More examples

This is a more complex example of a xAPI statement. It addresses some of the questions

```
{
  "actor": {
    "objectType": "person",
    "person""displayName": "Carlos Araya"
  },
  "verb""id": "h": ""id": "https://w3id.org/xapi/dod-isd/verbs/presented"
    "display"n-US": "pr": ""en-US"     "es-CL": "presen": ""es-CL"    "nar
  ": ""name"ct": {
    "id": "h": ""object": {
    "id"t/activities/conference-presentation",
    "objectType": "Presentation"": ""objectType": "Presentation",
    "definition" "Conference presentation",
   "Conference presentation" en conferencia"
    },
    "displayName":"displayName"US": "The what and the why of xAPI",
      ": ""displayName": {
      "en-US": "The what and the why of xAPI",
      "es-CL": {
      "name": "The Conference",
      "id": ""context": {
    "Conference": {
      "name": "The Conference",
      "id": "https://theconference.com/"
    },
  }
}
```

Converting content by hand

There are tools that will convert SCORM to cmi5. Tools like TinCanJS, and Rustici (SCORM) Driver require you to open the courses and change the SCORM scripts and API calls to xAPI (when using TinCanJS) and platform neutral calls (when using Rustici Driver)

The following examples use TinCanJS to build the different parts of an xAPI interaction with an LRS.

The first block attempts to connect to the specified LRS using the credentials given in the user name and password fields.

Unlike many things on the web today. This code must be run synchronously, nothing else in the page can run until we know if we were able to log in to the LRS.

In the catch block we could also pop an alert or some other way to notify the user that the connection was unsuccessful.

```
let lrs;

try {
  lrs = new TinCan.LRS({
    endpoint: "https://cloud.scorm.com/tc/public/",
    username: "<Test User>",
    password: "<Test Password>",
    allowFail: false
  });
}
catch (ex) {
 "<Test User>""Failed to setup LRS object: ", ex);
  "Failed to setup LRS object: "// TODO: do something with error
  // can't communicate with LRS
}
```

The next section is where we build the statement that we want to send to the LRS. See Structuring the xAPI JSON object for more details of the different parts of the statement we're building.

```
var statement = new TinCan.Statement(
  {
    actor: {
        mbox: "mailto:info@tincanapi.com"
    },
    verb: {
```

```
        id: "http://adlnet.gov/expapi/verbs/experienced"
    },
    target: {
        id: "http://rusticisoftware.github.com/TinCanJS"
    }
  }
);
```

We use XHR to save the statement to the LRS. While most modern browsers support fetch, we're bound to support older systems where fetch is not supported. Rather than using a polyfill, the creators of TinCan.js chose to support the older protocol as it'll allow older browsers to talk to the LRS.

```
lrs.saveStatement(statement, {
    callback: function (err, xhr) {
      if (err !== null) {
        if (xhr !== null) {
          console.log("Failed to save statement: " + xhr.responseText + "
          " ("// TODO: do something with error, didn't save statement
        }

        console.log("Failed to save statement: " + err);
        // T"Failed to save statement: "// TODO: do something with error,
      // TODO: do something with succe"Statement saved"// TODO: do someth
    }
  }
);
```

There are other portions of the TinCan.js library that we haven't used because they are not used to send data to the LRS.

xAPI Quarterly published an article about how to customize Adobe Captivate to generate xAPI statements that shows how we could make other authoring tools generate xAPI statements even if the tool doesn't support the API natively.

Another way to get statements on our content and products is to roll up our slves and code the statement and communication with the LRS ourselves. This would make our content less dependent on third party authoring tools and it

would give us more flexibility on how we create and deploy our content.

We'll discuss more about one way to add xAPI statements to our content later.

What content do we want to track? From what sources?

Do we need an LMS?

Is the LMS budled with an LRS? Do we need it to be?

Are we hosting on premise? private cloud? vendor?

How will learners authenticate?