treated as identical for filtering purposes. See agent object definition for details.
since	Timestamp		only statements stored since the specified timestamp (exclusive) are returned
until	Timestamp		only statements stored at or before the specified timestamp are returned
limit	Nonnegative Integer	0	Maximum number of statements to return. 0 indicates return the maximum the server will allow.
offset	Nonnegative Integer	0	Skip this many statements that would otherwise be in the list before starting to return statements.
authoritative	Boolean	True	Only include statements that are asserted by actors authorized to make this assertion (according to the LRS), and are not superseded by later statements.

sparse	Boolean	True	If true, only include minimum information necessary in actor and activity objects to identify them, If false, return full activity and actor objects.
Instructor	Actor Object (JSON/XML)	True	Same behavior as “actor” filter, except match against “context:instructor”.

Note: due to query string limits, this method may be called using POST and form fields if necessary. The LRS will differentiate a POST to add a statement or to list statements based on the parameters passed.

State

PUT | GET | DELETE http://example.com/TCAPI/activities/<activity ID>/state/<actor>/<State ID>[?registration=<registration>]

Stores, fetches, or deletes the specified state document in the context of the specified activity, actor, and registration (if specified). Actor may be an individual or a team.

DELETE http://example.com/TCAPI/activities/<activity ID>/state/<actor>[?registration=<registration>]

Deletes all state data for this context (activity + actor [+ registration if specified]).

GET http://example.com/TCAPI/activities/<activity ID>/state/<actor>[?since=<timestamp>][&registration=<registration>]

Fetches IDs of all state data for this context (activity + actor [+ registration if specified]). If “since” parameter is specified, this is limited to entries that have been stored or updated since the specified timestamp (exclusive).

Generally, this is a scratch area for activity providers that do not have their own internal storage, or need to persist state across devices.

ActivityProfile

PUT | GET | DELETE http://example.com/TCAPI/activities/<activity ID>/profile/<profile object key>

Saves/retrieves/deletes the specified profile document in the context of the specified activity

GET http://example.com/TCAPI/activities/<activity ID>/profile[?

since=<timestamp>]

Loads IDs of all profile entries for an activity. If "since" parameter is specified, this is limited to entries that have been stored or updated since the specified timestamp (exclusive).

GET http://example.com/TCAPI/activities/<activity ID>

Loads complete activity objects that have the specified ID. This includes populated child activity objects, but only ID information for the parent activity (if applicable).

Parameter	Type	Default	Description
Revision	String		If specified, only include activities of the specified revision
Platform	String		If specified, only include activities for the specified platform

ActorProfile

PUT | GET | DELETE http://example.com/TCAPI/actors/<actor>/profile/<profile object key>

Saves/retrieves/deletes the specified profile document in the context of the specified learner (learner may be an individual or a team)

GET http://example.com/TCAPI/actors/<actor>/profile[?since=<timestamp>]

Loads IDs of all profile entries for an actor. If "since" parameter is specified, this is limited to entries that have been stored or updated since the specified timestamp (exclusive).

GET http://example.com/TCAPI/actors/<actor>

Loads full actor object for the specified actor. Even though an actor object is specified in the get call, the LRS may have more information about that actor that can be returned. For example, the actor object passed in could include only an email address, but the LRS could return an actor object populated with name, department, and role.

Problems

Registration: LMS concept, but may need to be included in launch information, tracking.

Solution: Registration ID (UUID) becomes part of the statement stream, may be specified by clients when storing statements. If the LRS provides a launch link, it

would provide the registration to track against in that link, if the launch is based on a registration. If the LRS provides a registration ID, then the AP must use it when reporting statements, and should use it when querying statements unless specifically intending to retrieve previous registration information as well. This method allows for multiple simultaneous registrations.

Rejected Solution: "Registered for" verb causes all prior statements to be "non-authoritative", starts new registration. **Only one registration at a time is valid**. Clients don't really have to know about registrations. This does not work because different assignments may be made for both an activity and a child of that activity. In that case, using the registered verb on the child activity would reset progress from both the assignment of that particular activity, and its parent. Also, only one registration at a time being valid is an unreasonable restriction due to this same sort of overlap, a registration for an activity should not preclude a registration on another activity that has it as a child.

The concepts of "attempt" and "submitted" are similar. Furthermore, specifying some verbs as signifying a new attempt, and others not limits verb choice and therefore statement expressiveness. We should consider merging these concepts somehow. The problem with doing that is that "submitted" would be associated with ending an attempt, whereas the attempt vs. non-attempt verbs determine whether or not to start a new attempt. Both seem to be needed, submitted when it is known whether or not more details will be sent, and attempt vs non-attempt to determine whether a new statement is a revision of an existing attempt (an instructor grading the attempt for example), or the start of a new attempt.

Result: Determining "authoritative" results will be difficult unless each result field is constrained to only be enabled on one verb. If two statements have different verbs, and different results, particularly different partial results, how does that get summarized? Could potentially be solved with a "scored" verb, "passed" verb, etc.

Statement visibility (privacy) is a concern. There is nothing in the API to prevent any actor from viewing any statements written by any other actor, though an LRS may chose to limit this. To avoid an interoperability mess, at minimum best practices on what actor types (admin, instructor, etc) can view which statements should be established.

FOAF account is not defined as having the inverse functional property, however we need a way to uniquely identify agents (people) based on their LRS account as an option. Consider adding an extension property rather than changing the definition of FOAF account.

Should the FOAF agent object have a type added so the LRS can differentiate between a person and an agent? Does it matter?

Results/Score section should be updated to use CMI5

Possibilities

UUID of statement could be a hash of other statement fields (except store time). This could potentially allow two systems to generate the same statement, with the same ID, if describing the same event (at the same time).

Statements could be signed by the “Authority”. This would require canonicalization of statements first.

Appendix A: Bookmarklet

TCAPI enables using an “I learned this” bookmarklet to self-report learning. The following is an example of such a bookmarklet, and the statement that this bookmarklet would send if used on the page: http://scorm.com/tincan.

The bookmarklet would be provided by the LRS to track to, for a specific user. Therefore the LRS URL, authentication, and actor information is hard coded in the bookmarklet. Note that since the authorization token must be included in the bookmarklet, the LRS should provide a token with limited privileges, ideally only enabling the storage of self-reported learning statements.

The UUID generation is only necessary since the PUT method is being used, if a statement is POSTED without an ID the LRS will generate it.

In order to allow cross-domain reporting of statements, a browser that supports the “Access-Control-Allow-Origin” and “Access-Control-Allow-Methods” headers must be used, such as IE 8+, FF 3.5+, Safari 4+, Safari iOS Chrome, or Android browser. Additionally the server must set the required headers.

```
var url = "http://localhost:8080/TCAPI/
Statements/"+_ruuid();
var auth = "Basic dGVzdDpwYXNzd29yZA=="; var statement =
{actor:{"mbox" :
"mailto:learner@example.scorm.com"},verb:"",object:{id:"",d
efinition:{}}};
var definition = statement.object.definition;

statement.verb='experienced';
statement.object.id = window.location.toString();
definition.title = document.title;
definition.type="Media";
definition.my_extension_activity_type="bookmarklet link";

var xhr = new XMLHttpRequest();
xhr.open("PUT", url, true);
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Authorization", auth);
xhr.onreadystatechange = function() {
     if(xhr.readyState == 4 ) {
                alert(xhr.status + " : " + xhr.responseText);
     }
};
xhr.send(JSON.stringify(statement));

/*!
```

```
Modified from: Math.uuid.js (v1.4)
http://www.broofa.com
mailto:robert@broofa.com

Copyright (c) 2010 Robert Kieffer
Dual licensed under the MIT and GPL licenses.
*/
function _ruuid() {
     return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/
[xy]/g, function(c) {
          var r = Math.random()*16|0, v = c == 'x' ? r :
(r&0x3|0x8);
          return v.toString(16);
     });
 }
```

Example statement using bookmarklet

Headers:

```
{ 'content-type': 'application/json; charset=UTF-8',
  authorization: 'd515309a-044d-4af3-9559-c041e78eb446',
  referer: 'http://scorm.com/tincan/',
  'content-length': '265',
  origin: 'http://scorm.com' }
```

Method/Path:

```
PUT : /TCAPI/Statements/ed1d064a-eba6-45ea-a3f6-
34cdf6e1dfd9
```

Body:

```
{
     "actor":{
          "mbox":mailto:learner@example.scorm.com
     },
     "verb":"experienced",
     "object":{
          "id":"http://scorm.com/tincan/ ",
          "definition":{
               "name":"SCORM » Project Tin Can: SCORM
          Communication Modernization",
               "type":"Media",
               "my_extension_activity_type":"bookmarklet
          link"
          }
     }
```

```
}
```