**1.0 Introduction**

The Experience API (xAPI) is a technical specification that aims to facilitate the documentation and communication of learning experiences. It specifies a structure to describe learning experiences and defines how these descriptions can be exchanged electronically.

The xAPI is an effort of the Advanced Distributed Learning (ADL) Initiative. ADL was established in 1997 to standardize and modernize training and education management and delivery in the US. Since that time, there has been a growing realization of the need to track learning experiences of individuals beyond formal, structured computer-based training. In assessing candidates' suitability for positions or their capability for performing various tasks, there is a need to consider a wide range of formal and informal learning experiences, both on and offline. That information, more often than not is scattered across a wide variety of sources.

Out of this perceived need, the xAPI community and specification were born. xAPI assumes that:

* There is a need to be able to analyze information about learning experiences and their outcomes distributed across a wide variety of sources, platforms and technologies.
* Developing a commonly-accepted framework for gathering, storing and exchanging this information represents the best way of achieving this.

The goals of the xAPI are:

* To make it easier to understand and compare learning experiences and their outcomes recorded across a wide variety of contexts, platforms and technologies.
* To maximize interoperability of services which create, gather, store and process information about learning experiences.
* To provide a guide to those who want to build applications that conform to and implement this specification.
* To provide criteria against which conformance to this specification can be tested.

The document that follows sets out the xAPI specification which is designed to achieve these goals.

This is a definitive document which describes how the Experience API is to be implemented. It is a technical document authored specifically for individuals and organizations implementing this technology with the intent of such individuals developing interoperable tools, systems and services that are independent of each other and interoperable with each other.

Whenever possible, the language and formatting used in this document is intended to be *considerate* of non-technical readers because various tools, systems and services are based on the specification set described below. For this reason, sections that provide a *high-level overview* of a given facet of the Experience API are labeled **description** or **rationale**. Items in this document labeled as **requirements**, **details** or **examples** are more technical.

This specification is split into three parts. Part one is this introduction. It offers some background, high-level summaries and direction on how to read the rest of the specification.

Part two of this specification defines a data model for various data objects that are used in this specification. The most significant object within the xAPI data model is the "Statement" object. This specification defines the properties of the Statement object (including "Actor", "Verb", "Object", "Result", and "Context") and the rules of syntax for the values of those properties and how they are represented. This part helps to ensure that services implementing the specification follow a consistent data structure.

Part three of this specification sets out the transfer methods that must be used when communicating information about learning experiences between services that adhere to the specification. This includes the format of requests and the expected responses. Note that communication in xAPI is not restricted to a "Learning Record Store" (LRS) receiving data from "content". LRSs can communicate with services ranging from "Learning Record Providers" to "Learning Record Consumers" to other LRSs. xAPI follows the guidelines of the REST software architecture style, and as such data is transferred via HTTP requests and responses. Part three also defines security methods allowing for the trusted exchange of information between the LRS and trusted "Clients".

**2.1 MUST / SHOULD / MAY**

There are three levels of obligation with regards to conformance to the xAPI specification identified by the terms MUST, SHOULD and MAY. A service or system that fails to implement a MUST (or a MUST NOT) requirement is non-conformant. Failing to meet a SHOULD requirement is not a violation of conformity, but goes against the recommendations of the specification. MAY indicates an option, to be decided by the developer with no consequences for conformity. Usage of these terms outside of requirement language does not designate a requirement and is avoided whenever possible. Complete definitions of MUST, SHOULD, MAY, MUST NOT and SHOULD NOT are found in [RFC 2119](https://www.ietf.org/rfc/rfc2119.txt).

The use of an asterisk\* following SHOULD indicates a very strong recommendation. It is planned that these recommendations will become MUST requirements in a future version. Not following these recommendations could risk interoperability and and/or lead to various other issues depending on the specifics of the recommendation. These recommendations cannot be MUST requirements within this version as these would be breaking changes. The xAPI Working Group strongly encourages adopters to implement these requirements as though they were MUST requirements, while continuing to support other adopters that might not do so.

**2.2 Guidelines for Interpreting Descriptive Text and Tables**

As a rule of thumb, if the guideline appears technical or seems to be a requirement, interpret it as such. This is especially true of longer, more, detailed explanations and of tables, each of which would be unintuitive and/or lengthy to dissect into a list of requirements.

Tables are used throughout this specification to define requirements for lists of properties, parameters, etc. The tables define which properties are required, recommended and optional. Generally, the notion of "optional" relates to the service creating the object, while services receiving and interpreting the object need to be able to interpret all properties of that object. Often, properties are optional because the data may not be relevant in every context; if the data is relevant in a particular context, then it is expected the property will be populated.

If an optional property or parameter contains an object with properties that are recommended or required, then these properties are only recommended/required if the property or parameter containing them is used.

Examples are provided throughout the specification and in appendices to illustrate implementation. The content of these examples is fictional in order to illustrate the requirements of the specification and may not always illustrate the best practice approach to tracking the particular learning experience used in the example. Examples can be used to inform interpretation of requirements, but are not intended to take precedence over requirements.

Where the specification does not include requirements relating to a particular facet of implementation, that detail can be considered to be outside of the scope of this specification. It is up to the implementer to determine a sensible approach. This specification tries to avoid vagueness and will usually give a rationale even if there no requirement in a given area.

**3.0 Serialization and JavaScript Object Notation (JSON)**

Serialization is the process of translating data objects and structures into a format for storage or transmission, such that the original data object can be recreated from the resulting serialization. In some cases it might be possible to serialize a piece of data in more than one way. For example, a boolean property with a value corresponding to true might be represented as true or 1 depending on the serialization used.

xAPI follows the rules of JSON for serializations (so boolean values are represented as true or false). It might also be possible to represent the objects defined in this specification using other serializations, such as XML. This is out of scope of this specification and use of anything other than JSON to represent the objects defined in this specification is not conformant.

Even within the rules of JSON there are possible variations of how data can be serialized, especially in relation to data about time. This is significant as a number of features of xAPI rely on systems being able to determine whether two Statements are equivalent. See [Immutability and exceptions](https://github.com/adlnet/xAPI-Spec/blob/master/xAPI-Data.md#statement-immutability-and-exceptions) for more details about the properties of the Statement affected by this.

JSON allows for objects to have properties that contain empty objects. This is not recommended within xAPI; if the Statement is not intended to contain data about a property then it is expected that the property will not be used at all. All required properties are required to contain values.

## 5.0 xAPI Components

This section explains and shows graphically how different pieces of xAPI can fit together.

##### Tracking Experiences

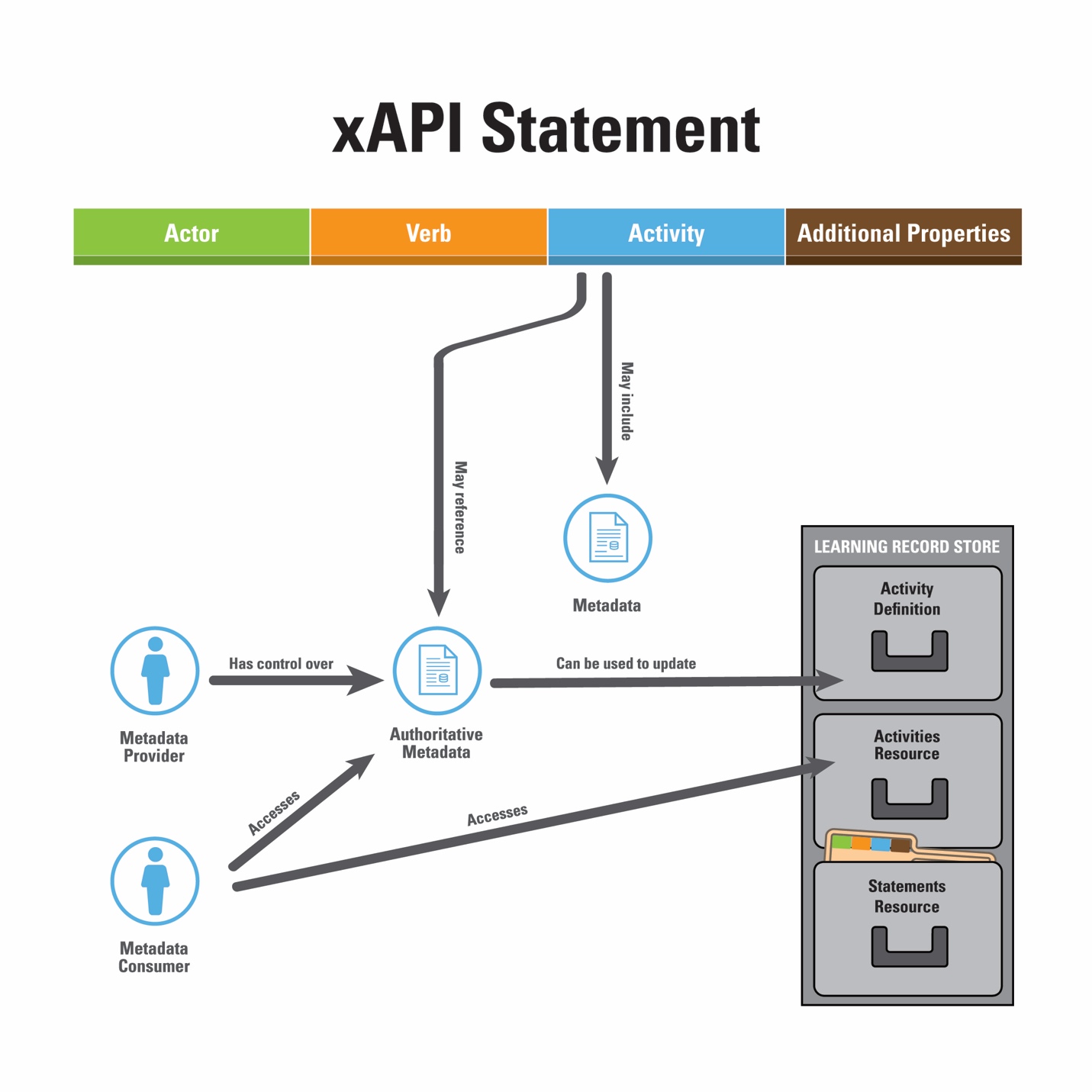
[](https://github.com/adlnet/xAPI-Spec/blob/master/xAPIDataFlow.jpg)

this shows the tracking of learning experiences. A learner has a learning experience. This experience could take place in an online course, it could be on the job or it could be part of recreation. It really could be anything. This experience is tracked, on the learner’s behalf, by a trusted Learning Record Provider (LRP). The Learning Record Provider can also be responsible for the trusted relationship between the experience and the learner. This might even include launching content for the learner and managing digital rights associated with the content.

The Learning Record Provider creates Learning Records and sends them to one or more Learning Record Stores (LRSs). The LRS stores the Learning Records and makes them available to any authorized Client. A Learning Record Consumer (LRC) is a type of Client that accesses Learning Records and makes use of them.

##### Activity Data and Metadata

###### Figure 2: xAPI Activity Data and Metadata

[](https://github.com/adlnet/xAPI-Spec/blob/master/xAPIMetadata.jpg)

Understanding how a single Activity (as uniquely identified by its IRI) is defined and described is a key concept in xAPI. Figure 2 shows this process. An Activity (as a part of a Statement) has metadata properties that may be populated within the Statement itself. This is done in the Statement's Activity Definition. The id of each Activity is an IRI which also could have metadata located at the resolution location of the IRI. Any metadata where the IRI resolves is under the control of the Metadata Provider. The Metadata Provider is also responsible for making sure the IRI is permanent and resolves correctly.

Any metadata located at where the IRI resolves is the authoritative source of metadata, and could be used to populate the LRS's canonical version of the Activity's metadata (LRS's Activity Definition) as a preference to what it receives from Statements. A Metadata Consumer can access metadata via the IRI for the authoritative version or can query the Activities Resource for the canonical version.