SCORM and xAPI

One of the hottest things when I was working in higher education was SCORM. It provided a way for courses coded and packaged to the SCORM specification to send data to a Learning Management System for it to be tracked and recorded.

Jump to 2010 when Rustici Software began the first research into *SCORM 2.0* with the specification reaching a 1.0 release in 2013 and becoming first the TinCanAPI and, eventually, the Experience API (xAPI).

xAPI is based on Activity Streams (and its corresponding draft RFC) and provides an education-focused extension to the activity streams specification so, in theory, we should be able to combine streams from xAPI-enabled courses and other sources.

A technical definition:

> The Experience API is a set of RESTful web services. The services allow communications between a *Learning Activity Provider* and a *Learning Record Store (LRS)* using one or more “Statements”. Each “Statement” describes a learning experience of an “Actor” performing a “Verb” (representing an action) on an “Object”. For example: **Peter finished an exam**, **Helen read a document**, or **David started a project**.

When you think of an *Activity Provider*, it’s easy to compare it to traditional e-learning content but that’s not completely accurate. xAPI can be used not only track learning events, but also other type of performance and learning activities like books, videos, web pages, etc. The variety of learning events gives different departments (HR, L&D, direct managers) the possibility to improve performance by targetting learning for individual users based on their learning record.

Why is this important

xAPI gives us bigger and better insights into the learning process, and flexibility defining learning; We’re no longer limited to SCORM packages and tied to the LMS as our source of truth.

In technical terms, we can pass an xAPI compliant JSON object to an LRS from

wherever we can run a script on a page or from a custom data entry point. We are no longer tied to Learning Management Systems to deliver learning although we can use LMSs to deliver xAPI content using cmi5 (discussed later in the post).

In the same way we are not tied to SCORM to communicate success and failure of our learning modules. With xAPI we get a much richer and customizable vocabulary to express learning activities and outcomes.

cmi5

When talking about SCORM and xAPI you may ocassionally hear the term ***CMI 5*** thrown around. It started as a parallel attempt by the AICC (Aviation Industry Computer-Based Training Committee) to create a successor to SCORM and AICC standards in 2010.

In 2012 the AICC and ADL began working on a new specification for LMS-to-Assignable Unit (AU) communication. The new specification uses xAPI as the communication and data layer, combines the features of AICC and SCORM, and takes advantage new features on xAPI.

cmi5, conceptually, is the LMS use case for xAPI; it defines how the LMS and the content will communicate using the LRS.

ADL developed cmi5 with the following goals:

Interoperability
: A cmi5 assignable unit should work the same across all LMS systems that support the specification, just like a SCORM package. cmi5 has a similar import specification, but with cmi5 only the course structure is imported, not the actual content. This means the content can reside anywhere—behind a firewall, as an app on a mobile device, etc.

Extensibility
: Unlike SCORM, the data cmi5 tracks is not limited. Since cmi5 is based on xAPI it supports extensions. You can also track binary data like videos, pictures, and audio clips. You can even share data across multiple assignable units.

Mobile Support
: Here again, cmi5 benefits from xAPI: since the base communication mechanism handles mobile devices, so does cmi5.

See Experience API, cmi5, and Future SCORM for additional information about cmi5, verbs defined for interoperability, features of compliant systems and use cases that show how these features and advantages work. For a more detailed comparison for different groups, see: SCORM vs cmi5 Comparison.

Interoperability: A possible use case

The thing that intrigued me the most about xAPI is the concept of interoperability.

Imagine, for example, Jenny, a recent high school graduate who enters a community college where, thanks to the system-wide Learning Record Store (LRS), her academic record is tracked along with co-curricular and extracurricular activities that she voluntarily chose to share.

She takes courses from multiple community colleges in the state and, because they all share the same LRS, the courses are recorded seamlessly on the statewide LRS, making it easier to transfer courses between different institutions.

After graduation Jenny is accepted to a 4-year university. She requests her learning record to be transferred to the University's LRS so the university can do credit transfer and prerequisite evaluation.

She stays at the university to complete her bachelor's and Master's degree, upon which she takes a job at a large company. There she transfers her learning record the company's LRS so HR can see her activities all the way back to community college and assist with training and career progression.

This sounds far fetched, particularly when today we have to go through the registrars and HR departments to get a small amount of what learners have done in their educational careers. With one or more organizations using LRS systems it should be possible to leverage the learning histories of our users to provide them with more challenging and rewarding training and help with career progression.

Each school or business uses common data as defined in one or more registries so the meaning of each activity is understood by all groups that received the data.

There is also a way for the users to request data transfer and an implicit understanding that the users own their data. It is a balancing game between data ownership and privacy

See Moving and Receiving xAPI Data and Semantic interoperability for more ideas of how we can do this.

Integrating xAPI to external content

One of the adavantages of xAPI over SCORM, in my opinion, is that xAPI is not tied to content in an LMS. Because of this, xAPI offers a much wider definition of what construes learning and it can be used in a variety of fields before grading.

The only “requirement” to work with xAPI is to have a Learning Record Store (LRS) where to store the data generated by the xAPI statements. During development I’ve chosen to use the free tier of Rustici’s SCORM Cloud

I’ve also chosen to use Rustici’s TinCanJS as my xAPI library. TinCan was the codename for xAPI while under development as Rustici.

Structuring the xAPI JSON object

xAPI uses JSON as the payload language. JSON is an international standard (ECMA-404) for a data interchange format.

In the simplest format the JSON we need to build an xAPI statement looks like this:

```
{
  "actor": {
    "name": "Sally Glider",
    "Sally Glider""mbox": "mailto:sally@example.com"
  },
  "verb""http://adlnet.gov/expapi/verbs/experienced",
    "display": {
 "http://adlnet.gov/expapi/verbs/experienced""display"    "id": "htt": ""
    }
  },
  "object"   "definition": {
      "name": {
        "en-US": "Solo Hang Glidin": ""definition""definition": {
      "name": {
        "en-US": "Solo Hang Gliding"
      }
    }
  }
}
```

We'll discuss each part of the statements in detail below:

Actor

The person who executes the action on an object. We can use more than one identifier for the same actor.

```
"actor": {
  "name": "Sally Glider",
  "Sally Glider""mbox": "mailto:sally@example.com",
  "account"age": "h": ""homePage"com",
    "name": "sallyglid": ""name": "sallyglider434"
  }
},
```

In the fragment above we've refined Sally's information by adding a Twitter account with the name `sallyglider434`. We could also use an internal company URL for the homepage and an LDAP credential as the name.

Verb

```
"verb": {
  "id": "http://adlnet.g"http://adlnet.gov/expapi/verbs/experienced""displ
  }
},
```

The verb consists of an ID pointing to a URI and a display that has one or more children indicating the language used and the value for that verb in the specified language.

Think of it as a JSON version of an XML namespaces.

Because we have to agree on what the verbs mean there the people who created xAPI also created a registry that define the different verbs in a way that we can agree to. We'll review why this is important when we talk about interoperability.

For verbs that we create, we can create and maintain our own registry to present the verbs we use so internal and external people can reference them when reviewing a learner's record.

Object

The object is what action the actor (person) did (verb). You can define multiple

objects that, as verbs, resolve to a unique URL.

```
"object": {
  "id": "http://example.co"http://example.com/activities/solo-hang-gliding
    "name": { "en-US": "Solo Hang Gliding" }
  }
}
```

The activities we define don't have to be tied to activities explictly related to education and training. They can also represent professional development, additional certifications, courses or workshops that a student took outside school, events and conferences they attended, presented at or helped organize.

A more complete version of the xAPI statement is shown below. It defines both an object (what item the `actor` does `verb` to) and a target (the recipient of the action on the object). This example, taken from activity streams base schema shows this second type of statement.

```
{
  "actor": {
    "objectType": "person",
    "person""displayName"
  },
  "verb": "sell"": ""object": {
"object""object": "product",
    "di"product"": "A cool produc": ""displayName"": {
    "objectType": "pe": ""objectType""target": "Laura"
  }
}
"Laura": "person",
    "displayName": "Laura"
  }
}
```

Making it work in Javascript

Now that we have an idea of what we want to do, let's look at how to do it.

Because most of my content is web based it makes the most sense to code this as a set of Javascript functions that run when the user completes certain actions on the page.

Before we can submit a statement we have to connect an authenticate to the LRS we are working with.

We use TinCanJS LRS class to define the LRS location (endpoint), credentials (user name and password), and any additonal information (in this case that we don't want the LRS to allow failed logins).

If the LRS login is not successful, for whatever reason, we log the information to the console. We should also warn the user that the login did not succeed so they won't try to complete work that won't be recorded.

```
try {
  const lrs = new TinCan.LRS({
    endpoint: "https://cloud.scorm.com/tc/public/",
    username: "<Test User>",
    password: "<Test Password>",
    allowFail: false
  });
}
catch (err"<Test User>".log("Failed to setup LRS object: ", err);
  "Failed to setup LRS object: "// What else do we want to do with the er
}
```

Once we have a successful login, we can jump to creating a statement. This example uses the Statement class from TinCanJS to build a statement that says: "John Smith experienced TinCanJS".

```
const statement = new TinCan.Statement({
  actor: {
    name: "John Smith",
    mbox: "mailto:john@example.org"
  },
  verb: {
    id: "http://adlnet.gov/expapi/verbs/experienced"
```

```
  },
  target: {
    id: "http://rusticisoftware.github.com/TinCanJS"
  }
});
```

The final step is to save the statement to the LRS. I'm somewhat dissapointed that TinCanJS still abstracts XHR instead of Fetch, but I think it's a good solution to provide support for older browsers.

We test to see if there is an error (err !== null) then, if there is no XHR we log the error to console and bail.

If XHR failed (xhr === null) we also bail and log the error to console. In either of these cases there's nothing for the code to do.

If there is no error and XHR returned successfully then the statement was saved and we report that to console.

We should also notify the user whether the statement was saved to the LRS or not.

```
lrs.saveStatement(statement, {
  callback: function (err, xhr) {
    if (err !== null) {
      if (xhr !== null) {
        console.log("Failed to save statement: " + xhr.responseText + " ("
        " ("// Notify the user that statement wasn't saved     return;
      }

      console.log("Failed to save statement: " + err);
      // N"Failed to save statement: "// Notify the user that statement wa
    // Notify the user that the data"Statement saved"// Notify the user th
  }
});
```

There's a lot more to do with the code that we've covered so far.

The code so far doesn't include user authentication. We need to authenticate users so we can get the actor information for the statement.

We should also notify the user if we're unable to save the statement to the LRS, but that's dependent on the type of application we're creating.

Links and Resources (and more questions)

- SCORM
  - SCORM explained
  - Cooking up a SCORM — Click2Learn (content hosted in scorm.com, Click2Learn site is no longer available)
- xAPI
  - What is the Experience API?
  - cmi5
  - Interoperability
    - Learning Design Transformed
    - Kirkpatrick
    - 70:20:10
    - Blended Learning
    - Action Mapping
    - ADDIE
  - xAPI Tools Update: Spring 2019
- Tools
  - xAPI Client Libraries — Rustici
  - TinCanStatementViewer — Rustici
- Activities, Verbs and Objects
  - xAPI Registry — Rustici
  - Activity Streams
  - xAPI Statements 101
  - xAPI Lab