VERSION 1.0.0

SPECIFICATION RELEASE DATE: 26 APRIL 2013

THE ADVANCED DISTRIBUTED LEARNING (ADL) INITIATIVE

This document was authored by members of the Experience API Working Group (see list on pages 3-4) in support of the Office of the Deputy Assistant Secretary of Defense (Readiness), Director, Training Readiness and Strategy, Advanced Distributed Learning (ADL) Initiative.

This PDF copy of the specification was synchronized with the authoritative document that resides on the ADL GitHub site - https://github.com/adlnet/xAPI-Spec/blob/master/xAPI.md - on 26 April 2013 and is considered to be version 1.0.0. It represents a snapshot of the authoritative document. Please go to https://github.com/adlnet/xAPI-Spec to view the most current version of the specification.

Minor version releases, which include corrections and/or additional changes that are made to the specification on GitHub after the 1.0.0 release date, will not necessarily result in publication of an updated PDF file. Updated PDF files may be published to reflect significant changes and/or corrections to the specification.

Please send all feedback and inquiries to: helpdesk@adlnet.gov

‘

TABLE OF CONTENTS

THE EXPERIENCE API, VERSION 1.0.0

1.0 REVISION HISTORY OF THE SPECIFICATION

0.8 (ProjectTin Can API Deliverable) to 0.9 (March 31, 2012)	Rustici Software, who delivered the Project Tin Can API, made modifications to the API prior to the April 2012 Kickoff Meeting. It was voted in this meeting to move those changes into the current spec and revision to 0.9.
0.90 to 0.95 (August 31, 2012)	"Core" Verbs and Activity types were removed from the specification. References to these Verbs in results, context, interactions, and Activity definitions were also removed. It was recommended that implementers prefer community defined verbs to creating their own Verbs. • Verbs, Activity types, and extension keys are now URIs. • Restructured and added language around some of the other implementation details and scope. • Changed from using a person-centric view of Agents to a persona-centric view. • Friend of a Friend (FOAF) Agent merging requirement was removed. • Agent Objects must now have exactly 1 uniquely identifying property, instead of at least one.
0.95 to 1.0.0 (April 26, 2013)	Various refinements and clarifications including: • Adding attachments • Activity metadata is now stored as JSON rather than XML • Changes to voiding Statements • Clarification and naming of the Document APIs • Changes to querying the Statement API • Signed Statements
(May 21, 2013)	• Corrected Table of Contents section numbers • Changed URL and URI to IRL and IRI where appropriate • Fixed minor editorial inconsistencies

2.0 ROLE OF THE EXPERIENCE API

The Experience API is a service that allows for statements of experience to be delivered to and stored securely in a Learning Record Store (LRS). These statements of experience are typically learning experiences, but the API can address statements of any kind of experience. The Experience API is dependent on Activity Providers to create and track these learning experiences; this specification provides a data model and associated components on how to accomplish these tasks.

Specifically, the Experience API provides:

- The structure and definition of Statement, State, Actor (i.e., “the learner”), Activity and Objects, which are the means by which experiences are conveyed by a Activity Provider.
- Data Transfer methods for the storage and retrieval (but not validation) of these Objects to/from a Learning Record Store. Note that the systems storing or retrieving records need not be Activity Providers. LRSs may communicate with other LRSs, or reporting systems.
- Security methods allowing for the trusted exchange of information between the Learning Record Store and trusted sources.

The Experience API is the first of many envisioned technologies that will enable a richer architecture of online learning and training. Authentication services, querying services, visualization services, and personal data services are some examples of additional technologies which the Experience API is designed to support. While the implementation details of these services are not specified here, the Experience API is designed with this larger architectural vision in mind.

ADL’s Role in the Experience API 2.1

The Advanced Distributed Learning (ADL) Initiative has taken on the roles of steward and facilitator in the development of the Experience API. The Experience API is seen as one piece of the ADL Training and Learning Architecture, which facilitates learning anytime and anywhere. ADL views the Experience API as an evolved version of SCORM that can support similar use cases, but can also support many of the use cases gathered by ADL and submitted by those involved in distributed learning that SCORM could not enable.

Contributors 2.2

My thanks to everyone who contributed to the Experience API project. Many of you have called into the weekly meetings and helped to shape the specification into something that is useful for the entire distributed learning community. Many of you assisted in releasing code samples, products, and documentation to aid those who are creating and adopting the specification. I'd also like to thank all of those who were involved in supplying useful, honest information about your organization's use of SCORM and other learning best practices. Through the use-cases, shared experiences, and knowledge you have shared, ADL and the community clearly identified the first step in creating the Training and Learning Architecture--the Experience API. You are truly the community leaders on which we depend to make our training and education the very best.

Kristy S. Murray, Ed.D.
Director, ADL Initiative
OSD, Training Readiness & Strategy (TRS)

Working Group Participants 2.2.1

Panjak Agrawal	*Next Software Solutions*	David N. Johnson	*Clear Learning Systems*
Dan Allen	*Litmos*	Eric Johnson	*Planning and Learning Technologies, Inc.*
Anthony Altieri	*American Red Cross*	Patrick Kedziora	*Kedzoh*
Jonathan Archibald	*Brightwave*	John Kleeman	*Questionmark*
Avron Barr	*Aldo Ventures, Inc.*	Dan Kuemmel	*Sentry Insurance*
Steve Baumgartner		Richard Lenz	*Organizational Strategies, Inc.*
Al Bejcek	*NetDimensions*	Fiona Leteney	*Feenix e-Learning*
Marcus Birtwhistle	*ADL*	Robert Lowe	*NetDimensions*
Megan Bowe	*Rustici Software*	Tim Martin	*Rustici Software*
Jeremy Brockman		Bill McDonald	*Boeing*
Jennifer Cameron	*Sencia Corporate Web Solutions*	Brian J. Miller	*Rustici Software*
Rob Chadwick	*ADL*	Kris Miller	*edcetera Training*
Rich Chetwynd	*Litmos*	Dave Mozealous	*Articulate*
Ben Clark	*Rustici Software*	Mike Palmer	*OnPoint Digital*
Tom Creighton	*ADL*	Jeff Place	*Questionmark*
Ingo Dahn	*University Koblenz-Landau*	Jonathan Poltrack	*ADL*
Mark Davis	*Exambuilder*	Rick Raymer	
Jhorlin De Armas	*Riptide Software*	Michael Roberts	*vTrainingRoom*
Andrew Downes	*Epic*	Paul Roberts	*Questionmark*
Russell Duhon	*SaLTBOX*	Kris Rockwell	*Hybrid Learning Systems*
David Ells	*Rustici Software*	Mike Rustici	*Rustici Software*
Paul Esch	*Nine Set*	Chris Sawwa	*Meridian Knowledge Solutions*
Michael Flores	*Here Everything's Better*	Matteo Scaramuccia	
Steve Flowers	*XPConcept*	Ali Shahrazad	*SaLTBOX*
Richard Fouchaux	*Ontario Human Rights Commission*	Aaron Silvers	*ADL*
Joe Gorup	*CourseAvenue*	Greg Tatka	*Menco Social Learning*
Walt Grata	*ADL*	Steven Tevorrow	*Problem Solutions, LLC*
Jason Haag	*ADL*	Chad Udell	*Float Mobile Learning*
Doug Hagy	*Twin Lakes Consulting Corporation*	Anton Valan	*Omnivera Learning Solutions*
Luke Hickey	*dominKnow*	Melanie VanHorn	*ADL*
Thomas Ho		Nick Washburn	*Riptide Software*
Lang Holloman		Andy Whitaker	*Riptide Software*
Nikolaus Hruska	*ADL*	Lou Wolford	*ADL*
Andy Johnson	*ADL*		

Requirements Gathering Participants 2.2.2

In collection of requirements for the Experience API, many people and organizations provided invaluable feedback to the Sharable Content Object Reference Model (SCORM®), distributed learning efforts, and learning technology efforts in general. While not an exhaustive listing, the white papers gathered in 2008 by the Learning Education and Training Standards Interoperability (LETSI) group, the Rustici Software UserVoice website, one-on-one interviews and various blogs were important sources from which requirements were gathered for the Experience API specification.

Reading Guidelines for the non-technically inclined 2.3

This is a definitive document which describes how the Experience API is to be implemented across a variety of systems. 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 **details** or **examples** are more technical.

3.0 DEFINITIONS

Activity	Something with which an Actor interacted. It can be a unit of instruction, experience, or performance that is to be tracked in meaningful combination with a Verb. Interpretation of Activity is broad, meaning that Activities can even be tangible objects.
Activity Provider (AP):	The software object that is communicating with the LRS to record information about a learning experience. May be similar to a SCORM package in that it is possible to bundle learning assets with the software object that performs this communication, but an Activity Provider may also be separate from the experience it is reporting about.
Actor	An identity or persona of an individual or group tracked using Statements as doing an action (Verb) within an Activity.
Authentication	The concept of verifying the identity of a user or system. Authentication allows interactions between the two “trusted” parties.
Authorization	The affordance of permissions based on a user or system's role; the process of making one user or system "trusted" by another.
Client	Refers to any entity that may interact with an LRS. A Client can be an Activity Provider, reporting tool, an LMS, or another LRS.
Community of Practice	A group, usually connected by a common cause, role or purpose, which operates in a common modality.
Experience API (xAPI):	The API defined in this document, the product of "Project Tin Can". A simple, lightweight way for any permitted Actor to store and retrieve extensible learning records, learner and learning experience profiles, regardless of platform.
Immutable	Adjective used to describe things which cannot be changed. With some exceptions, Statements in the xAPI are immutable. This ensures that when statements are shared between LRSs, multiple copies of the statement remain the same.
Internationalized Resource Identifiers (IRI):	A unique identifier which may be a IRL. In the xAPI, all IRIs should be a full absolute IRI including a scheme. Relative IRIs should not be used. IRLs should be defined within a domain controlled by the person creating the URL.
Inverse Functional Identifier	An identifier which is unique to a particular persona or group. Used to identify Agents and Groups.
Learning Management System (LMS):	"A Learning Management System is a software package used to administer one or more courses to one or more learners. An LMS is typically a web-based system that allows learners to authenticate themselves, register for courses, complete courses and take assessments” (Learning Systems Architecture Lab definition). In this document the term will be used in the context of existing systems implementing learning standards.
Learning Record Store (LRS):	A system that stores learning information. Prior to the xAPI most LRSs were Learning Management Systems (LMSs); however this document uses the term LRS to be clear that a full LMS is not necessary to implement the xAPI. The xAPI is dependent on an LRS to function.
MUST / SHOULD / MAY	Three levels of obligation with regards to conformance to the xAPI specification. A 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 best practices. MAY indicates an option, to be decided by the developer with no consequences for conformity.
Profile	A construct where information about the learner or activity is kept, typically in name/document pairs that have meaning to an instructional system component.
Registration	An instance of a learner experiencing a particular Activity.

Representational State Transfer (REST)	An architecture for designing networked web Services. It relies on HTTP methods and uses current web best practices.
Service	A software component responsible for one or more aspects of the distributed learning process. An LMS typically combines many services to design a complete learning experience.
Statement	A simple construct consisting of `<Actor (learner)>` `<verb>` `<object>`, with `<result>`, in `<context>` to track an aspect of a learning experience. A set of several Statements may be used to track complete details about a learning experience.
Tin Can API (TCAPI):	The previous name of the API defined in this document, often used in informal references to the Experience API.
Verb	Defines the action being done by the Actor within the Activity within a Statement.

4.0 STATEMENT

The Statement is the core of the xAPI. All learning events are stored as Statements. A Statement is akin to a sentence of the form "I did this".

Statement Properties 4.1

Actor, Verb, and Object are required, all other properties are optional. Properties can occur in any order, but are limited to one use each. Each property is discussed below.

Property	Type	Description
id		UUID assigned by LRS if not set by the Activity Provider.
actor	UUID	Who the Statement is about, as an Agent or Group Object. Represents the "I" in "I Did This".
verb	Object	Action of the Learner or Team Object. Represents the "Did" in "I Did This".
object	Object	Activity, Agent, or another statement that is the Object of the Statement. Represents the "This" in "I Did This". Note that Objects which are provided as a value for this field should include an "objectType" field. If not specified, the Object is assumed to be an Activity.
result	Object	Result Object, further details representing a measured outcome relevant to the specified Verb.
context	Object	Context that gives the Statement more meaning. Examples: a team the Actor is working with, altitude at which a scenario was attempted in a flight simulator.
timestamp	Date/Time	Timestamp (Formatted according to ISO 8601) of when the events described within this Statement occurred. If not provided, LRS should set this to the value of "stored" time.
stored	Date/Time	Timestamp (Formatted according to ISO 8601) of when this Statement was recorded. Set by LRS.
authority	Object	Agent who is asserting this Statement is true. Verified by the LRS based on authentication, and set by LRS if left blank.
version	Version	The Statement's associated xAPI version, formatted according to Semantic Versioning 1.0.0
attachments	Array of attachment Objects	Headers for attachments to the Stateme

Aside from (potential or required) assignments of properties during LRS processing ("id", "authority", "stored", "timestamp", "version") Statements are immutable. Note that the content of Activities that are referenced in Statements is not considered part of the Statement itself. So while the Statement is immutable, the Activities referenced by that Statement are not. This means a deep serialization of a Statement into JSON will change if the referenced Activities change (see the Statement API's "format" parameter for details).

An example of the simplest possible Statement using all properties that MUST or SHOULD be used:

```
{
    "id": "12345678-1234-5678-1234-567812345678",
    "actor":{
        "mbox":"mailto:xapi@adlnet.gov"
    },
    "verb":{
        "id":"http://adlnet.gov/expapi/verbs/created",
        "display":{
            "en-US":"created"
        }
    },
    "object":{
        "id":"http://example.adlnet.gov/xapi/example/activity"
    }
}
```

See Appendix D: Example Statements for more examples.

ID 4.1.1

Description A UUID (see RFC 4122 for requirements, and the UUID must be in a standard string form).

Requirements
- Ids MUST be generated by the LRS if a Statement is received without an id.
- Ids SHOULD be generated by the Activity Provider.

Actor 4.1.2

Description A mandatory Agent or Group Object.

When the Actor ObjectType is Agent 4.1.2.1

Description An Agent (an individual) is a persona or system.

Details
- An Agent MUST be identified by one (1) of the four types of Inverse Functional Identifiers (see 4.1.2.3 Inverse Functional Identifier).
- An Agent MUST NOT include more than one (1) Inverse Functional Identifier.
- An Agent SHOULD NOT use inverse functional identifiers that are also used as a Group identifier.

The table below lists the properties of Agent objects.

Property	Type	Description	Required
objectType	String	"Agent". This property is optional except when the Agent is used as a Statement's Object.	no
name	String	Full name of the Agent.	no
see 4.1.2.3 Inverse Functional Identifier		An Inverse Functional Identifier unique to the Agent.	yes

When the Actor ObjectType is Group 4.1.2.2

Description

A Group represents a collection of Agents and can be used in most of the same situations an Agent can be used. There are two types of Groups, anonymous and identified.

Requirements

- A system consuming Statements MUST consider each anonymous Group distinct even if it has an identical set of members.
- A system consuming Statements MUST NOT assume that Agents in the 'member' property comprise an exact list of Agents in a given anonymous or identified Group.

For Anonymous Groups

- An anonymous Group MUST include a 'member' property listing constituent Agents.
- An anonymous Group MUST NOT contain Group Objects in the 'member' property.
- An anonymous Group MUST NOT include any Inverse Functional Identifiers.

For Identified Groups

- An identified Group MUST include exactly one (1) Inverse Functional Identifier.
- An identified Group MUST NOT contain Group Objects in the 'member' property.
- An identified Group SHOULD NOT use Inverse Functional Identifiers that are also used as Agent identifiers.
- An identified Group MAY include a 'member' property listing constituent Agents.

Details

An Anonymous Group is used describe a cluster of people where there is no ready identifier for this cluster, e.g. an ad hoc team.

The table below lists all properties of an Anonymous Group.

Property	Type	Description	Required
objectType	String	"Group".	yes
name	String	Name of the group.	no
member	Array of Agent Objects	The members of this Group.	yes

An Identified Group is used to uniquely identify a cluster of Agents.

The table below lists all properties of an Identified Group.

Property	Type	Description	Required
objectType	String	"Group".	yes
name	String	Name of the group.	no
member	Array of Agent Objects	The members of this Group.	no
see 4.1.2.3 Inverse Functional Identifier		An inverse functional identifier unique to the Group.	yes

Inverse Functional Identifier 4.1.2.3

Description An "Inverse Functional Identifier" is a value of an Agent or Identified Group that is guaranteed to only ever refer to that Agent or Identified Group.

Rationale Learning experiences become meaningless if they cannot be attributed to identifiable individuals and/or groups. In an xAPI Statement this is accomplished with a set of Inverse Functional Identifiers loosely inspired on the widely accepted FOAF principle (see: Friend Of A Friend).

Details The table below lists all possible Inverse Functional Identifier properties:

Property	Type	Description
mbox	mailto IRI	The required format is "mailto:email address". Only email addresses that have only ever been and will ever be assigned to this Agent, but no others, should be used for this property and mbox_sha1sum.
mbox_sha1sum	String	The SHA1 hash of a mailto IRI (i.e. the value of an mbox property). An LRS MAY include Agents with a matching hash when a request is based on an mbox.
openID	URI	An openID that uniquely identifies the Agent.
account	Object	A user account on an existing system e.g. an LMS or intranet.

Account Object 4.1.2.4

Description A user account on an existing system, such as a private system (LMS or intranet) or a public system (social networking site).

Requirements

- If the system that provides the account Object uses OpenID, the Activity Provider SHOULD use the OpenID property instead of an account Object.
- If the Activity Provider is concerned about revealing personally identifiable information about an Agent or Group, it SHOULD use an opaque account name (for example an account number) to identify all statements about a person while maintaining anonymity
  .

Details The table below lists all properties of Account Objects.

Property	Type	Description
homePage	IRL	The canonical home page for the system the account is on. This is based on FOAF's accountServiceHomePage.
name	String	The unique id or name used to log in to this account. This is based on FOAF's accountName.

Example This example shows an Agent identified by an opaque account:

```
{
    "objectType": "Agent",
    "account": {
        "homePage": "http://www.example.com",
        "name": "1625378"
    }
}
```

Verb 4.1.3

Description The Verb defines the action between Actor and Activity.

Rationale The Verb in an xAPI Statement describes the action performed during the learning experience. The xAPI does not specify any particular Verbs. (With one exception, namely the reserved Verb 'http://adlnet.gov/expapi/verbs/voided'). Instead, it defines how to create Verbs so that communities of practice can establish Verbs meaningful to their members and make them available for use by anyone. A predefined list of Verbs would be limited by definition and might not be able to effectively capture all possible future learning experiences.

Requirements Verbs appear in Statements as Objects consisting of an IRI and a set of display names corresponding to the multiple languages or dialects which provide human–readable meanings of the Verb.

- The display property MUST be used to illustrate the meaning which is already determined by the Verb IRI.
- A system reading a Statement MUST use the Verb IRI to infer meaning.
- The display property MUST NOT be used to alter the meaning of a Verb.
- A system reading a Statement MUST NOT use the display property to infer any meaning from the Statement.
- A system reading a Statement MUST NOT use the display property for any purpose other than display to a human. Using the display property for aggregation or categorization of Statements is an example of violating this requirement.
- The display property SHOULD be used by all Statements.
- The IRI contained in the id SHOULD be human-readable and imply the Verb meaning.

Details The table below lists all properties of the Verb Object.

Property	Type	Description
Id	IRI	Corresponds to a Verb definition. Each Verb definition corresponds to the meaning of a Verb, not the word. The IRI should be human-readable and imply the Verb meaning.
display	Language Map	The human readable representation of the Verb in one or more languages. This does not have any impact on the meaning of the Statement, but serves to give a human-readable display of the meaning already determined by the chosen Verb.

Example

```
{
    "verb" : {
        "id":"http://www.adlnet.gov/XAPIprofile/ran(travelled_a_distance)",
        "display":{
            "en-US":"ran",
            "es" : "corrió"
        }
    }
}
```

The Verb in the example above is included for illustrative purposes only. This is not intended to imply that a Verb with this meaning has been defined with this id. This applies to all example Verbs given in this specification document, with the exception of the reserved Verb 'http://adlnet.gov/expapi/verbs/voided'.

Use in Language and Semantics of Verbs 4.1.3.1

Details

Semantics

The IRI represented by the Verb id identifies the particular semantics of a word, not the word itself.

For example, the English word "fired" could mean different things depending on context, such as "fired a weapon", "fired a kiln", or "fired an employee". In this case, an IRI MUST identify one of these specific meanings, not the word "fired".

The display property has some flexibility in tense. While the Verb IRIs are expected to remain in the past tense, if conjugating verbs to another tense (using the same Verb) within the Activity makes sense, it is allowed.

Language

A Verb in the Experience API is an IRI, and denotes a specific meaning not tied to any particular language.

For example, a particular Verb IRI such as `http://example.org/firearms#fire` might denote the action of firing a gun, or the Verb IRI `http://example.com/خواندن/فعل` might denote the action of reading a book.

Use in Communities of Practice 4.1.3.2

Requirements for Communities of Practice

- Anyone establishing a new Verb MUST own the IRI, or MUST have permission from the owner to use it to denote an xAPI verb.
- Anyone establishing a new Verb SHOULD make a human-readable description of the intended usage of the verb accessible at the IRI.

Requirements for Activity Providers

- Activity Providers SHOULD use a corresponding existing Verb whenever possible.
- Activity Providers MAY create and use a Verb if no suitable Verb exists.

Details

Communities of practice will, at some point in time, need to establish new Verbs to meet the needs of their constituency.

It is expected that xAPI generates profiles, lists, and repositories that become centered on Verb vocabularies. ADL is one such organization that is creating a companion document containing Verbs for xAPI. In fulfillment of the requirements above, a collection of IRIs of recommended Verbs exists. There are times when Activity Providers may wish to use a different verb for the same meaning.

Object 4.1.4

Definition The Object of a Statement can be an Activity, Agent/Group, Sub-Statement, or Statement Reference. It is the "this" part of the Statement, i.e., "I did this".

Some examples:

- The object is an Activity: "Jeff wrote an essay about hiking."
- The Object is an Agent: "Nellie interviewed Jeff."
- The Object is Sub-Statement or Statement Reference (different implementations, but similar when human-read): "Nellie commented on 'Jeff wrote an essay about hiking. ' "

Details Objects which are provided as a value for this field SHOULD include an "objectType" field. If not specified, the objectType is assumed to be "Activity". Other valid values are: Agent, Group, Sub-Statement or StatementRef. The properties of an Object change according to the objectType.

When the ObjectType is Activity 4.1.4.1

A Statement may represent an Activity as the Object of the Statement.

The following table lists the Object properties in this case.

Property	Type	Description
objectType	String	MUST be "Activity" when present. Optional in all cases.
id	IRI	An identifier for a single unique Activity. Required.
definition	Object	Optional Metadata, See below

Activity Definition

Details The table below lists the properties of the Activity Definition Object:

Property	Type	Use	Description
name	Language Map	Recommended	The human readable/visual name of the Activity.
description	Language Map	Recommended	A description of the Activity.
type	IRL	Recommended	The type of Activity.
moreInfo	IL	Optional	SHOULD resolve to a document human-readable information about the Activity, which MAY include a way to 'launch' the Activity.
Interaction properties, See: Interaction Activities			
extensions	Object	Optional	A map of other properties as needed (see: Extensions)

Note IRI fragments (sometimes called relative IRLs) are not valid IRIs. As with Verbs, it is recommended that Activity Providers look for and use established, widely adopted, Activity types.

An LRS should update its internal representation of an activity's definition upon receiving a Statement with the same Activity id, but with a different definition of the Activity from the one stored, but only if it considers the Activity Provider to have the authority to do so.

Activity id

Requirements for Activity ids

- An Activity id MUST be unique.
- An Activity id MUST always reference the same Activity.
- An Activity id SHOULD use a domain that the creator is authorized to use for this purpose.
- An Activity id SHOULD be created according to a scheme that makes sure all Activity ids within that domain remain unique.
- An Activity id MAY point to metadata or the IRL for the Activity.

Requirements for the LRS

- An LRS MUST ignore any information which indicates two authors or organizations may have used the same Activity id.
- An LRS MUST NOT treat references to the same id as references to different Activities.
- Upon receiving a Statement with an Activity Definition that differs from the one stored, an LRS SHOULD decide whether it considers the Activity Provider to have the authority to change the definition and SHOULD update the stored Activity Definition accordingly if that decision is positive.
- An LRS MAY accept small corrections to the Activity's definition. For example, it would be okay for an LRS to accept spelling fixes, but it may not accept changes to correct responses.

Requirements for the Activity Provider

- An Activity Provider MUST ensure that Activity ids are not re-used across multiple Activities.
- An Activity Provider MUST only generate states or Statements against a certain Activity id that are compatible and consistent with states or Statements previously stored against the same id.
- An Activity Provider MUST NOT allow new versions (i.e., revisions or other platforms) of the Activity to break compatibility.

Details

If it were possible to use the same id for two different Activities, the validity of Statements about these Activities could be questioned. This means an LRS may never treat (references to) the same Activity id as belonging to two different Activities, even if it thinks this was intended. Namely, when a conflict with another system occurs, it's not possible to determine the intentions.

Activity Metadata

Requirements

- If an Activity IRI is an IRL, an LRS SHOULD attempt to GET that IRL, and include in HTTP headers: "Accept: application/json, /". This SHOULD be done as soon as practical after the LRS first encounters the Activity id.
- Upon loading JSON which is a valid Activity Definition from an IRL used as an Activity id, an LRS SHOULD incorporate the loaded definition into its internal definition for that Activity, while preserving names or definitions not included in the loaded definition.
- An Activity with an IRL identifier MAY host metadata using the Activity Definition JSON format which is used in Statements, with a Content-Type of "application/json"
- Upon loading any document from which the LRS can parse an Activity Definition from an IRL used as an Activity id, an LRS MAY consider this definition when determining its internal representation of that Activity's definition.

Interaction Activities

Rationale

Traditional e-learning has included structures for interactions or assessments. As a way to allow these practices and structures to extend Experience API's utility, this specification includes built-in definitions for interactions, which borrows from the SCORM 2004 4th Edition Data Model. These definitions are intended to provide a simple and familiar utility for recording interaction data. These definitions are simple to use, and consequently limited. It is expected that communities of practice requiring richer interactions definitions will do so through the use of extensions to an Activity's type and definition.

Requirements

- Interaction Activities MUST have a valid interactionType.
- Interaction Activities SHOULD have the Activity type "http://adlnet.gov/expapi/activities/cmi.interaction".
- An LRS, upon consuming a valid interactionType, MAY validate the remaining properties as specified in the table below and MAY return HTTP 400 "Bad Request" if the remaining properties are not valid for the Interaction Activity.

Details

The table below lists the properties for Interaction Activities.

Property	Type	Description
interactionType	String	As in "cmi.interactions.n.type" as defined in the SCORM 2004 4th Edition Run-Time Environment.
correctResponsesPattern	An array of strings	Corresponds to "cmi.interactions.n.correct_responses.n.pattern" as defined in the SCORM 2004 4th Edition Run-Time Environment, where the final *n* is the index of the array.
choices | scale | source | target | steps	Array of interaction components	Specific to the given interactionType (see below).

Interaction Components

Requirements

- Within an array of all interaction components, all values MUST be distinct.
- An interaction component's id value SHOULD not have whitespace.

Details

Interaction components are defined as follows:

Property	Type	Description
id	String	A value such a used in practice for "cmi.interactions.n.id" as defined in the SCORM 2004 4th Edition Run-Time Environment
description	Language Map	A description of the interaction component (for example, the text for a given choice in a multiple-choice interaction).

The following table shows the supported lists of CMI interaction components for an interaction activity with the given interactionType.

interactionType	supported component list(s)
choice, sequencing	choices
Likert	scale
Matching	source, target
Performance	steps
true-false, fill-in, numeric, other	[no component lists defined]

See Appendix C for examples of Activity definitions for each of the cmi.interaction types.

When the "Object" is an Agent or a Group 4.1.4.2

Requirements

- Statements that specify an Agent or Group as an Object MUST specify an 'objectType' property.

See Section 4.1.2 Actor for details regarding Agents.

When the "Object" is a Statement 4.1.4.3

Rationale There are two possibilities for using a Statement as an Object. First, an Object can take on the form of a Statement that already exists by using a Statement Reference. A common use case for Statement References is grading or commenting on an experience that could be tracked as an independent event. The special case of voiding a Statement would also use a Statement Reference. Second, an Object can be brand new Statement by using a Sub-Statement. A common use case for Sub-Statements would be any experience that would be misleading as its own Statement. Each type is defined below.

Statement References

Description A Statement Reference is a pointer to another pre-existing Statement.

Requirements

- A Statement Reference MUST specify an "objectType" property with the value "StatementRef".
- A Statement Reference MUST set the "id" property to the UUID of a Statement.

Details The table below lists all properties of a Statement Reference Object:

Property	Type	Description
objectType	String	In this case, MUST be "StatementRef".
id	UUID	The UUID of a Statement.

Example Assuming that some Statement has already been stored with the ID 8f87ccde-bb56-4c2e-ab83-44982ef22df0, the following example shows how a comment could be issued on the original Statement, using a new Statement:

```
{
    "actor" : {
        "objectType": "Agent",
        "mbox":"mailto:test@example.com"
    },
    "verb" : {
        "id":"http://example.com/commented",
        "display": {
            "en-US":"commented"
        }
    },
    "object" : {
        "objectType":"StatementRef",
        "id":"8f87ccde-bb56-4c2e-ab83-44982ef22df0"
    },
    "result" : {
        "response" : "Wow, nice work!"
    }
}
```

Sub-Statements

Description A Sub-Statement is a new Statement included as part of a parent Statement.

Requirements

- A Sub-Statement MUST specify an "objectType" property with the value "SubStatement".
- A Sub-Statement MUST be validated as a Statement in addition to other Sub-Statement requirements.
- A Sub-Statement MUST NOT have the "id", "stored", "version" or "authority" properties.
- A Sub-Statement MUST NOT contain a Sub-Statement of their own, i.e., cannot be nested.

Example One interesting use of sub-statements is in creating Statements of intention. For example, using Sub-Statements we can create statements of the form `"<I> <planned> (<I> <did> <this>)"` to indicate that we've planned to take some action. The concrete example that follows logically states that "I planned to visit 'Some Awesome Website'".

```
{
    "actor": {
        "objectType": "Agent",
        "mbox":"mailto:test@example.com"
    },
    "verb" : {
        "id":"http://example.com/planned",
        "display":{
            "en-US":"planned"
        }
    },
    "object": {
        "objectType": "SubStatement",
        "actor" : {
            "objectType": "Agent",
            "mbox":"mailto:test@example.com"
        },
        "verb" : {
            "id":"http://example.com/visited",
            "display":{
                "en-US":"will visit"
            }
        },
        "object": {
            "id":"http://example.com/website",
            "definition": {
                "name" : {
                    "en-US":"Some Awesome Website"
                }
            }
        }
    }

}
```

Result 4.1.5

Description An optional field that represents a measured outcome related to the Statement in which it is included.

Details The following table contains the properties of the Results Object.

Property	Type	Description
score	Object	The score of the agent in relation to the success or quality of the experience. See: Score
success	Boolean	Indicates whether or not the attempt on the Activity was successful.
completion	Boolean	Indicates whether or not the Activity was completed.
response	String	A response appropriately formatted for the given Activity.
duration	Formatted according to ISO 8601 with a precision of 0.01 seconds	Period of time over which the Statement occurred.
extensions	Object	A map of other properties as needed. See: Extensions

Score 4.1.5.1

Description An optional numeric field that represents the outcome of a graded Activity achieved by an Agent.

Requirements

- The Score Object SHOULD include 'scaled' if a logical percent based score is known.
- The Score Object SHOULD NOT be used for scores relating to progress or completion. Consider using an extension from an extension profile instead.

Details The table below defines the Score Object.

Property	Type	Description
scaled	Decimal number between -1 and 1, inclusive	Cf. 'cmi.score.scaled' in SCORM 2004 4th Edition
raw	Decimal number between min and max (if present, otherwise unrestricted), inclusive	Cf. 'cmi.score.raw'
min	Decimal number less than max (if present)	Cf. 'cmi.score.min'
max	Decimal number greater than min (if present)	Cf. 'cmi.score.max'

Context 4.1.6

Description An optional field that provides a place to add contextual information to a Statement. All properties are optional.

Rationale The "context" field provides a place to add some contextual information to a Statement. It can store information such as the instructor for an experience, if this experience happened as part of a team Activity, or how an experience fits into some broader activity.

Requirements

- The *revision* property MUST NOT be used if the Statement's Object is an Agent or Group.
- The *platform* property MUST NOT be used if the Statement's Object is an Agent or Group.
- The *language* property MUST NOT be used if not applicable or unknown.
- The *revision* property SHOULD be used to track fixes of minor issues (like a spelling error).
- The *revision* property SHOULD NOT be used if there is a major change in learning objectives, pedagogy, or assets of an Activity. (Use a new Activity id instead).

Details

Property	Type	Description
registration	UUID	The registration that the Statement is associated with.
instructor	Agent (may be a group)	Instructor that the Statement relates to, if not included as the Actor of the statement.
team	Group	Team that this Statement relates to, if not included as the Actor of the statement.
contextActivities	contextActivities Object	A map of the types of learning activity context that this Statement is related to. Valid context types are: "parent", "grouping", "category" and "other".
revision	String	Revision of the learning activity associated with this Statement. Format is free.
platform	String	Platform used in the experience of this learning activity.
language	String (as defined in RFC 5646)	Code representing the language in which the experience being recorded in this Statement (mainly) occurred in, if applicable and known.
statement	[Statement Reference] (#stmtref)	Another Statement which should be considered as context for this Statement.
extensions	Object	A map of any other domain-specific context relevant to this Statement. For example, in a flight simulator altitude, airspeed, wind, attitude, GPS coordinates might all be relevant (See Extensions)

Note Revision has no behavioral implications within the scope of xAPI. It is simply stored, so that it is available for reporting tools.

Registration Property 4.1.6.1

Description An instance of a learner undertaking a particular learning activity.

Details When an LRS is an integral part of an LMS, the LMS likely supports the concept of registration. The Experience API applies the concept of registration more broadly. A registration could be considered to be an attempt, a session, or could span multiple Activities. There is no expectation that completing an Activity ends a registration. Nor is a registration necessarily confined to a single Agent.

ContextActivities Property 4.1.6.2

Description A map of the types of learning activity context that this Statement is related to.

Rationale Many Statements do not just involve one Object Activity that is the focus, but relate to other contextually relevant Activities. "Context activities" allow for these related Activities to be represented in a structured manner.

Requirements

- Every key in the contextActivities Object MUST be one of parent, grouping, category, or other.
- Every value in the contextActivities Object MUST be either a single Activity object or an array of Activity objects.
- The LRS MUST return every value in the contextActivities Object as an array, even if it arrived as a single Activity object;
- The LRS MUST return single Activity Objects as an array of length one containing the same Activity.
- The Client SHOULD ensure that every value in the contextActivitiesOobject is an array of Activity objects instead of a single Activity object.

Details There are four valid context types. All, any or none of these MAY be used in a given Statement:

1. **Parent**: an Activity with a direct relation to the activity which is the Object of the Statement. In almost all cases there is only one sensible parent or none, not multiple. For example: a Statement about a quiz question would have the quiz as its parent Activity.

2. **Grouping**: an Activity with an indirect relation to the activity which is the Object of the Statement. For example: a course that is part of a qualification. The course has several classes. The course relates to a class as the parent, the qualification relates to the class as the grouping.

3. **Category**: an Activity used to categorize the Statement. "Tags" would be a synonym. Category SHOULD be used to indicate a "profile" of xAPI behaviors, as well as other categorizations. For example: Anna attempts a biology exam, and the Statement is tracked using the CMI-5 profile. The Statement's Activity refers to the exam, and the category is the CMI-5 profile.

4. **Other**: a context Activity that doesn't fit one of the other fields. For example: Anna studies a textbook for a biology exam. The Statement's Activity refers to the textbook, and the exam is a context Activity of type "other".

Single Activity Objects are allowed as values so that 0.95 Statements will be compatible with 1.0.0.

Note The values in this section are not for expressing all the relationships the Statement Object has. Instead, they are for expressing relationships appropriate for the specific Statement (though the nature of the Object will often be important in determining that). For instance, it is appropriate in a Statement about a test to include the course the test is part of as parent, but not to include every possible degree program the course could be part of in the grouping value.

Example Consider the following hierarchical structure: "Questions 1 to 6" are part of "Test 1" which in turn belongs to the course "Algebra 1". The six questions are registered as part of the test by declaring "Test 1" as their parent. Also they are grouped with other Statements about "Algebra 1" to fully mirror the hierarchy. This is particularly useful when the Object of the Statement is an Agent, not an Activity. "Andrew mentored Ben with context Algebra 1".

```
{
    "parent" : [{
    "id" : "http://example.adlnet.gov/xapi/example/test 1"
    }],
    "grouping" : [{
    "id" : "http://example.adlnet.gov/xapi/example/Algebra1"
    }]
}
```

Timestamp 4.1.7

Description

The time at which a Statement was generated.

Requirements

- A timestamp MUST be formatted according to ISO 8601.
- A timestamp SHOULD include the time zone.
- A timestamp SHOULD be the current or a past time when it is outside of a Sub-Statement.
- A timestamp MAY be truncated or rounded to a precision of at least 3 decimal digits for seconds (millisecond precision MUST be preserved).
- A timestamp MAY be a moment in the future, to denote a deadline for planned learning, provided it is included inside a Sub-Statement.

Details

A timestamp in a Statement related to learning that occurs outside of the system can differ from Stored (the system time of the event). Namely, there can be delays between the occurrence of the experience and the reception of the corresponding Statement by the LRS. Another cause is when Statements are propagated to other systems.

Stored 4.1.8

Description

The time at which a Statement is stored by the LRS.

The stored property is the literal time the Statement was stored. Use Timestamp to track a time at which the Statement was generated.

Requirements

- The stored property MUST be formatted according to ISO 8601.
- The stored property SHOULD include the time zone.
- The stored property SHOULD be the current or a past time.
- The stored property MAY be truncated or rounded to a precision of at least 3 decimal digits for seconds (millisecond precision MUST be preserved).

Authority 4.1.9

Description The authority property provides information about whom or what has asserted that this Statement is true.

Requirements

- Authority MUST be an Agent, except in 3-legged OAuth, where it must be a Group with two Agents. The two Agents represent an application and user together.
- The LRS MUST include the user as an Agent as the entire authority if the user connects directly (using HTTP Basic Authentication) or is included as part of a Group.
- The LRS MUST ensure that all Statements stored have an authority.
- The LRS SHOULD overwrite the authority on all stored received Statements, based on the credentials used to send those Statements.
- The LRS MAY leave the submitted authority unchanged but SHOULD do so only where a strong trust relationship has been established, and with extreme caution.
- The LRS MAY identify the user with any of the legal identifying properties if a user connects directly (using HTTP Basic Authentication) or a part of a 3-legged OAuth.

Details The asserting authority represents the authenticating user of some system or application.

OAuth Credentials as Authority

Description This is a workflow for use of OAuth. 2-legged and 3-legged OAuth are both supported.

Requirements

- The authority MUST contain an Agent Object that represents the OAuth consumer, either by itself, or as part of a group in the case of 3-legged OAuth.
- The Agent representing the OAuth consumer MUST be identified by account.
- The Agent representing the OAuth consumer MUST use the consumer key as the “account name” field.
- If the Agent representing the OAuth consumer is a registered application, the token request endpoint MUST be used as the account homePage.
- If the Agent representing the OAuth consumer is not a registered application, the temporary credentials endpoint MUST be used as the account homePage.
- An LRS MUST NOT trust the application portion of the authority in the event the account name is from the same source as the unregistered application. (Multiple unregistered applications could choose the same consumer key. As a result, there is no consistent way to verify this combination of temporary credentials and the account name.)
- Each unregistered consumer SHOULD use a unique consumer key.

Details This workflow assumes a Statement is stored using a validated OAuth connection and the LRS creates or modifies the authority property of the Statement.

In a 3-legged OAuth workflow, authentication involves both and OAuth consumer and a user of the OAuth service provider. For instance, requests made by an authorized Twitter plug-in on their Facebook account will include credentials that are specific not only to Twitter as a Client application, or them as a user, but the uniqe combination of both.

Example

The pairing of an OAuth consumer and a user.

```
"authority": {
    "objectType" : "Group",
    "member": [
        {
            "account": {
                "homePage":"http://example.com/xAPI/OAuth/Token",
                "name":"oauth_consumer_x75db"
            }
        },
        {
            "mbox":"mailto:bob@example.com"
        }
    ]
}
```

Version 4.1.10

Description

Version information in Statements helps systems that process data from an LRS get their bearings. Since the Statement data model is guaranteed consistent through all 1.0.x versions, in order to support data flow among such LRSs the LRS is given some flexibility on Statement versions that are accepted.

Requirements

- Version MUST be formatted as laid out for the API version header in API Versioning.

LRS Requirements

- An LRS MUST accept all Statements where their version starts with "1.0." if they otherwise validate.
- An LRS MUST reject all Statements with a version specified that does not start with "1.0."
- Statements returned by an LRS MUST retain the version they are accepted with. If they lack a version, the version MUST be set to 1.0.0

Client Requirements

- If Clients set the Statement version, they MUST set it to 1.0.0
- Clients SHOULD NOT set the Statement version.

Attachments 4.1.11

Description A digital artifact providing evidence of a learning experience.

Rationale In some cases an attachment may logically be an important part of a learning record. Think of a simulated communication with ATC, an essay, a video, etc. Another example of such an attachment is (the image of) a certificate that was granted as a result of an experience. It is useful to have a way to store these attachments in and retrieve them from an LRS.

Requirements A Statement batch, Statement results, or a single Statement that includes attachments:

for Attachment Statement Batches

- MUST be of type "application/json" and include a fileUrl for every attachment EXCEPT for Statement results when the attachment filter is false or
- MUST conform to the definition of multipart/mixed in RFC 1341 and:
  - The first part of the multipart document MUST contain the Statements themselves, with type "application/json".
  - Each additional part contains the raw data for an attachment and forms a logical part of the Statement. This capability will be available when issuing PUT or POST against the Statement resource.
  - MUST include an X-Experience-API-Hash field in each part's header after the first (Statements) part.
  - This field MUST be set to match the "sha2" property of the attachment declaration corresponding to the attachment included in this part.
  - MUST include a Content-Transfer-Encoding field with a value of "binary" in each part's header after the first (Statements) part.
  - SHOULD only include one copy of an attachment's data when the same attachment is used in multiple Statements that are sent in one batch.
  - SHOULD include a Content-type field in each part's header, for the first part this MUST be "application/json".

Requirements *for the LRS*

- An LRS MUST include attachments in the Transmission Format described above when requested by the Client (see Section 7.2 "Statement API").
- An LRS MUST NOT pull Statements from another LRS without requesting attachments.
- An LRS MUST NOT push Statements into another LRS without including attachment data received, if any, for those attachments.
- When receiving a PUT or POST with a document type of "application/json"
  - An LRS MUST accept batches of Statements which contain either no attachment Objects, or only attachment Objects with a populated fileUrl;
- Otherwise:
  - An LRS MUST accept batches of Statements via the Statements resource PUT or POST that contain attachments in the Transmission Format described above.
  - An LRS MUST reject batches of Statements having attachments that neither contain a fileUrl nor match a received attachment part based on their hash.
  - An LRS SHOULD assume a Content-Transfer-Encoding of binary for attachment parts.
- An LRS MAY reject (batches of) Statements that are larger than the LRS is configured to allow.

Note There is no requirement that Statement batches using the mime/multipart format contain attachments.

Requirements *for the Client*

- The Client MAY send Statements with attachments as described above;
- The Client MAY send multiple Statements where some or all have attachments if using "POST".
- The Client MAY send batches of type "application/json" where every attachment Object has a fileUrl, ignoring all requirements based on the "multipart/mixed" format.

Details

The table below lists all properties of the Attachment Object.

Property	Type	Description	Required
usageType	IRI	Identifies the usage of this attachment. For example: one expected use case for attachments is to include a "completion certificate". A type IRI corresponding to this usage should be coined, and used with completion certificate attachments.	yes
display	Language Map	Display name (title) of this attachment.	yes
description	Language Map	A description of the attachment.	no
contentType	Internet Media Type	The content type of the attachment.	yes
length	integer	The length of the attachment data in octets.	yes
sha2	base64	The SHA-2 hash of the attachment data. A minimum key size of 256 bits is recommended.	yes
fileUrl	IRL	An IRL at which the attachment data may be retrieved, or from which it used to be retrievable.	no

Procedure for the exchange of attachments

1. A Statement including an attachment is construed according to the Transmission Format described below.
2. The Statement is sent to the receiving system using a content-Type of "multipart/mixed". The attachments are placed at the end of such transmissions.
3. The receiving system decides whether to accept or reject the Statement based on the \ information in the first part.
4. If it accepts the attachment, it can match the raw data of an attachment with the attachment header in a Statement by comparing the SHA-2 of the raw data to the SHA-2 declared in the header. It MUST not do so in any other way.

Example

Below is an example of a very simple Statement with an attachment. Please note the following:

- The boundary in the sample was chosen to demonstrate valid character classes;
- The selected boundary does not appear in any of the parts;
- For readability the sample attachment is text/plain. Even if it had been a 'binary' type like 'image/jpeg' no encoding would be done, the raw octets would be included;
- Per RFC 1341, the boundary is followed by -- followed by the boundary string declared in the header.

Don't forget the `<CRLF>` when building or parsing these messages.

Headers:

```
Content-Type: multipart/mixed; boundary=abcABC0123'()+_,-./:=?

X-Experience-API-Version:1.0.0
```

Content:

```
--abcABC0123'()+_,-./:=?
Content-Type:application/json

{
    "actor": {
        "mbox": "mailto:sample.agent@example.com",
        "name": "Sample Agent",
        "objectType": "Agent"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/answered",
        "display": {
            "en-US": "answered"
        }
    },
    "object": {
        "id": "http://www.example.com/tincan/activities/multipart",
        "objectType": "Activity",
        "definition": {
            "name": {
                "en-US": "Multi Part Activity"
            },
            "description": {
                "en-US": "Multi Part Activity Description"
            }
        }
    },
    "attachments": [
        {
            "usageType": "http://example.com/attachment-usage/test",
            "display": { "en-US": "A test attachment" },
            "description": { "en-US": "A test attachment (description)" },
            "contentType": "text/plain; charset=ascii",
            "length": 27,
            "sha2":
"495395e777cd98da653df9615d09c0fd6bb2f8d4788394cd53c56a3bfdcd848a"
        }
    ]
}
```

```
--abcABC0123'()+_,-./:=?
Content-Type:text/plain
Content-Transfer-Encoding:binary
X-Experience-API-
Hash:495395e777cd98da653df9615d09c0fd6bb2f8d4788394cd53c56a3bfdcd848a

here is a simple attachment
--abcABC0123'()+_,-./:=?--
```

Data Constraints 4.1.12

Description

All the properties used in Statements are restricted to certain types, and those types constrain the behavior of systems processing Statements. For clarity, certain key requirements are documented here, emphasizing where compliant systems have a responsibility to act in certain ways.

Requirements for the Client

The following requirements reiterate especially important requirements already included elsewhere, to emphasize, clarify, and provide implementation guidance.

- Values requiring IRIs MUST be sent with valid IRIs. Please use a library to construct them instead of string concatenation. Complete IRI validation is extremely difficult, so much of the burden for ensuring data portability is on the Client.
- Keys of language maps MUST be sent with valid RFC 5646 language tags, for similar reasons.

Requirements for the LRS

- The LRS MUST reject Statements
  - with any null values (except inside extensions).
  - with strings where numbers are required, even if those strings contain numbers.
  - with strings where booleans are required, even if those strings contain Booleans.
  - with any non-format-following key or value, including the empty string, where a. string with a particular format (such as mailto IRI, UUID, or IRI) is required.
  - where the case of a key does not match the case specified in the standard.
  - where the case of a value restricted to enumerated values does not match an enumerated value given in the standard exactly.
- The LRS MUST reject Statements containing IRL, IRI, or IRI values without a scheme.
- The LRS MUST at least validate that the sequence of token lengths for language map keys matches the RFC 5646 standard.
- The LRS MUST process and store numbers with at least the precision of IEEE 754 32-bit floating point numbers.
- The LRS MUST validate parameter values to the same standards required for values of the same types in Statements. **Note:** string parameter values are not quoted as they are in JSON.
- The LRS MAY use best-effort validation for IRL, IRI, and IRI formats to satisfy the non-format-following rejection requirement.
- The LRS MAY use best-effort validation for language map keys to satisfy the non-format-following rejection requirement.

Retrieval of Statements 4.2

Description A collection of Statements can be retrieved by performing a query on the "statements" endpoint, see Section 7.2 “Statement API” for details.

Details The following table shows the data structure for the results of queries on the Statement API.

Property	Type	Description
statements	Array of Statements	List of Statements. If the list returned has been limited (due to pagination), and there are more results, they will be located at the "statements" property within the container located at the IRL provided by the "more" element of this Statement result Object.
more	IRL	Relative IRL that may be used to fetch more results, including the full path and optionally a query string but excluding scheme, host, and port. Empty string if there are no more results to fetch. This IRL must be usable for at least 24 hours after it is returned by the LRS. In order to avoid the need to store these IRLs and associated query data, an LRS may include all necessary information within the IRL to continue the query, but should avoid generating extremely long IRLs. The consumer should not attempt to interpret any meaning from the IRL returned.

Voided 4.3

Rationale

The certainty that an LRS has an accurate and complete collection of data is guaranteed by the fact that Statements cannot be logically changed or deleted. This immutability of Statements is a key factor in enabling the distributed nature of Experience API.

However, not all Statements are perpetually valid once they have been issued. Mistakes or other factors could require that a previously made Statement is marked as invalid. This is called "voiding a Statement" and the reserved Verb "http://adlnet.gov/expapi/verbs/voided" is used for this purpose. Any Statement that voids another cannot itself be voided.

Requirements

- When issuing a Statement that voids another, the Object of that voiding statement MUST have the "objectType" field set to "StatementRef".
- When issuing a Statement that voids another, the Object of that voiding statement MUST specify the id of the statement-to-be-voided by its "id" field.
- Upon receiving a statement that voids another, the LRS SHOULD reject the entire request which includes the voiding statement with HTTP 403 'Forbidden' if the request is not from a source authorized to void Statements.
- Upon receiving a statement that voids another, the LRS SHOULD return a descriptive error if the target Statement cannot be found.
- Upon receiving a statement that voids another, the LRS MAY roll back any changes to Activity or Agent definitions which were introduced by the Statement that was just voided.
- An Activity Provider that wants to "unvoid" a previously voided Statement SHOULD issue that Statement again under a new id.
- A reporting system SHOULD NOT show voided or voiding Statements by default.

Note

See "Statement References" in section 4.1.4.3 When the "Object" is a Statement for details about making references to other Statements. To see how voided statements behave when queried, see StatementRef in 7.2 Statement API.

Example

This example Statement voids a previous Statement which it identifies with the statement id "e05aa883-acaf-40ad-bf54-02c8ce485fb0".

```
{
    "actor" : {
        "objectType": "Agent",
        "name" : "Example Admin",
        "mbox" : "mailto:admin@example.adlnet.gov"
    },
    "verb" : {
        "id":"http://adlnet.gov/expapi/verbs/voided",
        "display":{
            "en-US":"voided"
        }
    },
    "object" : {
        "objectType":"StatementRef",
        "id" : "e05aa883-acaf-40ad-bf54-02c8ce485fb0"
    }
}
```

Signed Statements 4.4

Description

A Statement may include a digital signature to provide strong and durable evidence of the authenticity and integrity of the Statement.

Rationale

Some Statements will have regulatory or legal significance, or otherwise require strong and durable evidence of their authenticity and integrity. It may be necessary to verify these Statements without trusting the system they were first recorded in, or perhaps without access to that system. Digital signatures will enable a third-party system to validate such Statements.

Requirements

- A Signed Statement MUST include a JSON web signature (JWS) as defined here: http://tools.ietf.org/html/draft-ietf-jose-json-web-signature, as an attachment with a usageType of "http://adlnet.gov/expapi/attachments/signature" and a contentType of "application/octet-stream".
- The JWS signature MUST have a payload of a valid JSON serialization of the Statement generated before the signature was added.
- The JWS signature MUST use an algorithm of "RS256","RS384", or "RS512".
- The JWS signature SHOULD have been created based on the private key associated with an X.509 certificate.
- If X.509 was used to sign, the JWS header SHOULD include the "x5c" property containing the associated certificate chain.
- The LRS MUST reject requests to store Statements that contain malformed signatures, with `HTTP 400`, and SHOULD include a message describing the problem in the response. In order to verify signatures are well formed, the LRS MUST do the following:
  - Decode the JWS signature, and load the signed serialization of the Statement from the JWS signature payload.
  - Validate that the "original" Statement is logically equivalent to the received Statement.
    - When making this equivalence check, differences which could have been caused by allowed or required LRS processing of "id", "authority", "stored", "timestamp", or "version" MUST be ignored.
  - If the JWS header includes an X.509 certificate, validate the signature against that certificate as defined in JWS.
- Clients MUST NOT assume a signature is valid simply because an LRS has accepted it.

Details

Signed Statements include a JSON web signature (JWS) as an attachment. This allows the original serialization of the Statement to be included along with the signature. For interoperability, the "RSA + SHA" series of JWS algorithms have been selected, and for discoverability of the signer X.509 certificates SHOULD be used.

See Appendix F: Example Signed Statement for an example.

Note

The step of validating against the included X.509 certificate is intended as a way to catch mistakes in the signature, not as a security measure. Clients MUST NOT assume a signature is valid simply because an LRS has accepted it. The steps to authenticate a Signed Statement will vary based on the degree of certainty required and are outside the scope of this specification.

5.0 MISCELLANEOUS TYPES

Document 5.1

Description

The Experience API provides a facility for Activity Providers to save arbitrary data in the form of documents, which may be related to an Activity, Agent, or combination of both.

Details

Property	Type	Description
id	String	Set by AP, unique within state scope (learner, activity).
updated	Timestamp	When the document was most recently modified.
contents	Arbitrary binary data	The contents of the document.

Note

In the REST binding, State is a document, not an Object. The id is stored in the IRL, updated is HTTP header information, and contents is the HTTP document itself.

Language Map 5.2

Description

A language map is a dictionary where the key is an RFC 5646 Language Tag, and the value is a string in the language specified in the tag. This map should be populated as fully as possible based on the knowledge of the string in question in different languages.

Extensions 5.3

Description

Extensions are defined by a map. The keys of that map MUST be IRLs, and the values MAY be any JSON value or data structure. The meaning and structure of extension values under an IRL key are defined by the person who coined the IRL, who SHOULD be the owner of the IRL, or have permission from the owner. The owner of the IRL SHOULD make a human-readable description of the intended meaning of the extension supported by the IRL accessible at the IRL. A learning record store MUST NOT reject an Experience API Statement based on the values of the extensions map.

Extensions are available as part of Activity Definitions, as part of statement context, or as part of some statement result. In each case, they're intended to provide a natural way to extend those elements for some specialized use. The contents of these extensions might be something valuable to just one application, or it might be a convention used by an entire community of practice.

Extensions should logically relate to the part of the statement where they are present. Extensions in Statement context should provide context to the core experience, while those in the result should provide elements related to some outcome. For Activities, they should provide additional information that helps define an Activity within some custom application or community.

Note

A statement should not be totally defined by its extensions, and be meaningless otherwise. Experience API Statements should be capturing experiences among Actors and Objects, and SHOULD always strive to map as much information as possible into the built in elements, in order to leverage interoperability among Experience API conformant tools.

Identifier Metadata 5.4

Description

There are several types of IRI identifiers used in this specification:

- Verb
- Activity id
- Activity type
- extension key
- attachment usage type

For Activity ids, see Activity Definition.

For all other identifiers, metadata MAY be provided in the following JSON format:

Property	Type	Description
name	Language Map	The human readable/visual name
description	Language Map	description

Requirements

If metadata is provided, both name and description SHOULD be included.

- For any of the identifier IRIs above, if the IRI is an IRL that was coined for use with this specification, the owner of that IRL SHOULD make this JSON metadata available at that IRL when the IRL is requested and a Content-Type of "application/json" is requested.
- If this metadata is provided as described above, it is the canonical source of information about the identifier it describes.
- Other sources of information MAY be used to fill in missing details, such as translations, or take the place of this metadata entirely if it was not provided or cannot be loaded. This MAY include metadata in other formats stored at the IRL of an identifier, particularly if that identifier was not coined for use with this specification.

As with Verbs, we recommend that Activity Providers look for and use established, widely adopted identifiers for all types of IRI identifier other than Activity id. Where an identifier already exists, the Activity Provider:

- SHOULD use the corresponding existing identifier;
- MAY create and use their own Verbs where a suitable identifier does not already exist.

6.0 Runtime Communication

Sections 6 and 7 detail the more technical side of the Experience API, dealing with how Statements are transferred between Activity Provider and LRS. A number of libraries are under development for a range of technologies (including JavaScript) which handle this part of the specification. It therefore may not be necessary for content developers to fully understand every detail of this part of the specification.

Encoding 6.1

Requirement

- All strings must be encoded and interpreted as UTF-8.

API Versioning 6.2

Rationale

Future revisions of the specification may introduce changes such as properties added to Statements.

Systems retrieving statements may then receive responses that include statements of different versions. The version header allows for these version differences to be handled correctly, and to ascertain that no partial or mixed LRS version implementations exist.

Using Semantic Versioning will allow Clients and LRSs to reliably know whether they're compatible or not as the specification changes.

Requirements

Every request from a Client and every response from the LRS must include an HTTP header with the name “X-Experience-API-Version” and the version as the value. Starting with 1.0.0, xAPI will be versioned according to Semantic Versioning 1.0.0

Example: `X-Experience-API-Version : 1.0.0`

LRS requirements

- MUST include the "X-Experience-API-Version" header in every response.
- MUST set this header to "1.0.0".
- MUST accept requests with a version header of "1.0" as if the version header was "1.0.0".
- MUST reject requests with version header prior to "1.0.0" unless such requests are routed to a fully conformant implementation of the prior version specified in the header.
- MUST reject requests with a version header of "1.1.0" or greater.
- MUST make these rejects by responding with an HTTP 400 error including a short description of the problem.

Client requirements

- SHOULD tolerate receiving responses with a version of "1.0.0" or later;
- SHOULD tolerate receiving data structures with additional properties;
- SHOULD ignore any properties not defined in version 1.0.0 of the spec.

Converting Statements to other versions

- Systems MUST NOT convert Statements of newer versions into a prior version format, e.g., in order to handle version differences.
- Systems MAY convert Statements of older versions into a newer version only by following the methods described in Appendix E: Converting Statements to 1.0.0.

Concurrency 6.3

Description

Concurrency control makes certain that an API consumer does not PUT changes based on old data into an LRS.

Details

xAPI will use HTTP 1.1 entity tags (ETags) to implement optimistic concurrency control in the portions of the API where PUT may overwrite existing data, being:

- State API
- Agent Profile API
- Activity Profile API

The State API will permit PUT Statements without concurrency headers, since state conflicts are unlikely. The requirements below only apply to Agent Profile API and Activity Profile API.

Client requirements

An xAPI Client using either Agent Profile API or Activity Profile API:

- MUST include the If-Match header, or MUST include the If-None-Match header.

LRS requirements

The LRS that responds to a GET request:

- MUST add an ETag HTTP header to the response.
- MUST calculate the value of this header to be a hexidecimal string of the SHA-1 digest of the contents.
- MUST enclose the header in quotes.

The reason for specifying the LRS ETag format is to allow API consumers that can't read the ETag header to calculate it themselves.

The LRS that responds to a PUT request:

- MUST handle the If-Match header as described in RFC2616, HTTP 1.1 if it contains an ETag, in order to detect modifications made after the consumer last fetched the document.
- MUST handle the If-None-Match header as described in RFC2616, HTTP 1.1 if it contains "*", in order to detect when there is a resource present that the consumer is not aware of.

If the header precondition in either of the above cases fails, the LRS:

- MUST return HTTP status 412 "Precondition Failed".
- MUST NOT make a modification to the resource.

If a PUT request is received without either header for a resource that already exists, the LRS:

- MUST return HTTP status 409 "Conflict".
- MUST return a plain text body explaining that the consumer SHOULD
    - check the current state of the resource.
    - set the "If-Match" header with the current ETag to resolve the conflict.
- MUST NOT make a modification to the resource.

Security 6.4

Rationale

In order to balance the interoperability and the varying security requirements of different environments, several authentication options are defined.

Requirements

The LRS MUST support authentication using at least one of the following methods:

- OAuth 1.0 (RFC 5849), with signature methods of “HMAC-SHA1”, “RSA-SHA1”, and “PLAIN TEXT”
- HTTP Basic Authentication
- Common Access Cards (implementation details to follow in a later version)
- The LRS MUST handle making, or delegationg, decisions on the validity of Statements, and determining what operations may be performed based on the credentials used.

Details

Authentication Scenarios

The below matrix describes the possible authentication scenarios.

A **registered application** is an application that will authenticate to the LRS as an OAuth consumer that has been registered with the LRS. A **known user** is a user account on the LRS, or on a system which the LRS trusts to define users.

	Known user	**User unknown**
Application is registered	Standard workflow for OAuth.	LRS trusts application to access xAPI without additional user credentials. OAuth token steps are not invoked.
Application is not registered	The application Agent is not identified as a registered Agent and the LRS cannot make assumptions on its identity.	
No application	HTTP Basic Authentication is used instead of OAuth, since no application is involved.	
No authentication	MAY be supported by the LRS, possibly for testing purposes	

How To Handle Each Scenario 6.4.1

General
- The LRS must record the application's name and a unique consumer key (identifier).
- The LRS must provide a mechanism to complete this registration, or delegate to another system that provides such a mechanism. The means by which this registration is accomplished are not defined by OAuth or the xAPI.

Application registered + known user
- Use endpoints below to complete the standard workflow.
- If this form of authentication is used to record Statements and no authority is specified, the LRS should record the authority as a group consisting of an Agent representing the registered application, and an Agent representing the known user.

Application registered + user unknown
- LRS will honor requests that are signed using OAuth with the registered application's credentials and with an empty token and token secret.
- If this form of authentication is used to record Statements and no authority is specified, the LRS should record the authority as the Agent representing the registered application.

Application not registered + known user
- Use a blank consumer secret.
- Call "Temporary Credential" request.
- Specify "consumer_name" and other usual parameters. User will then see "consumer_name" plus a warning that the identity of the application requesting authorization cannot be verified.
- The LRS MUST record an authority that includes both that application and the authenticating user, as a group, since OAuth specifies an application.

No application + known user
- Use username/password combination that corresponds to an LRS login.
- Authority to be recorded as the Agent identified by the login, **unless…**
  - other Authority is specified **and**…
  - LRS trusts the known user to specify this Authority.

No authorization
- Requests should include headers for HTTP Basic Authentication based on a blank username and password, in order to distinguish an explicitly unauthenticated request from a request that should be given a HTTP Basic Authentication challenge.

Requirements

The LRS:

- MUST be able to be configured for complete support of the xAPI:
  - With any of the above methods.
  - In any of the workflow scenarios above.
- MAY (for security reasons):
  - Support a subset of the above methods.
  - Limit the known users or registered applications.
- SHOULD at a minimum supply Oauth with "HMAC-SHA1" and "RSA-SHA1" signatures.

OAuth Authorization Scope 6.4.2

Description These are recommendations for scopes which should enable an LRS and an application communicating using the xAPI to negotiate a level of access which accomplishes what the application needs while minimizing the potential for misuse. The limitations of each scope are in addition to any security limitations placed on the user account associated with the request.

Requirements The LRS:

- MUST accept a scope parameter as defined in OAuth 2.0;
- MUST assume a requested scope of "statements/write" and "statements/read/mine" if no scope is specified;
- MUST support the scope of "all" as a minimum;
- MAY support other scopes.

An xAPI Client:

- SHOULD request only the minimal needed scopes, to increase the chances that the request will be granted.

Details The following table lists xAPI scope values:

Scope	Permission
statements/write	write any Statement
statements/read/mine	read Statements written by "me", that is with an authority matching what the LRS would assign if writing a Statement with the current token.
statements/read	read any Statement
state	read/write state data, limited to Activities and Actors associated with the current token to the extent it is possible to determine this relationship.
define	(re)Define Activities and actors. If storing a Statement when this is not granted, ids will be saved and the LRS may save the original Statement for audit purposes, but should not update its internal representation of any Actors or Activities.
profile	read/write profile data, limited to Activities and Actors associated with the current token to the extent it is possible to determine this relationship.
all/read	unrestricted read access
all	unrestricted access

OAuth Extended Parameters

Note that the parameters "consumer_name" and "scope" are not part of OAuth 1.0, and therefore if used should be passed as query string or form parameters, not in the OAuth header.

OAuth Endpoints

Name	Endpoint	Example
Temporary Credential Request	OAuth/initiate	`http://example.com/xAPI/OAuth/initiate`
Resource Owner Authorization	OAuth/authorize	`http://example.com/xAPI/OAuth/authorize`
Token Request	OAuth/token	`http://example.com/xAPI/OAuth/token`

Example

The list of scopes determines the set of permissions that is being requested. For example, an instructor might grant "statements/read" to a reporting tool, but the LRS would still limit that tool to Statements that the instructor could read if querying the LRS with their credentials directly (such as Statements relating to their students).

7.0 DATA TRANSFER (REST)

Description

This section describes that the xAPI consists of 4 sub-APIs: Statement, State, Agent Profile, and Activity Profile. The four sub-APIs of the Experience API are handled via RESTful HTTP methods. The Statement API can be used by itself to track learning records.

LRS Requirements

The LRS MUST reject with `HTTP 400 Bad Request` status (see below) any request to any of these APIs using any parameters:

- the LRS does not recognize (Note: LRSs may recognize and act on parameters not in this specification);
- that match parameters described in this specification in all but case.

Note

In all of the example endpoints given in the specification, "http://example.com/xAPI/" is the example IRL of the LRS and everything after this represents the endpoint which MUST be used.

Error Codes 7.1

The list below offers some general guidance on HTTP error codes that could be returned from various methods in the API. An LRS MUST return the error code most appropriate to the error condition based on the list below, and SHOULD return a message in the response explaining the cause of the error.

`400 Bad Request`	Indicates an error condition caused by an invalid or missing argument. The term "invalid arguments" includes malformed JSON or invalid Object structures.
`401 Unauthorized`	Indicates that authentication is required, or in the case authentication has been posted in the request, that the given credentials have been refused.
`403 Forbidden`	Indicates that the request is unauthorized for the given credentials. Note this is different than refusing the credentials given. In this case, the credentials have been validated, but the authenticated Client is not allowed to perform the given action.
`404 Not Found`	Indicates the requested resource was not found. May be returned by any method that returns a uniquely identified resource, for instance, any State or Agent Profile or Activity Profile API call targeting a specific document, or the method to retrieve a single Statement.
`409 Conflict`	Indicates an error condition due to a conflict with the current state of a resource, in the case of State API, Agent Profile or Activity Profile API calls, or in the Statement PUT call. See Section 6.3 Concurrency for more details.
`412 Precondition Failed`	Indicates an error condition due to a failure of a precondition posted with the request, in the case of State or Agent Profile or Activity Profile API calls. See Section 6.3 Concurrency for more details.
`413 Request Entity Too Large`	Indicates that the LRS has rejected the Statement or document because its size is larger than the maximum allowed by the LRS. The LRS is free to choose any limit and MAY vary this limit on any basis, e.g., per authority, but MUST be configurable to accept Statements of any size.
`500 Internal Server Error`	Indicates a general error condition, typically an unexpected exception in processing on the server.

Statement API 7.2

Description The basic communication mechanism of the Experience API.

PUT Statements

Example endpoint: `http://example.com/xAPI/statements`

Stores Statement with the given id.

Returns: `204 No Content`

Parameter	Type	Default	Description
statementId	String		id of Statement to record

POST Statements

Example endpoint: `http://example.com/xAPI/statements`

Stores a Statement, or a set of Statements. Since the PUT method targets a specific Statement id, POST must be used rather than PUT to save multiple Statements, or to save one Statement without first generating a Statement id. An alternative for systems that generate a large amount of Statements is to provide the LRS side of the API on the AP, and have the LRS query that API for the list of updated (or new) Statements periodically. This will likely only be a realistic option for systems that provide a lot of data to the LRS.

Returns: `200 OK`, statement id(s) (UUID).

Common requirements for PUT and POST An LRS MUST NOT make any modifications to its state based on a receiving a Statement with a statementId that it already has a statement for. Whether it responds with `409 Conflict`, or `204 No Content`, it MUST NOT modify the Statement or any other Object.

If the LRS receives a Statement with an id it already has a Statement for, it SHOULD verify the received Statement matches the existing one and return `409 Conflict` if they do not match.

The LRS MAY respond before Statements that have been stored are available for retrieval.

GET Statements

Example endpoint: `http://example.com/xAPI/statements`

This method may be called to fetch a single Statement or multiple Statements. If the statementId or voidedStatementId parameter is specified a single Statement is returned.

Otherwise returns: A StatementResult Object, a list of Statements in reverse chronological order based on "stored" time, subject to permissions and maximum list length. If additional results are available, an IRL to retrieve them will be included in the StatementResult Object.

Returns: `200 OK`, Statement or Statement Result (See section 4.2 for details)

Parameter	Type	Default	Description
statementid	String		id of statement to fetch
voidedStatementid	String		id of voided statement to fetch. see Voided Statements
agent	Agent or Identified Group Object (JSON)		Filter, only return Statements for which the specified Agent or group is the Actor or Object of the Statement. ○ Agents or identified groups are equal when the same Inverse Functional Identifier is used in each Object compared and those Inverse Functional Identifiers have equal values. ○ For the purposes of this filter, groups that have members which match the specified Agent based on their Inverse Functional Identifier as described above are considered a match See agent/group Object definition for details
verb	Verb id (IRI)		Filter, only return statements matching the specified verb id.
activity	Activity id (IRI)		Filter, only return statements for which the Object of the statement is an Activity with the specified id.
registration	UUID		Filter, only return Statements matching the specified registration id. Note that although frequently a unique registration id will be used for one Actor assigned to one Activity, this should not be assumed. If only Statements for a certain Actor or Activity should be returned, those parameters should also be specified.
related_activities	Boolean	False	Apply the Activity filter broadly. Include Statements for which the Object, any of the context Activities, or any of those properties in a contained SubStatement match the Activity parameter, instead of that parameter's normal behavior. Matching is defined in the same way it is for the 'Activity' parameter."
related_agents	Boolean	False	Apply the Agent filter broadly. Include Statements for which the Actor, Object, authority, instructor, team, or any of these properties in a contained SubStatement match the Agent parameter, instead of that parameter's normal behavior. Matching is defined in the same way it is for the 'agent' parameter.

Parameter	Type	Default	Description
since	Timestamp		Only Statements stored since the specified timestamp (exclusive) are returned.
until	Timestamp		Only Statements stored at or before the specified timestamp are returned.
limit	Nonnegative Integer	0	Maximum number of Statements to return. 0 indicates return the maximum the server will allow.
format	String: ("ids", "exact", or "canonical")	exact	If "ids", only include minimum information necessary in Agent, Activity, and group objects to identify them. For anonymous groups this means including the minimum information needed to identify each member. If "exact", return Agent, Activity, and group Objects populated exactly as they were when the Statement was received. If "canonical", return activity objects populated with the canonical definition of the activity Objects as determined by the LRS, after applying the language filtering process defined below, and return the original Agent Objects as in "exact" mode. Activity Objects contain Language Map objects for name and description. Only one language should be returned in each of these maps. In order to provide these strings in the most relevant language, the LRS will apply the Accept-Language header as described in RFC 2616 (HTTP 1.1), except that this logic will be applied to each language map individually to select which language entry to include, rather than to the resource (list of Statements) as a whole. An LRS requesting Statements for the purpose of importing them SHOULD use a format of "exact"
attachments	Boolean	False	If true LRS MUST use the multipart response format and include any attachments as described in 4.1.11. Attachments, otherwise the LRS MUST NOT include attachment raw data and MUST sent the prescribed response with Content-Type application/json.
ascending	Boolean	False	If true, return results in ascending order of stored time.

The LRS MUST reject with an `HTTP 400` error any requests to this resource which:

- contain both statementId and voidedStatementId parameters;
- contain statementId or voidedStatementId parameters, and also contain any other parameter besides "attachments" or "format".

The LRS MUST include the header "X-Experience-API-Consistent-Through", in ISO 8601 combined date and time format, on all responses to Statements requests, with a value of the timestamp for which all Statements that have or will have a "stored" property before that time are known with reasonable certainty to be available for retrieval. This time SHOULD take into account any temporary condition, such as excessive load, which might cause a delay in Statements becoming available for retrieval.

Note Due to query string limits, this method MAY be called using POST and form fields if necessary. The LRS MUST differentiate a POST to add a Statement or to list Statements based on the parameters passed.

Filter Conditions for StatementRefs

For filter parameters which are not time or sequence based (that is, other than since, until, or limit), Statements which target another Statement (by using a StatementRef as the Object of the Statement) will meet the filter condition if the targeted Statement meets the condition. The time and sequence based parameters must still be applied to the Statement making the StatementRef in this manner. This rule applies recursively, so that "Statement a" is a match when a targets b which targets c and the filter conditions described above match for "Statement c".

For example, consider the Statement "Ben passed explosives training", and a follow up Statement: "Andrew confirmed <StatementRef to original statement>". The follow up Statement will not mention "Ben" or "explosives training", but when fetching Statements with an Actor filter of "Ben" or an Activity filter of "explosives training", both Statements match and will be returned so long as they fall into the time or sequence being fetched.

This section does not apply when retrieving Statements with statementId or voidedStatementId.

Note StatementRefs used in the Statement field in context do not affect how Statements are filtered.

Voided Statements

The LRS MUST not return any Statement which has been voided, unless that Statement has been requested by voidedStatementId. The LRS MUST still return any Statements targeting the voided Statement when retrieving statements using explicit or implicit time or sequence based retrieval, unless they themselves have been voided, as described in the section on filter conditions for StatementRefs. This includes the voiding Statement, which cannot be voided. Reporting tools can identify the presence and statementId of any voided Statements by the target of the voiding Statement. Reporting tools wishing to retrieve voided statements SHOULD request these individually by voidedStatementId.

Document APIs 7.3

The 3 Document APIs provide document storage for Activity Providers and Agents. The details of each API are found in the following sections, and the information in this section applies to all three APIs.

New Agents and Activities

An Activity Provider MAY send documents to any of the document APIs for Activities and Agents that the LRS does not have prior knowledge of. The LRS MUST NOT reject documents on the basis of not having prior knowledge of the Activity and/or Agent.

POST to store application/json arrays of variables

API	Method	Endpoint	Example
State API	POST	activities/state	`http://example.com/xAPI/activities/state`
Activity Profile API	POST	activities/profile	`http://example.com/xAPI/activities/profile`
Agent Profile API	POST	agent/profile	`http://example.com/xAPI/agents/profile`

APs MAY use Documents of content type "application/json" to store sets of variables. For example a document contains:

```
{
    "x" : "foo",
    "y" : "bar"
}
```

When an LRS receives a POST request with content type application/json for an existing document also of content type application/json, it MUST merge the posted document with the existing document. In this context ‘merge’ is defined as:

- de-serialize the Objects represented by each document.
- for each property directly defined on the Object being posted, set the corresponding property on the existing Object equal to the value from the posted Object.
- store any valid json serialization of the existing Object as the document referenced in the request.

Note that only top-level properties are merged, even if a top-level property is an Object. The entire contents of each original property are replaced with the entire contents of each new property.

For example, this document is POSTed with the same id as the existing document above:

```
{
    "x" : "bash",
    "z" : "faz"
}
```

the resulting document stored in the LRS is:

```
{
"x" : "bash",
"y" : "bar",
"z" : "faz"
}
```

If the original document exists, and the original document or the document being posted do not have a Content-Type: of "application/json", or if either document cannot be parsed as JSON Objects, the LRS MUST respond with HTTP status code `400 Bad Request`, and MUST NOT update the target document as a result of the request.

If the original document does not exist, the LRS MUST treat the request the same as it would a PUT request and store the document being posted.

If the merge is successful, the LRS MUST respond with HTTP status code `204 No Content.`

If an AP needs to delete a property, it SHOULD use a PUT request to replace the whole document as described below.

State API 7.4

Description Generally, this is a scratch area for Activity Providers that do not have their own internal storage, or need to persist state across devices. When using the State API, be aware of how the stateId parameter affects the semantics of the call. If it is included, the GET and DELETE methods will act upon a single defined state document identified by "stateId". Otherwise, GET will return the available ids, and DELETE will delete all state in the context given through the other parameters.

PUT | POST | GET | DELETE activities/state

Example endpoint: `http://example.com/xAPI/activities/state`

Stores, fetches, or deletes the document specified by the given stateId that exists in the context of the specified Activity, Agent, and registration (if specified).

Returns: (PUT | POST | DELETE) `204 No Content`, (GET) `200 OK`, State Content

Parameter	Type	Required	Description
activityid	String	yes	The activity id associated with this state.
agent	JSON	yes	The Agent associated with this state.
registration	UUID	no	The registration id associated with this state.
stateid	String	yes	The id for this state, within the given context.

GET activities/state

Example endpoint: `http://example.com/xAPI/activities/state`

Fetches ids of all state data for this context (Activity + agent [+ registration if specified]). If “since” parameter is specified, this is limited to entries that have been stored or updated since the specified timestamp (exclusive).

Returns: `200 OK`, Array of ids

Parameter	Type	Required	Description
activityid	String	yes	The Activity id associated with these states.
agent	JSON	yes	The Actor associated with these states.
registration	UUID	no	The registration id associated with these states.
since	Timestamp	no	Only ids of states stored since the specified timestamp (exclusive) are returned.

DELETE activities/state

Example endpoint: `http://example.com/xAPI/activities/state`

Deletes all state data for this context (activity + agent [+ registration if specified]).

Returns: `204 No Content`

Parameter	Type	Required	Description
activityid	String	yes	The activity id associated with this state.
agent	JSON	yes	The Actor associated with this state.
registration	UUID	no	The registration id associated with this state.

Activity Profile API 7.5

Description

The Activity Profile API is much like the State API, allowing for arbitrary key / document pairs to be saved which are related to an Activity. When using the Activity Profile API for manipulating documents, be aware of how the profileId parameter affects the semantics of the call. If it is included, the GET and DELETE methods will act upon a single defined document identified by "profileId". Otherwise, GET will return the available ids, and DELETE will delete all state in the context given through the other parameters.

The Activity Profile API also includes a method to retrieve a full description of an Activity from the LRS.

GET activities

Example endpoint: `http://example.com/xAPI/activities`

Loads the complete Activity Object specified.

Returns: `200 OK`, Content

Parameter	Type	Required	Description
activityid	String	yes	The id associated with the Activities to load.

PUT | POST | GET | DELETE activities/profile

Example endpoint: `http://example.com/xAPI/activities/profile`

Saves/retrieves/deletes the specified profile document in the context of the specified Activity.

Returns: (PUT | POST | DELETE) `204 No Content`, (GET) `200 OK`, Profile Content

Parameter	Type	Required	Description
activityid	String	yes	The Activity id associated with this profile.
profileid	String	yes	The profile id associated with this profile.

GET activities/profile

Example endpoint: `http://example.com/xAPI/activities/profile`

Loads ids of all profile entries for an Activity. If "since" parameter is specified, this is limited to entries that have been stored or updated since the specified timestamp (exclusive).

Returns: `200 OK`, List of ids

Parameter	Type	Required	Description
activityid	String	yes	The Activity id associated with these profiles.
since	Timestamp	no	Only ids of profiles stored since the specified timestamp (exclusive) are returned.

Agent Profile API 7.6

Description

The Agent Profile API is much like the State API, allowing for arbitrary key / document pairs to be saved which are related to an Agent. When using the Agent Profile API for manipulating documents, be aware of how the profileId parameter affects the semantics of the call. If it is included, the GET and DELETE methods will act upon a single defined document identified by "profileId". Otherwise, GET will return the available ids, and DELETE will delete all state in the context given through the other parameters.

The Agent Profile API also includes a method to retrieve a special Object with combined information about an Agent derived from an outside service, such as a directory service.

GET agents

Example endpoint: `http://example.com/xAPI/agents`

Return a special, Person Object for a specified Agent. The Person Object is very similar to an Agent Object, but instead of each attribute having a single value, each attribute has an array value, and it is legal to include multiple identifying properties. Note that the argument is still a normal Agent object with a single identifier and no arrays. Note that this is different from the FOAF concept of person, person is being used here to indicate a person-centric view of the LRS Agent data, but Agents just refer to one persona (a person in one context).

An LRS capable of returning multiple identifying properties for a Person SHOULD require the connecting credentials have increased, explicitly given permissions. An LRS SHOULD reject insufficiently privileged requests with 403 "Forbidden". If an LRS does not have any additional information about an Agent to return, the LRS MUST still return a Person when queried, but that Person Object will only include the information associated with the requested Agent.

Person Properties

All array properties must be populated with members with the same definition as the similarly named property from Agent Objects.

Property	Type	Description
objectType	String	"Person". Required.
name	Array of strings.	Optional. List of names of Agents to retrieve.
mbox	Array of IRIs in the form "mailto:email address".	List of e-mail addresses of Agents to retrieve.
mbox_sha1sum	Array of strings.	List of the SHA1 hashes of mailto IRIs (such as go in an mbox property).
openid	Array of strings.	List of openids that uniquely identify the Agents to retrieve.
account	Array of account objects.	List of accounts to match. Complete account Objects (homePage and name) must be provided.

See also: Section 4.1.2.1 Agent.

Returns `200 OK`, Expanded Agent Object

Parameter	Type	Required	Description
agent	Object (JSON)	yes	The Agent representation to use in fetching expanded Agent information.

PUT | POST | GET | DELETE agents/profile

Example endpoint: `http://example.com/xAPI/agents/profile`

Saves/retrieves/deletes the specified profile document in the context of the specified Agent.

Returns: (PUT | POST | DELETE) `204 No Content`, (GET) `200 OK`, Profile Content

Parameter	Type	Required	Description
agent	Object (JSON)	yes	The Agent associated with this profile.
profileid	String	yes	The profile id associated with this profile.

GET agents/profile

Example endpoint: `http://example.com/xAPI/agents/profile`

Loads ids of all profile entries for an Agent. If "since" parameter is specified, this is limited to entries that have been stored or updated since the specified timestamp (exclusive).

Returns: `200 OK` , List of IDs

Parameter	Type	Required	Description
agent	Object (JSON)	yes	The Agent associated with this profile.
since	Timestamp	no	Only ids of profiles stored since the specified timestamp (exclusive) are returned

About resource 7.7

GET about

Example endpoint: `http://example.com/xAPI/about`

Description Returns JSON Object containing information about this LRS, including the xAPI version supported.

Rationale Primarily this resource exists to allow Clients that support multiple xAPI versions to decide which version to use when communicating with the LRS. Extensions are included to allow other uses to emerge.

Details Returns: `200 OK`, Single 'about' JSON document.

Property	Type	Description
version	string	xAPI version this LRS supports
Extensions	Object	A map of other properties as needed.

LRS Requirements The LRS:

- MUST return the JSON document described above, with a version property that includes the latest minor and patch version the LRS conforms to, for each major version.
  - For version 1.0.0 of this specification, this means that "1.0.0" MUST be included; "0.9" and "0.95" MAY be included. (For the purposes of this requirement, "0.9" and "0.95" are considered major versions.).
- SHOULD allow unauthenticated access to this resource.
- MUST NOT reject requests based on their version header as would otherwise be required by 6.2 API Versioning.

Cross Origin Requests 7.8

One of the goals of the xAPI is to allow cross-domain tracking, and even though xAPI seeks to enable tracking from applications other than browsers, browsers still need to be supported. Internet Explorer 8 and 9 do not implement Cross Origin Resource Sharing, but rather use their own Cross Domain Request API, which cannot use all of the xAPI as described above due to only supporting "GET" and "POST", and not allowing HTTP headers to be set.

The following describes alternate syntax for consumers to use only when unable to use the usual syntax for specific calls due to the restrictions mentioned above. All LRSs must support this syntax.

Method: All xAPI requests issued must be POST. The intended xAPI method must be included as the only query string parameter on the request.
Example: `http://example.com/xAPI/statements?method=PUT`

Headers: Any required parameters which are expected to appear in the HTTP header must instead be included as a form parameter with the same name.

Content: If the xAPI call involved sending content, that content must now be encoded and included as a form parameter called "content". The LRS will interpret this content as a UTF-8 string. Storing binary data is not supported with this syntax.

Attachments: Sending attachment data requires sending a multipart/mixed request, therefore sending attachment data is not supported with this syntax. See 4.1.11. Attachments

See Appendix B for an example function written in JavaScript which transforms a normal request into one using this alternate syntax.

It should also be noted that versions of Internet Explorer lower than 10 do not support Cross Domain Requests between HTTP and HTTPS. This means that for IE9 and lower, if the LRS is on an HTTPS domain, the Client sending the Statement must also be on HTTPS. If the LRS is on HTTP, the Client must be too.

There may be cases where there is a requirement for the Client Activity Provider to support IE8 and IE9 where the Client code is hosted on a different scheme (HTTP or HTTPS) from the LRS. In these cases, proxy is needed to communicate to the LRS. Two simple solutions might be to:

1) set up a proxy pass through on the same scheme as the Client code to the LRS, or…
2) to host an intermediary server side LRS on the same scheme as the Client code to route Statements to the target LRS.

An LRS MAY choose to provide both HTTP and HTTPS endpoints to support this use case. HTTP is inherently less secure than HTTPS, and both LRS and Client should consider the security risks before making the decision to use this scheme.

Validation 7.9

The function of the LRS within the xAPI is to store and retrieve Statements. As long as it has sufficient information to perform these tasks, it is expected that it does them. Validation of Statements in the Experience API is focused solely on syntax, not semantics. It SHOULD enforce rules regarding structure, but SHOULD NOT enforce rules regarding meaning. Enforcing the rules that ensure valid meaning among Verb definitions, Activity types, and extensions is the responsibility of the Activity Provider sending the Statement.

.

HTTP HEAD 7.10

Description The LRS will respond to HEAD requests by returning the meta information only, using the HTTP headers, and not the actual document.

Rationale Clients accessing the LRS may need to check if a particular Statement exists, or determine the modification date of documents such as state or Activity or Agent profile. Particularly for large documents it's more efficient not to get the entire document just to check its modification date.

Requirements

- The LRS MUST respond to any HTTP HEAD request as it would have responded to an otherwise identical HTTP GET request except:
  - The message-body MUST be omitted.
  - The Content-Length header MAY be omitted, in order to avoid wasting LRS resources.

APPENDICES

Appendix A: Bookmarklet

An xAPI Bookmarklet enables individual user tracking with base authentication. Examples could be an "I think this," "I learned this," "I like this," or "I don't like this" Statement that allows self-reporting. The following is an example of such a bookmarklet, and the Statement that this bookmarklet would send if used on the page: http://adlnet.gov/xapi

The bookmarklet MAY be provided by the LRS to track a specific user for behavior analytics.

Therefore the LRS IRL, authentication, and Actor information is hard coded into the bookmarklet. Note that since the authorization token must be included in the bookmarklet, the LRS should provide a token with limited privileges. Ideally the token should enable the storage of self-reported learning Statements only.

The UUID SHOULD be included as part of the bookmarklet PUT Statement. If a statement is POSTed without a UUID, the LRS MUST generate one.

In order to allow cross-domain reporting of Statements, a browser that supports the "Access-Control-Allow-Origin" and "Access-Control-Allow-Methods" headers must be used, such as IE 8+, FF 3.5+, Safari 4+, Safari iOS Chrome, or Android browser. Additionally the server must set the required headers.

In the example below, the following values in the first few lines should be replaced with your own values. All other values should be left as they are.

Value in example	Explanation
http://localhost:8080/xAPI/	Endpoint of the LRS to send the Statements to.
dGVzdDpwYXNzd29yZA==	Base 64 encoded username and password, usually in the form "username : password".
learner@example.adlnet.gov	Email address of the learner using the bookmarklet.

```
var url = "http://localhost:8080/xAPI/statements?statementId="+_ruuid();
var auth = "Basic dGVzdDpwYXNzd29yZA==";
var statement = {
    "actor" : {
        "objectType" : "Agent",
        "mbox" : "mailto:learner@example.adlnet.gov"
    },
    "verb" : {
        "id" : "",
        "display" : {}
    },
    "object" : {
        "id" : "",
        "definition" : {}
    }
};
var definition = statement.object.definition;

statement.verb.id = 'http://adlnet.gov/expapi/verbs/experienced';
statement.verb.display = { "en-US" : "experienced" };
statement.object.id = window.location.toString();
definition.type = "http://adlnet.gov/expapi/activities/link";

var xhr = new XMLHttpRequest();
xhr.open("PUT", url, true);
xhr.setRequestHeader("X-Experience-API-Version", "1.0");
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Authorization", auth);
xhr.onreadystatechange = function() {
    if(xhr.readyState == 4) {
        alert(xhr.status + " : " + xhr.responseText);
    }
};
xhr.send(JSON.stringify(statement));

/*!
Modified from: Math.uuid.js (v1.4)
http://www.broofa.com
mailto:robert@broofa.com

Copyright (c) 2010 Robert Kieffer
Dual licensed under the MIT and GPL licenses.
*/
function _ruuid() {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
        var r = Math.random()*16|0, v = c == 'x' ? r : (r&0x3|0x8);
        return v.toString(16);
    });
}
```

Example Statement Using Bookmarklet

Headers

```
{
    "content-type": "application/json; charset=UTF-8",
    "authorization": "d515309a-044d-4af3-9559-c041e78eb446",
    "referer": "http://adlnet.gov/xapi/",
    "content-length": "###",
    "origin": "http://adlnet.gov"
}
```

Method path

```
PUT : /xAPI/Statements/?statementId=ed1d064a-eba6-45ea-a3f6-34cdf6e1dfd9

Body:
{
    "actor": {
        "objectType": "Agent",
        "mbox": "mailto:learner@example.adlnet.gov"
    },
    "verb": "http://adlnet.gov/expapi/verbs/experienced",
    "object": {
        "id": "http://adlnet.gov/xapi/ ",
        "definition": {
            "type": "http://adlnet.gov/expapi/activities/link"
        }
    }
}
```

Appendix B: Creating an "IE Mode" Request

```
function getIEModeRequest(method, url, headers, data){

    var newUrl = url;

    // Everything that was on query string goes into form vars
    var formData = new Array();
    var qsIndex = newUrl.indexOf('?');
    if(qsIndex > 0){
        formData.push(newUrl.substr(qsIndex+1));
        newUrl = newUrl.substr(0, qsIndex);
    }

    // Method has to go on querystring, and nothing else
    newUrl = newUrl + '?method=' + method;

    // Headers
    if(headers !== null){
        for(var headerName in headers){
            formData.push(
                headerName + "=" +
                    encodeURIComponent(
                        headers[headerName]));
        }
    }

    // The original data is repackaged as "content" form var
    if(data !== null){
        formData.push('content=' + encodeURIComponent(data));
    }

    return {
        "method":"POST",
        "url":newUrl,
        "headers":{},
        "data":formData.join("&")
    };
}
```

Appendix C: Example definitions for Activities of type "cmi.interaction"

True-false

```
"definition": {
    "description": {
        "en-US": "Does the xAPI include the concept of statements?"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "true-false",
    "correctResponsesPattern": [
        "true"
    ]
}
```

Choice

```
"definition": {
    "description": {
        "en-US": "Which of these prototypes are available at the beta site?"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "choice",
    "correctResponsesPattern": [
        "golf[,]tetris"
    ],
    "choices": [
        {
            "id": "golf",
            "description": {
                "en-US": "Golf Example"
            }
        },
        {
            "id": "facebook",
            "description": {
                "en-US": "Facebook App"
            }
        },
        {
            "id": "tetris",
            "description": {
                "en-US": "Tetris Example"
            }
        },
        {
            "id": "scrabble",
            "description": {
                "en-US": "Scrabble Example"
            }
        }
    ]
}
```

Fill-in

```
"definition": {
    "description": {
        "en-US": "Ben is often heard saying: "
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "fill-in",
    "correctResponsesPattern": [
        "Bob's your uncle"
    ]
}
```

Likert

```
"definition": {
    "description": {
        "en-US": "How awesome is Experience API?"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "likert",
    "correctResponsesPattern": [
        "likert_3"
    ],
    "scale": [
        {
            "id": "likert_0",
            "description": {
                "en-US": "It's OK"
            }
        },
        {
            "id": "likert_1",
            "description": {
                "en-US": "It's Pretty Cool"
            }
        },
        {
            "id": "likert_2",
            "description": {
                "en-US": "It's Damn Cool"
            }
        },
        {
            "id": "likert_3",
            "description": {
                "en-US": "It's Gonna Change the World"
            }
        }
    ]
}
```

Matching

```
"definition": {
    "description": {
        "en-US": "Match these people to their kickball team:"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "matching",
    "correctResponsesPattern": [
        "ben[.]3[,]chris[.]2[,]troy[.]4[,]freddie[.]1"
    ],
    "source": [
        {
            "id": "ben",
            "description": {
                "en-US": "Ben"
            }
        },
        {
            "id": "chris",
            "description": {
                "en-US": "Chris"
            }
        },
        {
            "id": "troy",
            "description": {
                "en-US": "Troy"
            }
        },
        {
            "id": "freddie",
            "description": {
                "en-US": "Freddie"
            }
        }
    ],
    "target": [
        {
            "id": "1",
            "description": {
                "en-US": "Swift Kick in the Grass"
            }
        },
        {
            "id": "2",
            "description": {
                "en-US": "We got Runs"
            }
        },
        {
            "id": "3",
            "description": {
                "en-US": "Duck"
            }
        },
        {
            "id": "4",
            "description": {
                "en-US": "Van Delay Industries"
            }
```

```
        }]}
```

Performance

```
"definition": {
    "description": {
        "en-US": "This interaction measures performance over a day of RS sports:"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "performance",
    "correctResponsesPattern": [
        "pong[.]1:[,]dg[.]:10[,]lunch[.]"
    ],
    "steps": [
        {
            "id": "pong",
            "description": {
                "en-US": "Net pong matches won"
            }
        },
        {
            "id": "dg",
            "description": {
                "en-US": "Strokes over par in disc golf at Liberty"
                }
            },
        {
            "id": "lunch",
            "description": {
                "en-US": "Lunch having been eaten"
            }
        }
    ]
}
```

Sequencing

```
"definition": {
    "description": {
        "en-US": "Order players by their pong ladder position:"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "sequencing",
    "correctResponsesPattern": [
        "tim[,]mike[,]ells[,]ben"
    ],
    "choices": [
        {
            "id": "tim",
            "description": {
                "en-US": "Tim"
            }
        },
        {
            "id": "ben", "description": {
                "en-US": "Ben"
            }
        },
        {
            "id": "ells",
            "description": {
                "en-US": "Ells"
            }
        },
        {
            "id": "mike",
            "description": {
                "en-US": "Mike"
            }
        }
    ]
}
```

Numeric

```
"definition": {
    "description": {
        "en-US": "How many jokes is Chris the butt of each day?"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "numeric",
    "correctResponsesPattern": [
        "4:"
    ]
}
```

Other

```
"definition": {
    "description": {
        "en-US": "On this map, please mark Franklin, TN"
    },
    "type": "http://adlnet.gov/expapi/activities/cmi.interaction",
    "interactionType": "other",
    "correctResponsesPattern": [
        "(35.937432,-86.868896)"
    ]}
```

Appendix D: Example statements

Example of a simple statement (line breaks are for display purposes only):

```
{
    "id":"fd41c918-b88b-4b20-a0a5-a4c32391aaa0",
    "actor":{
        "objectType": "Agent",
        "name":"Project Tin Can API",
        "mbox":"mailto:user@example.com"
    },
    "verb":{
        "id":"http://adlnet.gov/expapi/verbs/created",
        "display":{
            "en-US":"created"
        }
    },
    "object":{
        "id":"http://example.adlnet.gov/xapi/example/simplestatement",
        "definition":{
            "name":{
                "en-US":"simple statement"
            },
            "description":{
                "en-US":"A simple Experience API statement. Note that the LRS
                does not need to have any prior information about the actor
(learner), the
                verb, or the activity/object."
            }
        }
    }
}
```

Typical simple completion with verb "attempted":

```
{
    "actor":{
        "objectType": "Agent",
        "name":"Example Learner",
        "mbox":"mailto:example.learner@adlnet.gov"
    },
    "verb":{
        "id":"http://adlnet.gov/expapi/verbs/attempted",
        "display":{
            "en-US":"attempted"
        }
    },
    "object":{
        "id":"http://example.adlnet.gov/xapi/example/simpleCBT",
        "definition":{
            "name":{
                "en-US":"simple CBT course"
            },
            "description":{
                "en-US":"A fictitious example CBT course."
            }
        }
    },
    "result":{
        "score":{
            "scaled":0.95
        },
        "success":true,
        "completion":true
    }
}
```

Appendix E: Converting Statements to 1.0.0

Rationale

This is a 1.0.0 specification, and as such implementers should not have to consider prior versions of the specification. However, prior versions did see notable adoption. This data conversion is specified in order to preserve the data tracked using earlier versions, and make it available to new implementers in a consistent manner.

Details

Conversion of Statements created based on version 0.9

A 1.0.0 system converting a Statement created in 0.9 MUST follow the steps below:

- If the Statement has been voided or uses Verbs, Activity types, or properties not included in the 0.9 specification, do not convert it.
- Prefix "verb" with "http://adlnet.gov/expapi/verbs/".
- Prefix any Activity ids which are not full absolute IRIs with "tag:adlnet.gov,2013:expapi:0.9:activities:"
- Prefix any extension keys which are not full absolute IRIs with "tag:adlnet.gov,2013:expapi:0.9:extensions:"
- Prefix Activity types with http://adlnet.gov/expapi/activities/ for each Agent (Actor):
  - Search for Inverse Functional Identifiers in this order: "mbox, mbox_sha1sum, openId, account". Keep the first populated Inverse Functional Identifier found, discard the rest.
  - For the above Inverse Functional Identifier, take the first element in the array and use that as the value of that Inverse Functional Identifier property, discarding any remaining elements.
  - If the "name" property is present, set it equal to the first element in the "name" array, discard the remaining elements.
  - Remove all remaining properties.
- Remove the "voided" property from the Statement, if present. Remember, if the value of the voided property is true, then the Statement MUST NOT be converted.
- Add "version": "1.0.0"
- If an authority was not previously set, set the authority to an Agent identified by an account with a homePage set to the home page corresponding to the system performing the conversion and an accountName of "unknown".
- If the Statement field in context was set, remove it from the Statement.

Preserve all other fields without modification, including "stored". Stored should still be updated if the Statement is passed to another system.

Conversion of Statements created based on version 0.95

A 1.0.0 system converting a Statement created in 0.95 MUST follow the steps below:

- If the Statement is voided, do not convert it.
  Remove the "voided" property from the Statement, if present. Remember, if the value of the voided property is true, then the Statement MUST NOT be converted.
- Add "version": "1.0.0"
- If an authority was not previously set, set the authority to an Agent identified by an account with a homePage set to the home page corresponding to the system performing the conversion and an accountName of "unknown".
- If the Statement field in context was set to anything other than a StatementRef, remove it from the Statement.
- Preserve all other fields without modification, including "stored". Stored should still be updated if the Statement is passed to another system.

Example

A 0.9 statement:

```
{
    "id": "d1eec41f-1e93-4ed6-acbf-5c4bd0c24269",
```

```
    "actor": {
        "objectType": "Person",
        "name": [
            "Joe Schmoe",
            "Joseph Schmoseph"
        ],
        "mbox": [
            "mailto:joe@example.com"
        ],
        "openid": [
            "http://openid.com/joe-schmoe"
        ]
    },
    "verb": "completed",
    "inProgress": false,
    "object": {
        "objectType": "Activity",
        "id": "http://www.example.com/activities/001",
        "definition": {
            "name": {
                "en-US": "Example Activity"
            },
            "type": "course"
        }
    },
    "result": {
        "completion": true
    },
    "context": {
        "instructor": {
            "objectType": "Person",
            "lastName": [
                "Dad"
            ],
            "firstName": [
                "Joe's"
            ],
            "mbox": [
                "mailto:joesdad@example.com"
            ]
        },
        "contextActivities": {
            "parent": {
                "objectType": "Activity",
                "id": "non-absolute-activity-id",
                "definition": {
                    "name": {
                        "en-US": "Another Activity"
                    }
                }
            }
        }
    },
    "timestamp": "2012-06-01T19:09:13.245Z",
    "stored": "2012-06-29T15:41:39.165Z"}
```

Converted to 1.0.0:


```
{
    "version": "1.0.0",
    "id": "d1eec41f-1e93-4ed6-acbf-5c4bd0c24269",
    "actor": {
        "objectType": "Agent",
        "name": "Joe Schmoe",
        "mbox": "mailto:joe@example.com"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/completed",
        "display": {
            "en-US": "completed"
        }
    },
    "object": {
        "objectType": "Activity",
        "id": "http://www.example.com/activities/001",
        "definition": {
            "name": {
                "en-US": "Example Activity"
            },
            "type": "http://adlnet.gov/expapi/activities/course"
        }
    },
    "result": {
        "completion": true
    },
    "context": {
        "instructor": {
            "objectType": "Agent",
            "mbox": "mailto:joesdad@example.com"
        },
        "contextActivities": {
            "parent": [
                {
                    "objectType": "Activity",
                    "id": "tag:adlnet.gov,2013:expapi:0.9:activities:non-absolute-
activity-id",
                    "definition": {
                        "name": {
                            "en-US": "Another Activity"
                        }
                    }
                }
            ]
        }
    },
    "timestamp": "2012-06-01T19:09:13.245Z",
    "stored": "2012-06-29T15:41:39.165Z",
    "authority": {
        "objectType": "Agent",
        "account": {
            "homePage": "http://www.example.com",
            "name": "unknown"
        }
    }
}
```

Appendix F: Example Signed Statement

An example Signed Statement, as described in: 4.4 Signed Statements.

The original Statement serialization to be signed. New lines in this example are included via CR+LF (0x0D + 0x0A).

```
{
    "version": "1.0.0",
    "id": "33cff416-e331-4c9d-969e-5373a1756120",
    "actor": {
        "mbox": "mailto:example@example.com",
        "name": "Example Learner",
        "objectType": "Agent"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/experienced",
        "display": {
            "en-US": "experienced"
        }
    },
    "object": {
        "id": "https://www.youtube.com/watch?v=xh4kIiH3Sm8",
        "objectType": "Activity",
        "definition": {
            "name": {
                "en-US": "Tax Tips & Information : How to File a Tax Return "
            },
            "description": {
                "en-US": "Filing a tax return will require filling out either a
1040, 1040A or 1040EZ form"
            }
        }
    },
    "timestamp": "2013-04-01T12:00:00Z"
}
```

Example private key for X.509 certificate that will be used for signing

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDjxvZXF30WL4oKjZYXgR0ZyaX+u3y6+JqTqiNkFa/VTnet6Ly2
OT6ZmmcJEPnq3UnewpHoOQ+GfhhTkW13j06j5iNn4obcCVWTL9yXNvJH+Ko+xu4Y
l/ySPRrIPyTjtHdG0M2XzIlmmLqm+CAS+KCbJeH4tf543kIWC5pC5p3cVQIDAQAB
AoGAOejdvGq2XKuddu1kWXl0Aphn4YmdPpPyCNTaxplU6PBYMRjY0aNgLQE6bO2p
/HJiU4Y4PkgzkEgCu0xf/mOq5DnSkX32ICoQS6jChABAe20ErPfm5t8h9YKsTfn9
40lAouuwD9ePRteizd4YvHtiMMwmh5PtUoCbqLefawNApAECQQD1mdBW3zL0okUx
2pc4tttn2qArCG4CsEZMLlGRDd3FwPWJz3ZPNEEgZWXGSpA9F1QTZ6JYXIfejjRo
UuvRMWeBAkEA7WvzDBNcv4N+xeUKvH8ILti/BM58LraTtqJlzjQSovek0srxtmDg
5of+xrxN6IM4p7yvQa+7YOUOukrVXjG+1QJBAI2mBrjzxgm9xTa5odn97JD7UMFA
/WHjlMe/Nx/35V52qaav1sZbluw+TvKMcqApYj5G2SUpSNudHLDGkmd2nQECQFfc
lBRK8g7ZncekbGW3aRLVGVOxClnLLTzwOlamBKOUm8V6XxsMHQ6TE2D+fKJoNUY1
2HGpk+FWwy2D1hRGuoUCQAXfaLSxtaWdPtlZTPVueF7ZikQDsVg+vtTFgpuHloR2
6EVc1RbHHZm32yvGDY8IkcoMfJQqLONDdLfS/05yoNU=
-----END RSA PRIVATE KEY-----
```

Example public X.509 certificate

```
-----BEGIN CERTIFICATE-----
MIIDATCCAmqgAwIBAgIJAMB1csNuA6+kMA0GCSqGSIb3DQEBBQUAMHExCzAJBgNV
BAYTAlVTMRIwEAYDVQQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUgQ29t
cGFueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCAGCSqGSIb3DQEJARYTZXhhbXBsZUBl
eGFtcGxlLmNvbTAeFw0xMzA0MDQxNTI4MzBaFw0xNDA0MDQxNTI4MzBaMIGWMQsw
CQYDVQQGEwJVUzESMBAGA1UECBMJVGVubmVzc2VlMREwDwYDVQQHEwhGcmFua2xp
bjEYMBYGA1UEChMPRXhhbXBsZSBDb21wYW55MRAwDgYDVQQLEwdFeGFtcGxlMRAw
DgYDVQQDEwdFeGFtcGxlMSIwIAYJKoZIhvcNAQkBFhNleGFtcGxlQGV4YW1wbGUu
Y29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDjxvZXF30WL4oKjZYXgR0Z
yaX+u3y6+JqTqiNkFa/VTnet6Ly2OT6ZmmcJEPnq3UnewpHoOQ+GfhhTkW13j06j
5iNn4obcCVWTL9yXNvJH+Ko+xu4Yl/ySPRrIPyTjtHdG0M2XzIlmmLqm+CAS+KCb
JeH4tf543kIWC5pC5p3cVQIDAQABo3sweTAJBgNVHRMEAjAAMCwGCWCGSAGG+EIB
DQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4EFgQUVs3v
5afEdOeoYeVajAQE4v0WS1QwHwYDVR0jBBgwFoAUyVIc3yvra4EBz20I4BF39IAi
xBkwDQYJKoZIhvcNAQEFBQADgYEAgS/FF5D0Hnj44rvT6kgn3kJAvv2lj/fyjztK
IrYS33ljXGn6gGyA4qtbXA23PrO4uc/wYCSDICDpPobh62xTCd9qObKhgwWOi05P
SBLqUu3mwfAe15LJBJBqPVZ4K0kppePBU8m6aIZoH57L/9t4OoaL8yKs/qjKFeI1
OFWZxvA=
-----END CERTIFICATE-----
```

Example certificate authority certificate

```
-----BEGIN CERTIFICATE-----
MIIDNzCCAqCgAwIBAgIJAMB1csNuA6+jMA0GCSqGSIb3DQEBBQUAMHExCzAJBgNV
BAYTAlVTMRIwEAYDVQQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUgQ29t
cGFueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCAGCSqGSIb3DQEJARYTZXhhbXBsZUBl
eGFtcGxlLmNvbTAeFw0xMzA0MDQxNTI1NTNaFw0yMzA0MDIxNTI1NTNaMHExCzAJ
BgNVBAYTAlVTMRIwEAYDVQQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUg
Q29tcGFueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCAGCSqGSIb3DQEJARYTZXhhbXBs
ZUBleGFtcGxlLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1sBnBWPZ
0f7WJUFTJy5+01SlS5Z6DDD6Uye9vK9AycgV5B3+WC8HC5u5h91MujAC1ARPVUOt
svPRs45qKNFIgIGRXKPAwZjawEI2sCJRSKV47i6B8bDv4WkuGvQaveZGI0qlmN5R
1Eim2gUItRj1hgcC9rQavjlnFKDY2rlXGukCAwEAAaOB1jCB0zAdBgNVHQ4EFgQU
yVIc3yvra4EBz20I4BF39IAixBkwgaMGA1UdIwSBmzCBmIAUyVIc3yvra4EBz20I
4BF39IAixBmhdaRzMHExCzAJBgNVBAYTAlVTMRIwEAYDVQQIEwlUZW5uZXNzZWUx
GDAWBgNVBAoTD0V4YW1wbGUgQ29tcGFueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCAG
CSqGSIb3DQEJARYTZXhhbXBsZUBleGFtcGxlLmNvbYIJAMB1csNuA6+jMAwGA1Ud
EwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADgYEADhwTebGk735yKhm8DqCxvNnEZ0Nx
sYEYOjgRG1yXTlW5pE691fSH5AZ+T6fpwpZcWY5QYkoN6DnwjOxGkSfQC3/yGmcU
DKBPwiZ5O2s9C+fE1kUEnrX2Xea4agVngMzR8DQ6oOauLWqehDB+g2ENWRLoVgS+
ma5/Ycs0GTyrECY=
-----END CERTIFICATE-----
```

JWS Header

Note that along with specifying the algorithm, the certificate chain for the signing certificate has been included.


```
{
    "alg": "RS256",
    "x5c": [

"MIIDATCCAmqgAwIBAgIJAMB1csNuA6+kMA0GCSqGSIb3DQEBBQUAMHExCzAJBgNVBAYTAlVTMRIwEAYDV
QQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUgQ29tcGFueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCA
GCSqGSIb3DQEJARYTZXhhbXBsZUBleGFtcGxlLmNvbTAeFw0xMzA0MDQxNTI4MzBaFw0xNDA0MDQxNTI4M
zBaMIGWMQswCQYDVQQGEwJVUzESMBAGA1UECBMJVGVubmVzc2VlMREwDwYDVQQHEwhGcmFua2xpbjEYMBY
GA1UEChMPRXhhbXBsZSBDb21wYW55MRAwDgYDVQQLEwdFeGFtcGxlMRAwDgYDVQQDEwdFeGFtcGxlMSIwI
AYJKoZIhvcNAQkBFhNleGFtcGxlQGV4YW1wbGUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQD
jxvZXF30WL4oKjZYXgR0ZyaX+u3y6+JqTqiNkFa/VTnet6Ly2OT6ZmmcJEPnq3UnewpHoOQ+GfhhTkW13j
06j5iNn4obcCVWTL9yXNvJH+Ko+xu4Yl/ySPRrIPyTjtHdG0M2XzIlmmLqm+CAS+KCbJeH4tf543kIWC5p
C5p3cVQIDAQABo3sweTAJBgNVHRMEAjAAMCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZ
XJ0aWZpY2F0ZTAdBgNVHQ4EFgQUVs3v5afEdOeoYeVajAQE4v0WS1QwHwYDVR0jBBgwFoAUyVIc3yvra4E
Bz20I4BF39IAixBkwDQYJKoZIhvcNAQEFBQADgYEAgS/FF5D0Hnj44rvT6kgn3kJAvv2lj/fyjztKIrYS3
3ljXGn6gGyA4qtbXA23PrO4uc/wYCSDICDpPobh62xTCd9qObKhgwWOi05PSBLqUu3mwfAe15LJBJBqPVZ
4K0kppePBU8m6aIZoH57L/9t4OoaL8yKs/qjKFeI1OFWZxvA=",

"MIIDNzCCAqCgAwIBAgIJAMB1csNuA6+jMA0GCSqGSIb3DQEBBQUAMHExCzAJBgNVBAYTAlVTMRIwEAYDV
QQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUgQ29tcGFueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCA
GCSqGSIb3DQEJARYTZXhhbXBsZUBleGFtcGxlLmNvbTAeFw0xMzA0MDQxNTI1NTNaFw0yMzA0MDIxNTI1N
TNaMHExCzAJBgNVBAYTAlVTMRIwEAYDVQQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUgQ29tcGF
ueTEQMA4GA1UEAxMHRXhhbXBsZTEiMCAGCSqGSIb3DQEJARYTZXhhbXBsZUBleGFtcGxlLmNvbTCBnzANB
gkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1sBnBWPZ0f7WJUFTJy5+01S1S5Z6DDD6Uye9vK9AycgV5B3+WC8
HC5u5h91MujAC1ARPVUOtsvPRs45qKNFIgIGRXKPAwZjawEI2sCJRSKV47i6B8bDv4WkuGvQaveZGI0qlm
N5R1Eim2gUItRj1hgcC9rQavjlnFKDY2rlXGukCAwEAAaOB1jCB0zAdBgNVHQ4EFgQUyVIc3yvra4EBz20
I4BF39IAixBkwgaMGA1UdIwSBmzCBmIAUyVIc3yvra4EBz20I4BF39IAixBmhdaRzMHExCzAJBgNVBAYTA
lVTMRIwEAYDVQQIEwlUZW5uZXNzZWUxGDAWBgNVBAoTD0V4YW1wbGUgQ29tcGFueTEQMA4GA1UEAxMHRXh
hbXBsZTEiMCAGCSqGSIb3DQEJARYTZXhhbXBsZUBleGFtcGxlLmNvbYIJAMB1csNuA6+jMAwGA1UdEwQFM
AMBAf8wDQYJKoZIhvcNAQEFBQADgYEADhwTebGk735yKhm8DqCxvNnEZ0NxsYEYOjgRG1yXTlW5pE691fS
H5AZ+T6fpwpZcWY5QYkoN6DnwjOxGkSfQC3/yGmcUDKBPwiZ5O2s9C+fE1kUEnrX2Xea4agVngMzR8DQ6o
OauLWqehDB+g2ENWRLoVgS+ma5/Ycs0GTyrECY="
    ]
}
```

JWS signature

ew0KICAgICJhbGciOiAiUlMyNTYiLA0KICAgICJ4NWMiOiBbDQogICAgICAgICJNSUlEQVRDQ0FtcWdBd0
lCQWdJSkFNQjFjc051QTYra01BMEdDU3FHU0liM0RRRUJCUVVBTUhFeEN6QUpCZ05WQkFZVEFsVlRNUkl3
RUFZRFZRUUlFd2xVWlc1dVpYTnpaV1V4R0RBV0JnTlZCQW9URDBWNFlXMXdiR1VnUTI5dGNHRnVlVEVRTU
E0R0ExVUVBeE1IUlhoaGJYQnNaVEVpTUNBR0NTcUdTSWIzRFFFSkFSWVRaWGhoYlhCc1pVQmxlR0Z0Y0d4
bExtTnZiVEFlRncweE16QTBNRFF4TlRJNE16QmFGdzB4TkRBME1EUXhOVEk0TXpCYU1JR1dNUXN3Q1FZRF
ZRUUdFd0pWVXpFU01CQUdBMVVFQ0JNSlZHVnVibVZ6YzJWbE1SRXdEd1lEVlFRSEV3aEdjbUZ1YTJ4cGJq
RVlNQllHQTFVRUNoTVBSWGhoYlhCc1pTQkRiMjF3Vc1NU1SQXdEZ1lEVlFRTEV3ZEZlR0Z0Y0d4bE1SQX
dEZ1lEVlFRREV3ZEZlR0Z0Y0d4bE1TSXdJQVlKS29aSWh2Y05BUWtCRmhObGVHRnRjR3hsUUdWNFlXMXdi
R1V1WTI5dE1JR2ZNQTBHQ1NxR1NJYjNEUUVCQVFVQUE0R05BRENCaVFLQmdRRGp4dlpYRjMwV0w0b0tqWl
lYZ1IwWnlhWCt1M3k2K0pxVHFpTmtGYS9WVG5ldDZMeTJPVDZabW1jSkVQbnEzVW5ld3BIb09RK0dmaGhU
a1cxM2owNmo1aU5uNG9iY0NWV1RMOXlYTnZKSCtLbyt4dTRZbC95U1BSck1QeVRqdEhkRzBNMlh6SWxtbU
xxbStDQVMrS0NiSmVINHRmNTQza0lXQzVwQzVwM2NWUUlEQVFBQm8zc3dlVEFKQmdOVkhSTUVBakFBTUN3
R0NXQ0dTQUdHK0VJQkRRUWZGaDFQY0dWdVUxTk1JRWRsYm1WeVlYUmxaQ0JEWlhKMGFXWnBZMkYwWlRBZE
JnTlZIUTRFRmdRVVZzM3Y1YWZFZE9lb1llVmFqQVFFNHYwV1MxUXdId1lEVlIwakJCZ3dGb0FVeVZJYzN5
dnJhNEVCejIwSTRCRjM5SUFpeEJrd0RRWUpLb1pJaHZjTkFRUZCUUFEZ1lFQWdTL0ZGNUQwSG5qNDRydl
Q2a2duM2tKQXZ2MmxqL2Z5anp0S0lyWVMzM2xqWEduNmdHeUE0cXRiWEEyM1ByTzR1Yy93WUNTRElDRHBQ
b2JoNjJ4VENkOXFPYktoZ3dXT2kwNVBTQkxxVXUzbXdmQWUxNUxKQkpCcVBWWjRLMGtwcGVQQlU4bTZhSV
pvSDU3TC85dDRPb2FMOHlLcy9xaktGZUkxT0ZXWnh2QT0iLA0KICAgICAgICAiTUlJRE56Q0NBcUNnQXdJ
QkFnSUpBTUIxY3NOdUE2K2pNQTBHQ1NxR1NJYjNEUUVCQlFVQU1IRXhDekFKQmdOVkJBWVRBbFZUTVJJd0
VBWURWUVFJRXdsVVpXNXVaWE56WldVeEdEQVdCZ05WQkFvVEQwVjRZVzF3YkdVZ1EyOXRjR0Z1ZVRFUU1B
NEdBMVVFQXhNSFJYaGhiWEJzWlRFaU1DQUdDU3FHU0liM0RRRUpBUllUWlhoaGJYQnNaVUJsZUdGdGNHeG
xMbU52YlRBZUZ3MHhNekEwTURReE5USTFOVE5hRncweU16QTBNREl4TlRJMU5UTmFNSEV4Q3pBSkJnTlZC
QVlUQWxWVE1SSXdFQVlEVlFRSUV3bFVaVzV1WlhOelpXVXhHREFXQmdOVkJBb1REMFY0WVcxd2JHVWdRMj
l0Y0dGdWVURVFNQTRHQTFVRUF4TUhSWGhoYlhCc1pURWlNQ0FHQ1NxR1NJYjNEUUVKQVJZVFpYaGhiWEJz
WlVCbGVHRnRjR3hsTG1OdmJUQ0JuekFOQmdrcWhraUc5dzBCQVFFRkFBT0JqUUF3Z1lrQ2dZRUExc0JuQl
dQWjBmN1dKVUZUSnk1KzAxU2xTNVo2RERENlV5ZTl2SzlBeWNnVjVCMytXQzhIQzV1NWg5MU11akFDMUFS
UFZVT3RzdlBSczQ1cUtORklnSUdSWEtQQXdaamF3RUkyc0NKUlNLVjQ3aTZCOGJEdjRXa3VHdlFhdmVaR0
kwcWxtTjVSMUVpbTJnVUl0UmoxaGdjQzlyUWF2amxuRktEWTJybFhHdWtDQXdFQUFhT0IxakNCMHpBZEJn
TlZIUTRFRmdRVXlWSWMzeXZyYTRFQnoyMEk0QkYzOUlBaXhCa3dnYU1HQTFVZEl3U0JtekNCbUlBVXlWSW
MzeXZyYTRFQnoyMEk0QkYzOUlBaXhCbWhkYVJ6TUhFeEN6QUpCZ05WQkFZVEFsVlRNUkl3RUFZRFZRUUlF
d2xVWlc1dVpYTnpaV1V4R0RBV0JnTlZCQW9URDBWNFlXMXdiR1VnUTI5dGNHRnVlVEVRTUE0R0ExVUVBeE
1IUlhoaGJYQnNaVEVpTUNBR0NTcUdTSWIzRFFFSkFSWVRaWGhoYlhCc1pVQmxlR0Z0Y0d4bExtTnZiWUlK
QU1CMWNzTnVBNitqTUF3R0ExVWRFd1FGTUFNQkFmOHdEUVlKS29aSWh2Y05BUUVGQlFBRGdZRUFEaHdUZW
JHazczNXlLaG04RHFDeHZObkVaME54c1lFWU9qZ1JHMXlYVGxXNXBFNjkxZlNINUFaK1Q2ZnB3cFpjV1k1
UVlrb042RG53ak94R2tTZlFDMy95R21jVURLQlB3aVo1TzJzOUMrZkUxa1VFbnJYMlhlYTRhZ1ZuZ016UjJ
hEUTZvT2F1TFdxZWhEQitnMkVOV1JMb1ZnUyttYTUvWWNzMEdUeXJFQ1k9Ig0KICAgIF0NCn0.ew0KICAg
ICJ2ZXJzaW9uIjogIjEuMC4wIiwNCiAgICAiaWQiOiAiMzNjZmY0MTYtZTMzMS00YzlkLTk2OWUtNTM3M2
ExNzU2MTIwIiwNCiAgICAiYWN0b3IiOiB7DQogICAgICJtYm94IjogIm1haWx0bzpleGFtcGxlQGV4
YW1wbGUuY29tIiwNCiAgICAgICAgIm5hbWUiOiAiRXhhbXBsZSBMZWFybmVyIiwNCiAgICAgICAgIm9iam
VjdFR5cGUiOiAiQWdlbnQiDQogICAgfSwNCiAgICAidmVyYiI6IHsNCiAgICAgICAgImlkIjogImh0dHA6
Ly9hZGxuZXQuZ292L2V4cGFwaS92ZXJicy9leHBlcmllbmNlZCIsDQogICAgICAgICJkaXNwbGF5Ijogew
0KICAgICAgICAgICAgImVuLVVTIjogImV4cGVyaWVuY2VkIg0KICAgICAgICB9DQogICAgfSwNCiAgICAi
b2JqZWN0Ijogew0KICAgICAgICAiaWQiOiAiaHR0cHM6Ly93d3cueW91dHViZS5jb20vd2F0Y2g_dj14aD
RrSWlIM1NtOCIsDQogICAgICAgICJvYmplY3RUeXBlIjogIkFjdGl2aXR5IiwNCiAgICAgICAgImRlZmlu
aXRpb24iOiB7DQogICAgICAgICAgICAibmFtZSI6IHsNCiAgICAgICAgICAgICAgICAiZW4tVVMiOiAiVG
F4IFRpcHMgJiBJbmZvcm1hdGlvbiA6IEhvdyB0byBGaWxlIGEgVGF4IFJldHVybiAiDQogICAgICAgICAg
ICB9LA0KICAgICAgICAgICAgImRlc2NyaXB0aW9uIjogew0KICAgICAgICAgICAgICAgICJlbi1VUyI6IC
JGaWxpbmcgYSB0YXggcmV0dXJuIHdpbGwgcmVxdWlyZSBmaWxsaW5nIG91dCBlaXRoZXIgYSAxMDQwLCAx
MDQwQSBvciAxMDQwRVogZm9ybSINCiAgICAgICAgICAgIH0NCiAgICAgICAgfQ0KICAgIH0sDQogICAgIn
RpbWVzdGFtcCI6ICIyMDEzLTA0LTAxVDEyOjAwOjAwWiINCn0.FWuwaPhwUbkk7h9sKW5zSvjsYNugvxJ-
TrVaEgt_DCUT0bmKhQScRrjMB6P9O50uznPwT66oF1NnU_G0HVhRzS5voiXE-y7tT3z0M3-
8A6YK009Bk_digVUul-HA4Fpd5IjoBBGe3yzaQ2ZvzarvRuipvNEQCI0onpfuZZJQ0d8

Signed Statement


```
{
    "version": "1.0.0",
    "id": "33cff416-e331-4c9d-969e-5373a1756120",
    "actor": {
        "mbox": "mailto:example@example.com",
        "name": "Example Learner",
        "objectType": "Agent"
    },
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/experienced",
        "display": {
            "en-US": "experienced"
        }
    },
    "object": {
        "id": "https://www.youtube.com/watch?v=xh4kIiH3Sm8",
        "objectType": "Activity",
        "definition": {
            "name": {
                "en-US": "Tax Tips & Information : How to File a Tax Return "
            },
            "description": {
                "en-US": "Filing a tax return will require filling out either a
1040, 1040A or 1040EZ form"
            }
        }
    },
    "timestamp": "2013-04-01T12:00:00Z",
    "attachments": [
        {
            "usageType": "http://adlnet.gov/expapi/attachments/signature",
            "display": { "en-US": "Signature" },
            "description": { "en-US": "A test signature" },
            "contentType": "application/octet-stream",
            "length": 4235,
            "sha2":
"dc9589e454ff375dd5dfd6f556d2583e231e8cafe55ef40102ddd988b79f86f0"
        }
    ]}
```


Note Attached signature not shown, see attachments for attachment message format.