Deep Dive: Extensions

By Brian Miller on October 17, 2013

Posted in: Best practices, Deep dive, Statements

Several times throughout the Deep Dive series I've mentioned "catch all" objects and a future post — here it is. The framers of the Experience API specification knew that the overall structure of a statement, particularly with its oft mentioned Actor-Verb-Object pattern, could capture a great deal of information about learning experiences, but they also realized that there was no way for them to account for all types of experiences that people wished to record. So they left an out in the form of 'extensions' properties.

These 'extensions' properties take a special kind of object where the set of available properties is not known ahead of time, unlike all other objects in the specification. It is this unique structure that leaves it up to the Activity Provider to decide what to capture, and to own how it will be captured.

The 'extensions' property takes as its value an object where the properties of that object are strings in the form of URIs and the values can take any form, including objects. Using URIs as the properties allows for the identifiers to be owned via domain ownership, and therefore prevents the possibility of collisions as long as people respect that ownership. This means that to create (or coin) a new Extension property you should do so in a domain that you own, control, or have been given permission to use.

‹ Blog

Here are two example Extension objects, one used to include an ISBN (a book identifier), and the other used to specify a starting point and ending point (perhaps page numbers):

```
{
    "http://id.tincanapi.com/extension/isbn":
"978-1449304195"
}
```

```
{
    "http://id.tincanapi.com/extension/ending-
point": 36,
    "http://id.tincanapi.com/extension/starting-
point": 48
}
```

The 'extensions' property can be used in multiple locations within a statement, specifically the Activity Definition, Context, and Result objects. The first example above, the ISBN, would be a perfect fit to include in an Activity Definition for a book Activity. Another key is that specific 'extensions' properties can be used in any of these positions. In other words, the starting/ending point properties could make sense in the context of one statement and the result of a different one. The second example above might be used by a teacher to assign a student a set of pages to read which might be included in the context of a statement about the assigning, while the same object might be used in the result of the statement capturing the student's reading of those pages.

As mentioned, the format of the value for an Extension property is left open which is both maximally flexible for Activity Providers and problematic for reporting systems. In the above example the page numbers are captured as integers (or more specifically numbers), but some other starting point may need to be captured as a string. There is no right answer as to whether an extension will always have the same formatted value or not.

Using different types of values with the same identifier is problematic for systems that will leverage the data from the statements later. In some cases, distinguishing the format will be straightforward and reporting systems will be able to handle it relatively easily. In other cases, there will be a clear format that an extension value must use. For example, the "http://id.tincanapi.com/extension/geojson" extension has a very precise value format based on the GeoJSON specification. The ISBN extension property used earlier always takes a string but a user must examine the length to properly handle it. Still other times it may be left up to the context in which the property is used as to how to handle its value. This is the trickiest of cases and one which has been coming up in conversations around Experience API more and more frequently. Extension coiners should consider including in the description of their extension (more on that below) information about what form the value should take. The example of the ISBN specifically includes:

> Value should be either a 10 digit ISBN or 13 digit ISBN string. Either value is acceptable as implementing systems can easily distinguish the two based on the length of the value.

This allows developers to understand whether an extension will serve their purpose or not, and by conforming to the definition provided, they can expect their usage to interoperate with others' usages.

Unlike other objects in a Statement, because the extension property URIs are the Extension object's properties themselves, there is no place to provide metadata information about the property itself. In other words, there is no "local" way in the statement to provide a human readable name for the extension property or the description needed to explain how that extension property is to be used. This is another key to using URIs as the representation of the properties — many of them are easily convertible to URLs. Specifically using a URL and making that address resolvable enables a way to fetch metadata about the extension property. The format of the metadata is specified in the Experience API, when resolving the URL with a request for content-type of "application/json" the host should return a JSON object including a "name" and "description" properties whose values are language map objects as seen elsewhere in the specification.

By example, fetching the resource for an extension property such as "http://id.tincanapi.com/extension/tags" will return:

```
{
    "name": {
       "en-US": "tags"
    },
    "description": {
        "en-US": "A list of arbitrary tags to
associate with a statement.
                  Value of the extension should
be an array with each tag being
                  a string value as an element of
the array."
    }
}
```

As described in other posts in this series, The Registry has been created to catalog available extensions to assist with interoperability. New Extension properties can be created in the "id.tincanapi.com" domain using the web interface to ensure that they can always be resolvable. Other extension properties can also be recorded to make them easier to find. There is a nice and ever growing list of extensions already listed, some of which we pre-populated in anticipation of their need by the community. Browsing the list is an excellent way to see the extent of the varied ways extension properties can and will be used.

Additionally, 'extensions' are also prescribed for use in the "/about" resource that an LRS must provide, though including the property isn't specifically required at this time. An example is the "powered-by" extension as part of the /about resource result on SCORM Cloud, it returns:

```
{
    "extensions": {

"http://id.tincanapi.com/extension/powered-by": {
            "name": "xAPI Engine",
            "homePage": "../lrs-lms/lrs-for-lmss-
home/",
            "version": "2012.1.0.5039b"
        }
    },
    "version" : [ "1.0.0" ]
}
```

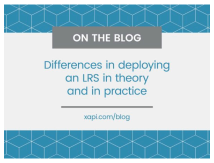
"With great power comes great responsibility," or so it is said, and using "extensions" is no different. With its flexibility it is simple to turn to the easy way out and just shove any data into an Extension when selecting a more complex statement structure or being more specific about an Activity may be more suitable. Even within our own walls, we recently had a conversation about using an Activity with a specific Activity Type rather than using an Extension property with a value that would be a URI itself. It turned out we didn't need to use "extensions" at all and were better served by using Context activities in its place.

Go now, make statements!

Related posts

What vendors can do to prepare for xAPI 2.0

Around the Rustici office, it's fairly common to hear that eLearning standards are often slow movers. xAPI turned 10 years old in April, the oldest versions...



Differences in deploying an LRS in theory and in practice

Theory teaches us how things should work, in a perfect world. Experience teaches us how they might—or might not—work, in an imperfect

Honoring our 2020 MVP blog authors

At Rustici Software, I'm lucky to work with colleagues who enjoy sharing their expertise with others through blog writing. What better way to show my appreciation...

Read more

world. What better way...

Read more

Read more

Privacy Badger has replaced this Disqus widget

Allow once

Always allow on this site

Brian Miller

Brian Miller is the Senior Director of Engineering and is also the most pedantic person at the office, which is saying something. That skill makes him great at ensuring our products support the standards, which is precisely what he spends his days doing. Brian is a IEEE LTSC voting member working on the advancement of learning standards, like xAPI and cmi5.

Select Language ▼