xAPI Deep Dive Object

So many objects, so little time…”Guacamole is extra, is that okay?”

Most anyone first encountering xAPI will find the “I did this” or “I did something” pattern for statements, we’ve used it once or twice ourselves on this site. This refers to the overall basic structure of a xAPI statement, in other words, the “actor-verb-object” pattern. The “something” then is the object of the statement, and correspondingly the specification includes an ‘object’ property as a required portion of a xAPI statement.

This seems straightforward; we’ve already seen in this series that the ‘actor’ property requires an Agent/Group object and the ‘verb’ property requires a Verb object, but the tricky part is that we don’t get a specific kind of object for use in the ‘object’ property. Instead we get a choice, and just like at Chipotle, making choices is hard. (Chicken burrito bowl, with a bag of chips by the way.) Our choice, though not as delicious, is amongst an Activity, an Agent, a Statement Reference, and a Sub-statement.

Activity

“Activities as objects” is the staple on the menu, the burrito of xAPI statements, and by far the most used structure. We explored Activity objects in depth in “Deep Dive: Activity”, covering very quickly that an Activity could be used as the value of the ‘object’ property of statements, and that doing so makes them query-able by that Activity. Consuming these kinds of statements is straightforward and somehow makes them feel self contained. I’ll assume you can find statements with this structure on your own — there are a couple in the Activity post and thousands in our public LRS.

Agent/Group

Coming in a close second on the menu is my favorite, the burrito bowl. Currently not as common, this statement structure appeals to me because it starts to extend us beyond

the common paths of other activity streams and further opens up possibilities related to social networking analysis. Just as with Activities, we covered that Agents/Groups can be the 'object' of statements in "Deep Dive: Actor/Agent" and that they too are query-able, though in this case requiring the "relatedAgents" query parameter. What I like to call Agent-Agent ("Double Agent"? nah, that's just bad) statements don't quite standalone, sometimes they require a fork, and go quite nicely with chips, but either way, you are going to have to dig into them to get the deliciousness out.
Some example usages of Agent-Agent statements:

- Brian contacted Mike.
- Brian was introduced to Tim.
- Mork was tutored by Mindy.
- Dr. Pepper met with Patient Zero.
- Mr. Obama defeated Mr. McCain and Mr. Romney (look a Group!).

In a couple of these statements, we are left without much context that seems very pertinent in the Activity as object world, but if our primary concern is about relationships between persons (or groups of people) then these statements can simplify an activity provider's work. In the last of these, because of who the Agents are, we can glean a significant amount of context. (The concept of Context I'll talk more about a future deep dive post.)

Here is an example Agent-Agent statement. Note that the 'objectType' property is required when the Agent is in the 'object' of a statement:

```
{
    "actor": {
        "name": "Brian Miller",
        "mbox": "mailto:brian.miller@scorm.com"
    },
    "verb": {
        "id": "http://id.tincanapi.com/verb/contacted",
        "display": {
```

```
            "en-US": "contacted"
        }
    },
    "object": {
        "objectType": "Agent",
        "name": "Mike Rustici",
        "mbox": "mailto:mike.rustici@scorm.com"
    }
}
```

Statement Reference

For those wanting to cater to a healthier lifestyle, there is always the salad. A Statement Reference is a special kind of object — it is essentially a wrapper around the identifier for an existing statement. But like with the salad, a statement using a Statement Reference as 'object' carries significant weight with it because all of the importance of the referenced statement can reflect on the referencing statement. (Cause let's face it, a salad is really just a burrito bowl with the lettuce on the bottom and dressing on the side.) A Statement Reference object has two properties and both are required: the 'objectType' property which must have a value of "StatementRef" and an 'id' with a value of a pre-existing statement. For example:

```
{
    "id": "fe2d144f-a5f5-4866-b498-620420fb3a9b",
    "objectType": "StatementRef"
}
```

There are a number of use cases where a Statement Reference makes sense as the object of a statement. Naturally, all of them relate to communications about statements, and some capture social activities common in other stream-based systems. These statements may look like the following:

- Brian favorited statement ‘c92dbac8-4a7f-47ac-a508-64136199c568’
- Ben commented on statement ‘1c580b1b-ab27-4836-9131-9be841139bf9’
- Tim trusted statement ‘a6227fe3-2caa-425a-a939-45cc661445cf’
- Grumpy Cat voided statement ’89f3403f-92e7-462c-a68c-547633533439’

That last is particularly important because it is a form that is predefined in the xAPI specification. The “voided” verb, specifically the Verb with ID “http://adlnet.gov/expapi/verbs/voided”, carries special meaning as covered in “Deep Dive: Verb”. A full voiding statement looks like:

```
{
    "actor": {
        "name": "Auto Test Learner",
        "mbox": "mailto:auto_tests@example.scorm.com",
        "objectType": "Agent"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/voided",
        "display": {
            "en-US": "voided"
        }
    },
    "object": {
        "id": "29d943ca-fb8f-4e85-ace4-cec8e157ba78",
        "objectType": "StatementRef"
    }
}
```

After a statement of this form has been issued, the referenced statement is marked as voided and will no longer be included in the statement stream. The statement itself is

still available in the LRS, but must be accessed in a direct way, and the voiding statement itself takes the original's place in the stream.
Taking out the statement IDs and replacing them with more readable versions of a statement shows how these types of statement references can start to be put togther, such as:

- Brian brokered "Tim bought house from Mike"
- Samuel refereed "USA defeated El Salvador"

Sub-Statement

Really, the only thing left in my analogy is the tacos and the kid's meal, and what better way to describe a Sub-Statement! A Sub-Statement has all of the basic parts of a Statement itself, the "actor-verb-object" pattern is still there, but it can't stand alone. (Who over the age of 10 really eats a single taco?) Some properties of normal Statements are forbidden to exist in a Sub-Statement, and a Sub-Statement has to have an 'objectType' property set to "SubStatement".

```
{
    "actor": {
        "name": "Brian Miller",
        "mbox": "mailto:brian.miller@scorm.com"
    },
    "verb": {
        "id": "http://id.tincanapi.com/verb/planned",
        "display": {
            "en-US": "planned"
        }
    },
    "object": {
```

```
        "objectType": "SubStatement",
        "actor": {
            "name": "Brian Miller",
            "mbox": "mailto:brian.miller@scorm.com"
        },
        "verb": {
            "id": "http://id.tincanapi.com/verb/ran",
            "display": {
                "en-US": "ran"
            }
        },
        "object": {
            "id":
"http://id.tincanapi.com/activity/sample/NashvilleMarathon"
        }
    }
}
```

The English form of Sub-Statements tends to read a little funny, particularly with verbs usually being seen in the past tense. The specification calls out one particular use case — the intention of doing something. In this way, Sub-Statements can be used to indicate that something will happen in the future rather than recording something that has happened.

- Brian planned “Brian ran the Nashville marathon”
- Brian un-planned “Brian ran the Nashville marathon”
- Tim scheduled “Ben attended ADL conference call”
- Jena contracted “Mr. Clean provided Jenafits”

To date, statements with Sub-Statements as the ‘object’ have probably seen the least amount of use in xAPI applications, but they have plenty of room to grow up, just as the kid’s meal does. For what is worth, my daughter is proof that you can have a single hard shell taco with just cheese, after all, you get your own little bag of chips and chocolate milk! Who doesn’t love chocolate milk?

Get Creative

In “Deep Dive: Activity”, I talked about the creative possibilities stemming from the malleability of Activities, but the flexibility inherent in the ‘object’ property’s value types takes the creative potential to the next level. Each of these types of objects brings a different dimension to the Experience API, some of which are not easily expressed in other types of streams. And while each ‘object’ type is important in its own right, the verb-object combination is the relationship within a statement that allows it to be so expressive. Utilizing all of the options in different pairings allows learning systems and non-learning systems alike to frame a pattern of use that elevates reporting and comprehension beyond what has been possible in the past.

The only remaining question is “if I have a burrito bowl today, Wednesday, do I get a burrito, a salad, or another bowl (!) on Thursday?”

Go now, make statements!