xAPI Deep Dive Activity

Anecdotally, the most common question asked when starting down the xAPI development road is: “How do I get/create an activity ID?”

It seems the simplest place to start with xAPI is sending a simple statement, and when showing xAPI to someone new we often start with the Actor-Verb-Object structure and give them the “I did something” example. Satisfying the first two parts of that structure is fairly straightforward, given an example statement or two. Most people can easily identify themselves with an email address which gets them to an Agent for use as the ‘actor’ pretty quickly, and with a list of common verbs already widely known, it is easy enough to copy and paste one of those.

Since xAPI is intended to be used for tracking experiences, the natural progression then is to include an Activity in the third part of the structure — the “something.” Now we run into a problem, or a bunch of them — nothing to copy and paste, nothing given to us by someone else to use in the statement. We are going to have to actually create something!

So what is an activity? Possibly the only thing we can say about an activity is that it has boundaries. We need a fundamental way to say that an activity is contained to a specific amount of time and/or potentially at a specific location. Those boundaries therefore are physical (or virtually physical, if that can possibly make sense) and/or temporal. Assuming we can identify a set of initial boundaries we may then have the ability to subdivide the area or time encompassed by those boundaries to create smaller and smaller sets of activities. The granularity with which we subdivide the activity space matters very little for the simple act of recording statements, but is extremely important to be able to later derive meaning from a set of statements for reporting or other types of use.

For instance, if we want to be able to track attendance at a conference, it may be sufficient to create an activity for the conference as a whole. But, if we want to know the most popular speaker at a conference, we will need to have at least session granularity to our data. If we want to determine the most popular page on a website, we will need to have a page-specific activity. But, if we only care about total visitors to a site, we only need an activity for the site itself.

These are pretty big activities, but an activity can be as small or short as a particular instant in a video being seen, or something less physically concrete such as a single question in a quiz.

Beyond defining the boundaries for an activity, it is important to define relationships between activities. In the above examples, a “larger” activity was subdivided into smaller activities, which forms a parent/child relationship. Other relationships can be indicated via the Experience API, and while it is possible to create statements using an activity in isolation, it is important to think through how activities can be grouped to realize better reporting and decision making later in the process. Just as relationships between people change over time, relationships between activities are not necessarily fixed. For instance, a session at a conference could be defined in such a way that the session is given at multiple conferences, so it may be that the session itself relates to multiple conference activities rather than being specific to a conference. This increases the reusability of the activity and can lead to more interesting social reporting possibilities. Alternatively, there could be a specific conference session activity **and** a generic session activity.

Along with the explicit relationships that are defined amongst activities, there is an implicit relationship within a xAPI statement between the activities and the Verb. We covered that territory in “Statements 101” so I won’t repeat it here.

An Activity (note the capital “A”) then, as it pertains to the Experience API, is a type of object. The xAPI specification lays out precisely the structure of that object and makes some recommendations on what should be included when creating statements. The

structure of an Activity object is quite basic — it includes only three properties. The only requirement is an ‘id’ property that has a value that is a URI (got me again, it is really an IRI). The other two optional properties are the ‘objectType’ which has a value of “Activity” when included, and the ‘definition’ which itself is an object and is where the complexity of an Activity structure lives.

Identifying

In so far as an Activity identifier is just a URI, constructing one is trivial. In a web-enabled world, URIs are all around us. Unlike with Verbs, as described in “Deep Dive: Verbs,” activities will be coined liberally and are unique to an Activity, so should only be re-used when specifically talking about the exact same activity, and therefore will generally be coined by an Activity Provider. When selecting identifiers for Activities, the creator should either own or have permission to use a particular domain name space to prevent collisions. Care should also be taken so that the Activity described by a specific identifier is not changed to reference or be reused for what could be considered a different activity, after all it is a unique identifier. While the specification only states that the identifier be a URI, it is considered a best practice to use a scheme that can ultimately be resolvable by a large number of applications, such as “http” and “https”, and to use a fully qualified domain name rather than some shortened representation as are often seen on Intranets. These best practices specifically target the interoperability of systems that the Experience API was designed to provide. Following these best practices will also mean that the URI may eventually become a URL with the ability to be resolved to meta data associated with that Activity. Just as with Verbs, there is no requirement to make URIs resolvable, but forward-looking systems will do so and minimally need to allow for it to be done in the future.

Some sample identifiers:

- ../TinCanJS/Test/TinCan_getStatement/sync
- ../JsTetris_TCAPI
- ../GolfExample_TCAPI
- ../GolfExample_TCAPI/GolfAssessment.html
- ../GolfExample_TCAPI/GolfAssessment/interactions.playing_1

Additionally, our xAPI bookmarklet will take the URL for any visited webpage and use it as an Activity "id" as the 'object' of a statement automatically.

Definition

Along with the "id", an Activity object may contain, and should for statements, a 'definition' property that points to an object itself that contains information about how that Activity is used, can be displayed, etc. It is important to remember that an activity has only one logical definition, even though you can include different definitions in separate statements without error. The LRS and statement consumers will have to pick what they consider to be the "right" definition and are free to do so as they choose.

Two optional (but recommended) properties are straightforward, specifically 'name' and 'description', with each being assigned a language map value that contains human-readable information about the Activity. A new property that was added in 1.0.0, 'moreInfo', provides for including a URL (IRL), a resolvable location, with more human readable information about an Activity. The Activity definition is one of the objects in the Experience API that allows for arbitrary extensions via an 'extensions' property (extensions are worth a whole post, so plan for one soon).

Finally, the Activity Definition object may contain a "type" property which must have a URI (IRI) as its value. Activity types are very similar to Verbs in a number of ways. Although they do not include a separate 'display' property, they should be generically re-usable, may resolve to metadata, and are included in our Registry. When defining a new Activity via a Definition object, the creator should take the time to determine whether there is an existing activity type that matches their activity before creating a new one. There is a nice list of pre-existing activity types that were borne out of the specification process and approved by ADL. We will be adding more to the Registry very soon, as well as accepting submissions from the community in a curated fashion.

```
{
    "id": "../GolfExample_TCAPI/GolfAssessment.html",
```

```
    "definition": {
        "name": {
            "en-US": "Golf Example Assessment"
        },
        "description": {
            "en-US": "An Assessment for the Golf Example
course."
        },
        "type": "http://adlnet.gov/expapi/activities/assessment"
    },
    "objectType": "Activity"
}
```

Similar to how the specification includes one pre-defined Verb (see “voided”), one activity type in particular is called out by the specification to have special meaning, namely an “Interaction Activity”. This activity type is rooted in the e-learning community and carries with it special properties that may be defined in the activity’s definition.

Interaction activities should have a type designated as ‘http://adlnet.gov/expapi/activities/cmi.interaction’, and are required to have an ‘interactionType’ property. For those not familiar with the common “interaction” term in the e-learning community, think of it as a question on a quiz (which is known as an “assessment”). The specification enumerates the list of possible interaction types and the associated properties that are added for each type (which also deserves its own post, man we have a lot to write).

```
{
    "id":
"../GolfExample_TCAPI/GolfAssessment/interactions.handicap_3",
    "definition": {
```

```
        "description": {
            "en-US": "A 'scratch golfer' has a handicap of ___"
        },
        "type":
"http://adlnet.gov/expapi/activities/cmi.interaction",
        "interactionType": "numeric",
        "correctResponsesPattern": [
            "0"
        ]
    },
    "objectType": "Activity"
}
```

Parts of a Statement

If the attention to detail paid to the identification and structure of an activity isn't sufficient to express its importance to xAPI, then the sheer number of places an Activity can be used will. As indicated by the examples above, a common pattern for the creation of statements is to include an Activity as the target, or specifically the 'object,' of a statement. The "Actor-Verb-Activity" is by far the most commonly used statement pattern to date.

```
{
    "actor": {
        "mbox": "mailto:info@tincanapi.com"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/attempted"
    },
```

```
    "object": {
        "id": "../GolfExample_TCAPI",
        "definition": {
            "name": {
                "en-US": "Golf Example - xAPI Course"
            },
            "description": {
                "en-US": "An overview of how to play the great
game of golf."
            },
            "type": "http://adlnet.gov/expapi/activities/course"
        },
        "objectType": "Activity"
    }
}
```

Moving beyond the ‘object’ property, Activities are an essential part of building context for a statement, so much so that “context” is a property of a statement that we haven’t gotten to in our Deep Dive series yet, but wherein there is a ‘contextActivities’ property that itself takes lists of activities. This is where the relationships amongst activities as mentioned above is codified in a statement, and to do so, Activity objects themselves are included. Within the ‘contextActivities’ object, there is the potential for four lists of activities, specifically ‘parent’, ‘grouping’, ‘category’, and ‘other.’ In each case, one or more activities are used to provide context for the rest of the statement. The ‘parent’ list suggests a very direct relationship, one that is potentially recursive through multiple “generations.” The other three types provide for more indirect relationships and are designed to be maximally flexible, but do put more onus on reporting systems to make correct connections amongst activities.

```
{
```

```
    "actor": {
        "mbox": "mailto:info@tincanapi.com"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/experienced"
    },
    "object": {
        "id": "../GolfExample_TCAPI/HavingFun/MakeFriends.html",
        "definition": {
            "name": {
                "en-US": "How to Make Friends on the Golf
Course"
            },
            "description": {
                "en-US": "An overview of how to make friends on
the golf course."
            }
        },
        "objectType": "Activity"
    },
    "context": {
        "contextActivities": {
            "parent": [
                {
                    "id": "../GolfExample_TCAPI",
                    "objectType": "Activity"
```

Beyond Statements

Just as we saw with Agents in "Deep Dive: Actor/Agent", Activities are used outside of statements as well. As a key component of statements, they need to be query-able. Activity objects are matched through the statements query resource by passing the 'id' property of the object as an "activity" parameter, this matches statements where the Activity is the 'object' of the statement. To retrieve statement results where the activity is the 'object' or in other locations of the statement, set the 'related_activities' query flag to "true," (particularly important when we want to get all statements from a nested activity using one of the 'contextActivities' slots.)

As with Agents, again, Activities get their own API methods as well. Activities have a profile for storing arbitrary data that can be used across Agents for all instances of that Activity. The Tetris game example from the xAPI Prototypes uses the Activity Profile API to store a list of high scores for the game (which is the base Activity). Each time a player finishes a game, that Activity Profile is accessed to see if their score makes the top ten, and if it does, then it is inserted into the proper rank location and the profile data is saved back to the LRS. Along with the Activity Profile API, an Activity 'id' is a required parameter when accessing the State API. State is then defined as arbitrary data associated with the combination of a unique Agent and a unique Activity (we'll ignore 'registration' for the time being).

Be Creative

The xAPI ecosystem is in its infancy and everyone has a chance to contribute to how statements will be built, how activities can be related, and the types of things we can

track. This is the chance to be influential on the community and decide what kind of data model is possible, and likely the most malleable part of the specification.