Building an xAPI profile

One thing that particularly caught my attention when researching xAPI and SCORM were the xAPI Activities.

An activity is a group of verbs and activities that describe actions specific to a group or a sub-group. They are part of the answer to my question:

> ***How can I build the statement: Person A implemented technology X?***.

This post will discuss some of the elements in a profile and provide some introductory information about them. While profiles are technical in nature, I believe that they are also useful for designers and teams looking to implement the verbs in a profile. When xAPI validators become available outside LRS implementation it will be nice to have the profiles to do local validation as well.

Planning what do we want to say

Before we start looking at how to write the JSON-LD for our profile we need to define what we want to create and what it means.

xAPI statements use an **actor — verb — object/activity** triplet model. The profile will deal with verbs and activities and will let the implementor deal with authenticating the actor and providing actor information to complete the triplet.

We need to document what we're defining and what external verbs and activities we want to use. It's particularly important not to reinvent the wheel.

Examples of activities:

- David *read* an *ebook*
- Kay was *technical lead* for a *project*
- Chris *implemented* an *API*
- Paul *completed* an *online course*

Using these as examples we'll build a profile that described these activities. In doing so we'll see how to leverage existing xAPI statement components in addition to creating our own.

The information in the xAPI Vocab Server is a good starting point to see what's out there in terms of verbs and activities and how have other people defined things that we may want to use or define.

However, we need to be sure that those external verbs and activities actually mean what we need to.

A quick overview

The following table, taken from the xAPI profile specification, describes the structure of an xAPI profile.

Property	Type	Description	Required
`id`	IRI	The IRI of the Profile overall (not a specific version)	Required
`@context`	URI	SHOULD be `https://w3id.org/xapi/profiles/context` and MUST contain this URI if array-valued.	Required
`type`	String	MUST be `Profile`.	Required
`conformsTo`	URI	Canonical URI of the Profile specification version conformed to. The Profile specification version of this document is https://w3id.org/xapi/profiles#1.0.	Required
`prefLabel`	Object	Language map of names for this Profile.	Required
`definition`	Object	Language map of descriptions for this Profile. If there are additional rules for the Profile as a whole that cannot be expressed using this specification, include them here, or at the seeAlso URL.	Required
`seeAlso`	URL	A URL containing information about the Profile. Recommended instead of especially long definitions.	Optional
`versions`	Array	An array of all Profile version objects for this Profile.	Required
`author`	Object	An Organization or Person.	Required
`concepts`	Array	An array of Concepts that make up this Profile.	Optional
`templates`	Array	An array of Statement Templates for this Profile.	Optional
`patterns`	Array	An array of Patterns for this Profile.	Optional

We'll explore things in more detail as we progress through building the profile.

Building the profile

Before we start adding verbs and activities we need to define some properties about the profile we are creating.

Some of these items use a `w3id.org` url. These will redirect to the correct URL and are meant to ensure that the files will remain available if the original URLs are removed or change address.

`@context` is a JSON-LD term and it is used to provide a way to map terms to IRI

> Simply speaking, a context is used to map terms to IRIs. Terms are case sensitive and most valid strings that are not reserved JSON-LD keywords can be used as a term. Exceptions are the empty string "" and strings that have the form of a keyword (i.e., starting with "@" followed exclusively by one or more ALPHA characters (see [RFC5234])), which must not be used as terms. Strings that have the form of an IRI (e.g., containing a ":") should not be used as terms.
>
> Source: The Context in JSON-LD 1.1

I use a URL/URI in the `ns.rivendellweb.net` namespace for the profile ID. I've used this in the past to represent XML name spaces. I'm not sure if I'll have to actually put something there or not.

xAPI profiles use IRIs (Internationalized Resource Identifiers) and not URIs.

The main difference between an IRI and a URI/URL as used on the web is the character set they use:

- IRIs use UCS, the ISO standard for Unicode, thus allowing for the identifiers to use characters outside ASCII
- URIs were specified to only use ASCII character set, making it impossible to use languages other than English

`type` is fixed and should always have the value of `profile`.

```
{
  "@context": "https://w3id."https://w3id.org/xapi/profiles/context""id"/
  "type": "P": ""type": "Profile",
```

`conformsTo` referes to the version of the xAPI profile spec that this profile follows. For now, the only version available is 1.0 and is represented by the `https://w3id.org/xapi/profiles#1.0` IRI

```
  "conformsTo": "https://w3id.org/xapi/profiles#1.0",
```

`prefLabel` is an array of the name of the profile in one or more languages. The developer profile only uses US English, but we could easily add languages.

```
  "prefLabel": {
    "en-US": "Rivendellweb Developer Profile"
  },
```

`definition` is an array of strings that describe the profile in different languages. Again, this example profile only uses US English.

`seeAlso` points to an IRI containing more information about the profile than what you can safely store

```
  "definition": {
    "en-US": "Rivendellweb Developer Profi"Rivendellweb Developer Profile"
```

The `author` can be a person or organization. Regardless of the kind of author you use, name and URL are required.

```
  "author": {
    "type": "Person",
    "Person""name"s Araya",
    "url": "": ""url": "https://publishing-project.rivendellweb.net/"
  },
```

We can have one or more versions of a profile. To indicate a single version, we use a single child in the versions array using a unique ID and a `generatedAtTime` attribute to tell when the profile was created.

```
  "versions": [{
    "id": "https://ns.rivendellweb."https://ns.rivendellweb.net/xapi/devpr
  }],
```

Once we update the profile we indicate so with additional children of the versions array.

To indicate that this is a later version, use the `wasRevisionOf` attribute and point it to the prior version of the profile.

Hypothetically, if we had 3 versions of the profile we could code them like this:

```
versions [{
        "id": "https://ns.rivendellweb"https://ns.rivendellweb.net/xapi/de
        "generatedAtTime"20:20:20Z"
    },
    {
        "id": "https:/": ""id"ndellweb.net/xapi/devprofile/v2.0",
        "wasRevisionOf": ["https://ns.riv": ["lweb.net/xapi/devprofile/v1
        "generatedAtTime": "201": ""generatedAtTime""generatedAtTime"  "id
        "generatedAtTime": "2010-01-14T12:13:14Z"
}]
": ""generatedAtTime": "2010-01-14T12:13:14Z"
}]
```

Concepts

The `concepts` section of a profile contains both `ActivityType` and `Verb` children.

The `ActivityType` concept indicates the object in the `actor-verb-object` triplet of our statements. This is the recipient of the action indicated on the verb.

The ID must be unique across all elements in the profile.

For an object, the `type` attribute must be `ActivityType`. We'll see other when we cover verbs and later when we talk about template and patterns.

The definition is important. This tells users what the concept means and how it should be used. We can have multiple versions of the definition in different languages.

`inScheme` tells what specific version of the profile this concept refers to.

`prefLabel` is an array of prefered names for the concept in different languages

There is a set of optional attributes that help indicate relationships between concepts in this profile and concepts both in this profile and others. This means that we no longer have to define every verb and every activity in our profile, we can link to other profiles and leverage their work.

deprecated (Boolean)
A boolean. If true, this Concept is deprecated.

broader (Array)
An array of IRIs of Concepts of the same type from this Profile version that have a broader meaning.

broadMatch (Array)
An array of IRIs of Concepts of the same type from a different Profile that have a broader meaning.

narrower (Array)
An array of IRIs of Concepts of the same type from this Profile version that have a narrower meaning.

narrowMatch (Array)
An array of IRIs of Concepts of the same type from different Profiles that have narrower meanings.

related (Array)
An array of IRIs of Concepts of the same type from this Profile version that are close to this Concept's meaning.

relatedMatch (Array)
An array of IRIs of Concepts of the same type from a different Profile or a different version of the same Profile that has a related meaning that is not

clearly narrower or broader.

Useful to establish conceptual links between Profiles that can be used for discovery.

exactMatch (Array)
: An array of IRIs of Concepts of the same type from a different Profile or a different version of the same Profile that have exactly the same meaning.

Some additional recommendations from the xAPI specification:

- **related** MUST only be used on Concepts that are deprecated to indicate possible replacement Concepts in the same Profile, if there are any.
- **relatedMatch** SHOULD be used to connect possible replacement Concepts to removed Concepts from previous versions of the same Profile, and for possible replacement Concepts in other Profiles of deprecated Concepts, as well as other loose relations.
- **exactMatch** SHOULD be used rarely, mostly to describe connections to vocabularies that are no longer managed and do not use good URLs.


```
}],
"concepts": [{
    "id": "https://ns.rivendellweb.net/xapi"https://ns.rivendellweb.net/
    "definition": {
      "en-US"mmunicated by or to or between individual actors or groups
    },
    "inScheme": "https://ns.r": ""inScheme": "https://ns.rivendellweb.ne
    "prefLabel": {
      "en-US"  {
    "id": "https://ns.rivendellwe": ""id"pi/devprofile/v1.0.0/activities
    "type": "ActivityType",
    "de": ""type" {
      "en-US": "A b": ""definition": {
      "en-US"t, file, or interactive application with optional formattin
    },
    "inScheme": "https://ns.rivendellweb.net/": ""inScheme"e/v1.0.0",
    "prefLabel": {
      "en-US": "e-book"
  ": ""prefLabel": {
      "en-US": "e-book"
```

```
      }
    },
```

The `Verb` concept is the action the actor did to the object, the verb component of the `actor-verb-object` definition of our action

```
    {
      "id": "https://ns.rive"https://ns.rivendellweb.net/xapi/devprofile/v
      "definition"    "en-US": "The a": "ity involving multiple actors wor
      },
      "inScheme": "https://n": ""inScheme".net/xapi/devprofile/v1.0.0",
      "prefLabel": {
        "en"prefLabel""prefLabel": {
        "en-US"
      "id": "https://ns.rivendellweb.net/xa": ""id"ofile/v1.0.0/verbs/wat
      "type": "Verb",
      "exactMatch": ["htt": ""type": "Verb",
      "exactMatch", "https://w3id.org/xapi/adb/verbs/watched"],
      "definition": {
        "en-US": "Indicates t"https://w3id.org/xapi/adb/verbs/watched""de
      },
      "inScheme"S": "watched"
      }
    }
  ]
}
": ""prefLabel": {
        "en-US": "watched"
      }
    }
  ]
}
```

Extensions

Activities may define one or more extensions to refine other parts of the profile. The xAPI profile specification defines three types of extensions:

- ContextExtension in context
- ResultExtension in result
- ActivityExtension in an Activity Definition.

For example, if we're defininig a a conference presentation as an activity, we could provide additional information about the presentation like its title as an activity extension and we could provide information about the conference where user presented as a context extension

I chose an inline schema to validate the object. It is valid JSON Schema fragment validated against the latest draft of the JSON Schema specification using JSON Schema Validator. The quotation marks inside the `inlineSchema` element are escaped to make sure the JSON validates.

The other option is to use the `schema` property and a schema defined in an external JSON file.

```
{
    "id": "https://n"https://ns.rivendellweb.net/xapi/devprofile/v1.0.0/re
    "inScheme": "https://ns.rivendellweb.net/xapi/devprofile/v1.0.0",
    "prefLabel"        "en": "C": "rence Information"
    },
    "definition": {
    "definition": {
        "en": "Stores the conference a presentation was delivered at."
    },
    "recommendedVerbs"s.rivendellweb.net/xapi/devprofile/v1.0.0/verbs/pres
      "https://ns.rivendellweb.net/xapi/devprofile/v1.0.0/verbs/attended"
    ],
    "contentType": "appl"https://ns.rivendellweb.net/xapi/devprofile/v1.0
}
": {\"type\": \"string\",\"required\": true}}"
```

The only remaining item to research is how to integrate these extensions into the activities they extend. How do you reference the extension inside the verb or object it extends?

Templates and Patterns

There are two more areas of a profile: templates and patterns.

Templates (formally Statement Template Rules) describe a location or locations within Statements using JSONPath, then describe the restrictions on the value(s) there, such as inclusion, exclusion, or specific allowed or disallowed values for the locations that match the path.

Patterns indicate a group of one or more statments happen in the order the pattern defines.

Both templates and patterns are more technical than previous sections. While they are still necessary to understand the profile they are a part of, they are less necessary to understand at the beginning of the process. I will continue reviewing these and will add new posts with more information about these elements of a profile.

Links and resources

- xAPI profiles
    - Part One: About xAPI Profiles
    - Part Two: xAPI Profiles Structure
    - Part Three: xAPI Profiles Communication and Processing
- xAPI Vocab Server
- Authoring Profiles
- Profile Recipes vs. xAPI Profiles
- JSON for Linking Data