cornerstone

CSOD xAPI (Tin Can)

Integration Guide

Cornerstone Edge CSX

Table of contents

Table of contents

1. CSOD xAPI (Tin Can) Support

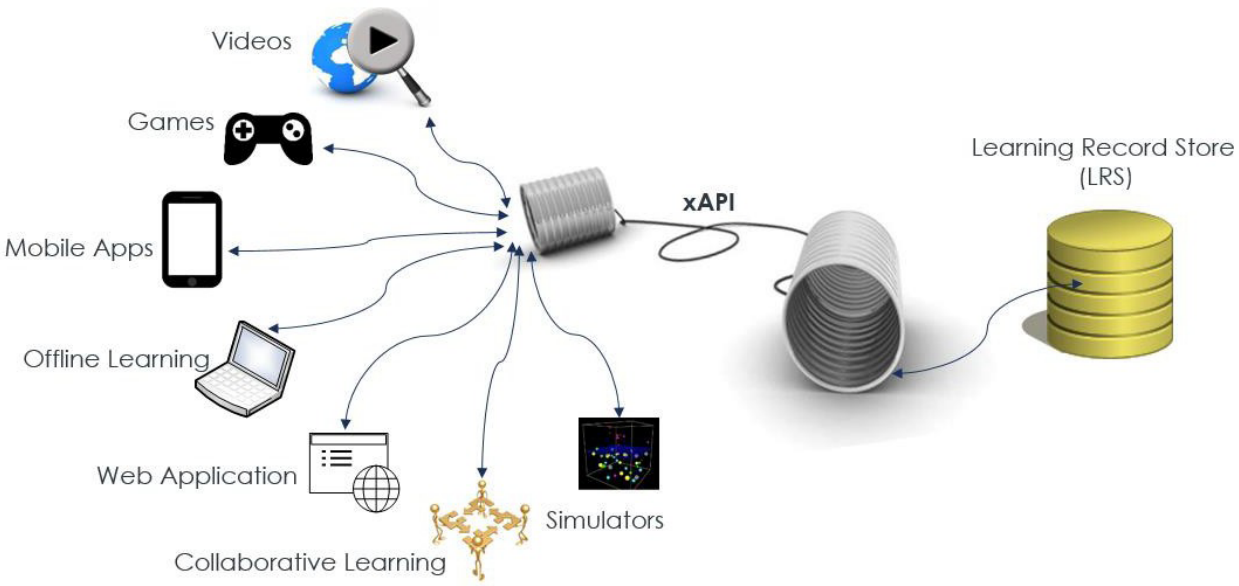
1.1 Introduction

The xAPI (also known as Tin Can API or Experience API) is the next generation e-learning protocol that enables recording of a wide range of learning experience, including native mobile applications, team-based e-learning, and more.

With this enhancement, you will be able to record and export xAPI learning experiences that occur out of the LMS.

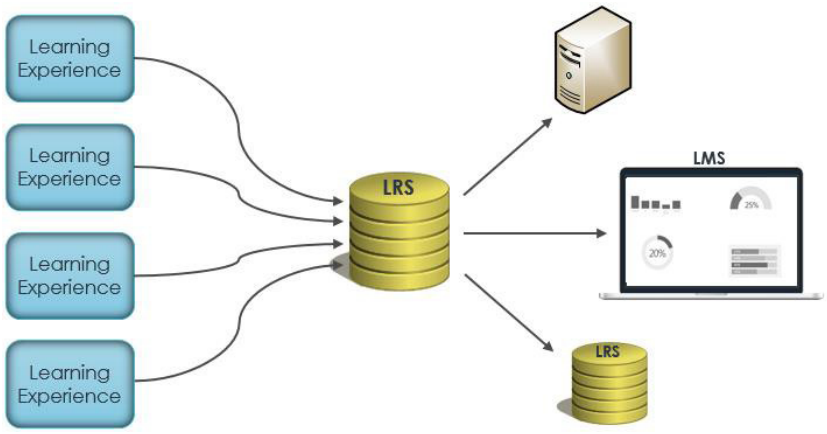
The xAPI protocol does not require the content to be structured in a specific format. It allows with the freedom of defining ANY kind of activity as a Learning Experience that can be captured on a Learning Record Store (LRS).

The xAPI is owned by ADL.



1.2 How does it work?

Learning occurs everywhere and in many different formats such as social learning activities, collaborative activities, and mobile learning activities. All of those activities can be recorded on the xAPI Statement form; a simple construct of “noun, verb, object” or “I did this”. The statements are sent securely to the LRS that records the statements. The recorded data can be shared with the LMS, LRSs, and reporting systems.



1.3 What is a Statement?

- A simple construct that tracks an aspect of a learning experience.
- A set of several Statements may be used to track complete details about a learning experience. For example; tracking a question-level data.
- The statement consists of:

<Actor (learner)> <verb> <object> with <result> in <context>

Examples:

- “Michael completed Compliance Training”
- “Judy performed CPR 9 out of 12 times”
- “Kelly read Informal Learning for Organizations article”
- “Juana added a new post: “The role of talent management in small organization”

2. Implementation

2.1 Feature Enablement

Upon release, this functionality is automatically enabled for all organizations using the Learning module.

A Learning Record Store (LRS) will be available per portal to enable recording of xAPI statements.

2.2 Access Key for an Activity Provider

When integrating xAPI into your content, an access key needs to be provided per Activity Provider, any system or thing that can be outfitted to generate Tin Can Statements based on actions that occur with it.

An Activity Provider (AP) definition by ADL:

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.

It is advised to define a separate AP for each content provider, platform, or application for analysis consumptions.

Note: To get an Access Key for an AP, contact Global Customer Support (GCS).

3. xAPI Integration

There are different approaches for integrating xAPI within content. xAPI Statements can be sent directly from a web or a mobile application or by a browser extension that provides users with an interface to send statements to the LRS.

The browser extension, also called Bookmarklet is a simple tool that helps communicating xAPI statements to the LRS.

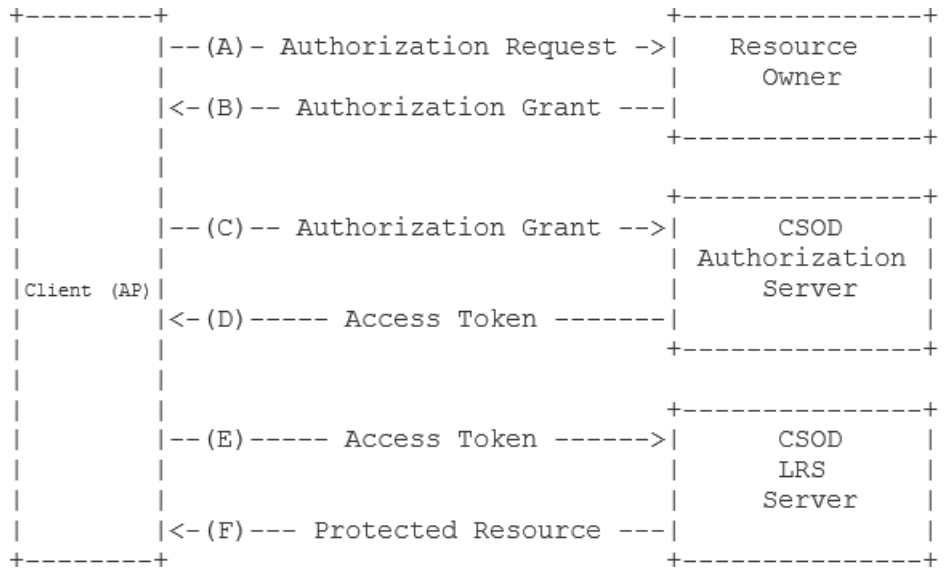
More information as well as code resources can be found at ADLNet github resources.

Regardless the selected approach of integration, The AP represents the system that sends data to the LRS.

4. Authentication Requirements

CSOD authentication method is based on oAuth 2.0 protocol. That means the oAuth 2.0 must be integrated to utilize CSOD LRS.

Protocol Flow



Workflow Description

Step 1: Register the application to establish communication between application and the service.

Step 2: Adhere to one or more of the following Grant Flow. Authorize end point is used to generate appropriate Code/Token:

a. Authorization Grant Flow

b. Implicit Grant Flow

c. Client Credentials Grant Flow

Step 3: Generate Access Token for the client (only needed for Authorization Grant Flow).

Step 4: Using the Access token, create a Bearer token request to access the xAPI.

4.1 Application Registration

Before using the oAuth service, the application must be registered with the service.

To get an application registered, contact Cornerstone through a Global Customer Support (GCS) case.

There are three supported grant flows:

- **Authorization Grant Flow** - Relevant for server-side Applications. Used to obtain both access tokens and refresh tokens and is optimized for confidential clients.
- **Implicit Grant Flow** - Relevant for Mobile Apps or Web Applications. Used to obtain access tokens.
- **Client Credentials** – Enables for a back-end integration between a trusted vendor and Cornerstone. Relevant for server to server communication.

4.2 Required registration information

The following information is required for application registration:

Application Name – Up to 150 characters. Accented characters are not supported.

Redirect URI – A separate redirect URI per application. This is an absolute URL. It is the location where the AP resides and it knows how to extract the returned oAuth token.

Grant Types – Authorization/Implicit/Client Credentials

Scope – Two available scopes for importing/exporting xAPI statements at the user level (choose one or both):

1. Write/mine – can write statements or documents that relate to the logged in user.
2. Read/mine – can view statements or document that relate to the logged in user.

One available scope for exporting xAPI statements at the system level:

1. Read – can view statements or documents for all users in the portal (might be restricted based on the viewing user's permissions).

Note: For Client Credentials grant type, additional settings may be required.

Items received from CSOD:

Client Id: Unique random publicly exposed string that is used by the authorization server to identify the client (e.g., tenant1AP1).

Client Secret: Secret key that is used to authenticate the identity of the client to the authorization server and must be kept confidential (e.g., 90f3fjaFOs23r).

5. oAuth 2.0 Requests

5.1 For Authorization Grant Flow:

URL

```
services/api/oauth2/authorize?client_id=<ClientID>&redirect_uri=<RedirectUri>&sc
ope=<Scope>&response_type=code&state=<State>
```

HTTP Verb = GET

Headers – N/A Body – N/A

<Client_ID>: Valid api_id of the application

<RedirectUri>: One of the registered redirect uri for the application (For Phase only one redirect_URI can be registered per application)

<scope>: One or more of the valid scopes registered for the application. Multiple scopes are space delimited.

<response_type>: code

<State>: Opaque value, optional. check for XSS and reject for illegal values

Response

The system will redirect the user to the Consent screen based on the authentication method

Item	Value
Status	304/ Redirect
Headers	
Body	

as described in the Authentication Possibilities section.

Obtain Access Token

To obtain the access token, the client makes a request to the token endpoint by sending the following parameters using the “application/x-www-form-urlencoded” format with a character encoding of UTF-8 in the HTTP request entity-body.

Request

Item	Value
URL	/services/api/oauth2/token
HTTP Verb	POST
Headers	Content-type=x-www-form-urlencoded
Body	x-www-form-urlencoded: code : <Code> grant_type: authorization_code redirect_uri: <Redirect URI> client_id: <Client Id> client_secret: <Client Secret> code=eWMyODk3NWNhODg1NjR5&grant_type=authorization_code&redir ect_uri=https%3A%2F%2Fwww.google.com&client_id=sadqd2ad&client_s ecret=asdji398hdoiahd8hdsahd3

<code>: Valid authorization code for the application

<RedirectUri> : One of the registered redirect uri for the authorization code generated(For Phase only one redirect_URI can be registered per application)

<grant_type>: authorization_code

<Client Id>: The client Id received when the application was registered.

<Client Secret>: The client secret received when the application was registered.

Response

Item	Value
Status	201 / Created
Headers	content-type=x-www-form-urlencoded

Body	{
	“access_token”:”2YotnFZFEjr1zCsicMWpAA”, “scope”:”read/mine write/mine”, “expires_in”:3200, “refresh_token”:”tGzv3JOkF0XG5Qx2TlKWIA”, “grant_type”:”authorization_code”
	}

<access_token>: Valid access token string
<scope>: scope value(s) specified while Authorizing (space delimited)
<expires_in>: TTL in sec
<refresh_token>: refresh token value defined for the application
<grant_type>: Authorization grant type used while Authorizing the request

Using the Access Token

To use the access token, create an Authorization header with the token that was received in either the grant methods described above.

Example Request:

URL	/services/api/LRS/statements
HTTP Verb	GET
Headers	Authorization: Bearer <Access Token>
Body	N/A

<Access Token>: valid token of the application

5.2 For Implicit Grant Flow:

URL

```
services/api/oauth2/authorize?client_id=<ClientID>&redirect_uri=<RedirectUri>&sc
ope=<Scope>&response_type=token&state=<State>
```

HTTP Verb = GET

Headers – N/A Body – N/A

<Client_ID>: valid api_id of the application

<RedirectUri> : One of the registered redirect uri for the application. (For Phase only one redirect_URI can be registered per application)

<scope>: One of the valid scopes registered for the application

<response_type>:token

<State>:opaque value. optional. check for XSS and reject for illegal values

Response

Item	Value
Status	302/ Redirect
Headers	
Body	

The system will redirect the user to the Consent screen based on the authentication method as described in the Authentication Possibilities section.

Using the Access Token

To use the access token, create an Authorization header with the token that was received in either the grant methods described above.

Example Request:

URL	/services/api/LRS/statements
HTTP Verb	GET
Headers	Authorization: Bearer <Access Token>
Body	N/A

<Access Token>: Valid token of the application

5.3 For Clients Credentials Grant Flow:

The client makes a request to the token endpoint by sending the following parameters using the “application/x-www-form-urlencoded” format with a character encoding of UTF-8 in the HTTP request entity-body.

Request

URL	/services/api/oauth2/token
HTTP Verb	POST
Headers	Content-type=x-www-form-urlencoded Basic Authentication (UserName:ClientId,Password:ClientSecret) will generate Authorization: Basic TsfjMmhoeWRtc2VoMzpSczBzNEhTU0huTTlx a3lUbDZvSVBaY01vL01LaGhwa2VxU2IwaXUvNFkrV011VndCVFpRM2t 1WVJuZWhqcWJhQlJOVVVjRXZnMXR5OWxrMUVCNURudz09
Body	x-www-form-urlencoded: scope:<scope> grant_type: client_credentials user_id: <user_id> scope=read&grant_type=client_credentials&user_id=1

Request

URL	/services/api/oauth2/token
HTTP Verb	POST
Headers	Content-type=x-www-form-urlencoded
Body	x-www-form-urlencoded: scope:<scope> grant_type: client_credentials client_id:<client_id> client_secret:<client_secret> user_id:<user_id> scope=read&grant_type=client_credentials&user_id=1&client_id =TsfjMmhoeWRtc2&client_ secret=WJhQlJOVVVjRXZnMXR5OWxrMUVCNURudz09

<grant_type>: client_credentials

<scope>: requesting scope

<user_id>: Active and NOT locked out for a given portal

Note: client_id and client_secret can be passed in the header as base64 encoded Authorization value or in the POST request body. The header values take precedence over the POST request body.

Response

Status	200 / OK
Headers	Content-Type: application/json;charset=UTF-8 Cache-Control: no-store Pragma: no-cache
Body	{ “access_token”:”2YotnFZFEjr1zCsicMWpAA”, “scope”:”read”, “expires_in”:3200, “token_type”:”bearer”,”refresh_token”:null }

<access_token>: Valid access token string

<scope>: scope value(s) specified while Authorizing (space delimited)

<expires_in>: TTL in sec

<token_type>: Bearer token

<refresh_token>: value is null as refresh token is not generated for this grant type

Refreshing logic:

There is no special protocol to refresh an access token using the client credentials authorization flow. Client application simply requests a new token once the previous token has expired.

Think of Client Credentials as a refresh token that doesn’t expire. Refresh Tokens exist to prevent you from having to reauthorize a user so often. But since there is no third-party in the Client Credentials exchange, there is no reason to have a refresh token.

Validations:

If multiple requests are made using the same valid code, existing access token will be returned. The Expires_in value should be reflected with the new value. (the time difference between now and the expiration time of the token)

If the token is invalid, return a error that the token is expired. (client should use the refresh token to generate a new token)

Admin UI Validations:

1. Redirect URL is not mandatory for Client Credentials.
2. If more than one Grant type is selected, and one of which is Client Credentials; then show the error message: Client credentials cannot be combined with other grant types.

Revoke:

Client ID Revoke: When a Client ID is revoked, we cannot reactivate or use that client ID. The access tokens and refresh tokesn associated with this client ID should be revoked too.

UI: A new link to revoke the Client ID should be added to the Application Details grid on the App registration Page. The revoked Client IDs will not show up in the list of Client IDs displayed in the grid.

Database Changes: Add a new nullable column “revoked_date” to sts_oauth2_clients. If this column has a date that implies the client ID is revoked.

Access/ Refresh Token Revoke: When an access token or refresh token is revoked, if that token is sued in service request error should be displayed.

UI: A textbox should be added to the App registration Page to enter teh token that needs to be revoked. Same text box will be used to enter access and refresh token.

Database Changes: Add a new nullable column “revoked_date” to **dbo.sts_oauth2_tokens** and **dbo.sts_oauth2_refresh_tokens**. If this column has a date that implies the token is revoked.

6. User Authorization – Consent Screen

As part of the oAuth 2.0 protocol, the user is required to authorize access to the application to read or write data to CSOD system.

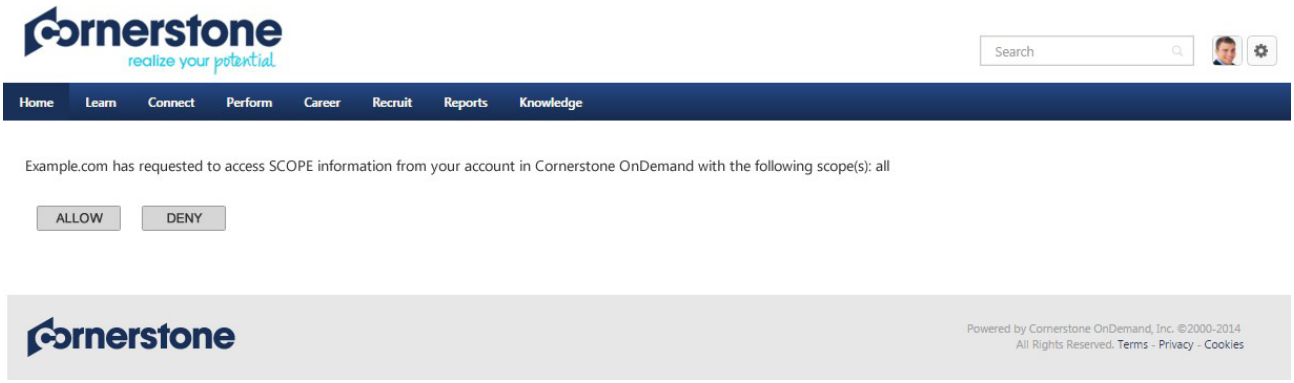
Note: This is relevant only when using Implicit or Authorization grant types. The Consent screen does not appear when using Client Credentials grant type.

When the application accepts the authentication request, the user is redirected to **Consent** page that displays an authorization message that requests the user authorizing the application to send data to the portal. The redirection to the Consent page will be intercepted if the user is not authenticated.

The consent screen is opened on CSOD portal and includes the following message:

<Application Name> has requested to access SCOPE information from your account in Cornerstone OnDemand with the following scope(s): <Scopes>

Allow and **Deny** buttons are available.



6.1 Consent Screen Functionality

Upon clicking Allow, the Consent page is closed and the system allows the application to access the user's xAPI data.

Upon clicking Deny, the Consent page is closed and the system rejects the authentication request.

If the user did not click Allow or Deny within 5 minutes, the session is expired and the user is directed to a timeout expired message.

6.2 Authentication Possibilities

There two possible scenarios:

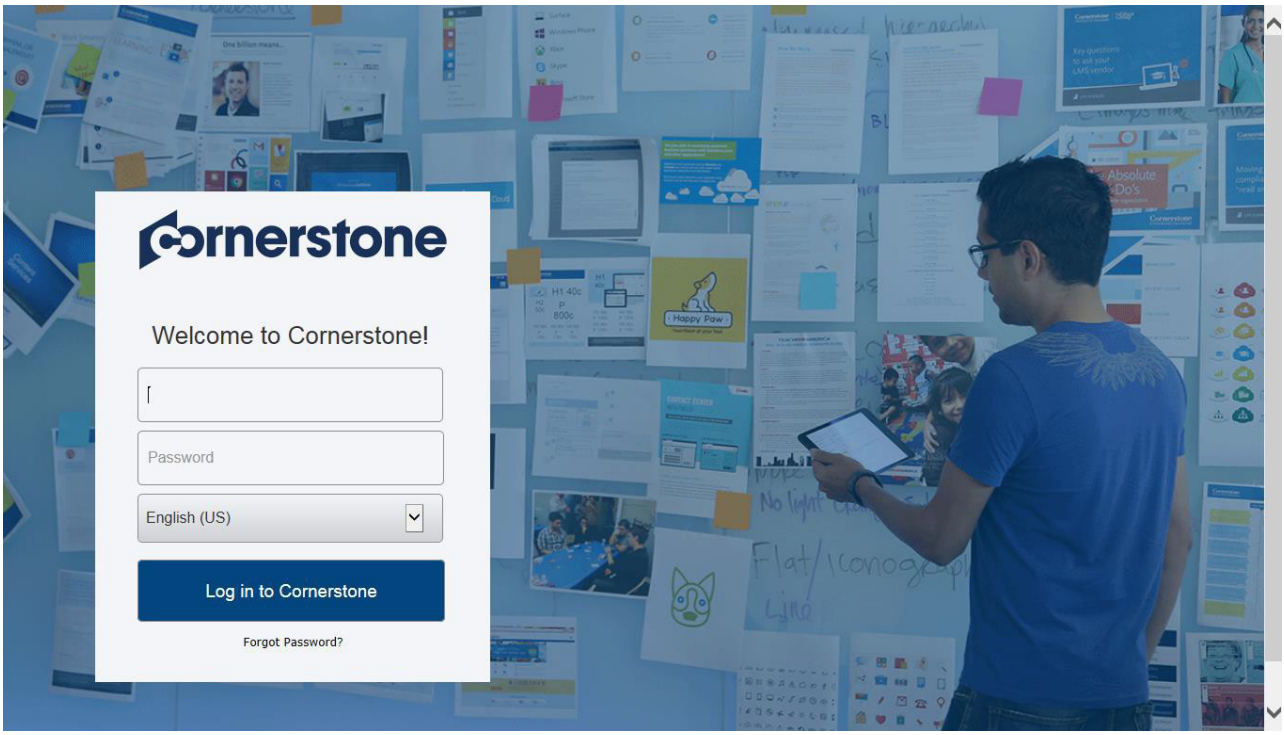
SSO Authentication

If SSO is configured for the portal, the user will authenticate using their portal's SSO configuration and will be redirected to the Consent page.

No SSO Authentication

If no SSO is configured for that portal, the page will be redirected to the CSOD Login page configured for that portal and the user will be required to enter UserName/Password.

Upon successfully log-in the system, the user will be redirected to Consent page.



7. xAPI Statements

xAPI statements can be sent to CSOD LRS. The statement structure and properties must be compliant with xAPI v 1.0.0 provided by ADL.

Statement Supported Properties:

Property	Type	Description
id	UUID	UUID assigned by LRS if not set by the Activity Provider.
actor	Object	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	Only version 1.0.0 is supported. The system does not enforce the required X-Experience-API-Version header, but when in-use, it must be between 1.0.0 and less than 1.1.0.
attachments	Array of attachment Objects	Not supported

7.1 Code Example

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

```
{
“id”: “ab123cd4-e56f-g7h8-i90j-l234m5n67m8m”,
“actor”:{
      “objectType”: “Agent”,
      “mbox”:”mailto:username@csod.com”
},
“verb”:{
      “id”:”http: //http://activitystrea.ms/schema/1.0/watch”,
      “display”:{
      “en-US”:”Watched”
      }
},
“object”:{
      “definition”: {
            “type”: “http://activitystrea.ms/schema/1.0/page”,
            “name”: {
              “en-US”: “Example Video”
            }
          },
      “id”: “https://www.examplevideo.com/”,
      “objectType”: “Activity”
      }
}
```

More examples are available on the Appendix

7.2 Actor Variables

An Agent value must be one of the following types:

mailto IRI - “mailto:agent@example.com” – The email of the user within Cornerstone’s system.

Account - A user account recognized by Cornerstone.

Example:

```
{“account”: { “name”: “<username>”,”homePage”: “<portal url>”}, “objectType”: “Agent” }
       <username> - The user’s username or user-ref in the portal.
       <portal url> - The base url for the portal.
```

For Groups: Anonymous/ Identified

Note: Users must be identifiable on the portal to be valid as Agents.

An Identified Group and an Agent must use different identifiers. An identified group that has the same identifier as an agent will be rejected.

7.3 Supported Verbs

Cornerstone LRS allows sending predefined verbs as described in the table below.

If a statement is sent with a different verb, the system will respond with an HTTP 403 Forbidden response

Label	Description	ID (IRI)
abandoned	The verb "Abandoned" indicates that the AU session was abnormally terminated by a learner's action (or due to a system failure).	https://w3id.org/xapi/adl/verbs/abandoned
answered	Indicates the actor replied to a question, where the object is generally an activity representing the question. The text of the answer will often be included in the response inside result.	http://adlnet.gov/expapi/verbs/answered
attempted	Indicates the actor made an effort to access the object. An attempt statement without additional activities could be considered incomplete in some cases.	http://adlnet.gov/expapi/verbs/attempted
attended	Indicates the actor was present at a virtual or physical event or activity.	http://adlnet.gov/expapi/verbs/attended
commented	Indicates the actor provided digital or written annotations on or about an object.	http://adlnet.gov/expapi/verbs/commented
completed	Indicates the actor finished or concluded the activity normally.	http://adlnet.gov/expapi/verbs/completed
exited	Indicates the actor intentionally departed from the activity or object.	http://adlnet.gov/expapi/verbs/exited
experienced	Indicates the actor only encountered the object, and is applicable in situations where a specific achievement or completion is not required.	http://adlnet.gov/expapi/verbs/experienced
failed	Indicates the actor did not successfully pass an activity to a level of predetermined satisfaction.	http://adlnet.gov/expapi/verbs/failed
initialized	Indicates the activity provider has determined that the actor successfully started an activity.	http://adlnet.gov/expapi/verbs/initialized
interacted	Indicates the actor engaged with a physical or virtual object.	http://adlnet.gov/expapi/verbs/interacted

launched	Indicates the actor attempted to start an activity.	http://adlnet.gov/expapi/verbs/launched
liked	Indicates that the actor marked the object as an item of special interest. The “like” verb is considered to be an alias of “favorite”. The two verbs are semantically identical.	http://activitystrea.ms/schema/1.0/like
mastered	Indicates the highest level of comprehension or competence the actor performed in an activity.	http://adlnet.gov/expapi/verbs/mastered
passed	Indicates the actor successfully passed an activity to a level of predetermined satisfaction.	http://adlnet.gov/expapi/verbs/passed
preferred	Indicates the selected choices, favored options or settings of an actor in relation to an object or activity.	http://adlnet.gov/expapi/verbs/preferred
progressed	Indicates a value of how much of an actor has advanced or moved through an activity.	http://adlnet.gov/expapi/verbs/progressed
read	Indicates that the actor read the object. This is typically only applicable for objects representing printed or written content, such as a book, a message or a comment. The “read” verb is a more specific form of the “consume”, “experience” and “play” verbs.	http://activitystrea.ms/schema/1.0/read
registered	Indicates the actor is officially enrolled or inducted in an activity.	http://adlnet.gov/expapi/verbs/registered
responded	Indicates an actor reacted or replied to an object.	http://adlnet.gov/expapi/verbs/responded
resumed	Indicates the application has determined that the actor continued or reopened a suspended attempt on an activity.	http://adlnet.gov/expapi/verbs/resumed
scored	Indicates a numerical value related to an actor’s performance on an activity.	http://adlnet.gov/expapi/verbs/scored
shared	Indicates the actor’s intent to openly provide access to an object of common interest to other actors or groups.	http://adlnet.gov/expapi/verbs/shared
suspended	Indicates the status of a temporarily halted activity when an actor’s intent is returning to the object or activity at a later time.	http://adlnet.gov/expapi/verbs/suspended

terminated	Indicates that the actor successfully ended an activity.	http://adlnet.gov/expapi/verbs/terminated
viewed	Indicates that the actor has viewed the object.	http://id.tincanapi.com/verb/viewed
voided	A special reserved verb used by a LRS or application to mark a statement as invalid. See the xAPI specification for details on Voided statements.	http://adlnet.gov/expapi/verbs/voided
watched	Indicates that the actor has watched the object. This verb is typically applicable only when the object represents dynamic, visible content such as a movie, a television show or a public performance. This verb is a more specific form of the verbs “experience”, “play” and “consume”.	http://activitystrea.ms/schema/1.0/watch

7.4 Object, Results, and Context Properties

	Property	Description
Object Properties	ID	An identifier for a single unique Activity. Required.
	Definition	Optional Metadata
	ObjectType	Optional. All types are supported. If not specified, the objectType is assumed to be “Activity”.
	Activity definition	Name DescriptionType MoreInfo Interaction properties interactionType correctResponsesPattern Extensions
Result Properties	Score	An optional numeric field. Properties: scaled, raw, min, max. Note: There is no validation if the provided raw score is in between min and max.
	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

	Property	Description
Context Properties	Registration	– UUID. The registration that the Statement is associated with.
	Instructor	Agent/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	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 (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. Another Statement which should be considered as context for this Statement. Note: The system does not validate that the referenced statement exists in the LRS.
	Extensions	Object. A map of any other domain-specific context relevant to this Statement.

7.5 xAPI course transcript updates via xAPI statement

xAPI statements can make changes to the user's transcript if they meet the criteria shown in the table below, and as long as the course has not been in a "Completed" status before the xAPI statement was sent.

	Values	User's Transcript update
Statement Properties	verb – Passed/Failed/Completed/ Attempted **AND** result.completion: true	The course will be marked as 'Completed' and move to the completed tab. Progress bar will be set to 100% in the training details page. Completion date will be based on the statement's timestamp.
	verb – Passed/Failed/Completed/ Attempted **AND** result.completion: true **AND** result.score.scaled /result.score. raw+result.score.max has a value	The course will be marked as 'Completed' and move to the completed tab. Progress bar will be set to 100% in the training details page. Completion date will be based on the statement's timestamp. In addition, the score will be updated in the training details page based on the score sent.
	verb – Terminated **AND** result.duration has a value	View Time in training details page will be updated based on the result duration value.

8. Export xAPI statements via API

xAPI statements can be exported from CSOD's LRS using API.
For example, you could export the statements to a third party LRS.

8.1 Export xAPI statements call structure:

URL

https://<portal_name>.csod.com/services/api/LRS/statements?format=exact&verb=https%3A%2F%2Fw3id.org%2Fxapi%2Fadl%2Fverbs%2Fabandoned&agent=1&related_agents=False&activity=1&related_activities=False&registration=1&statementId=1&voidedStatementId=1&since=06%2F08%2F2022%205%3A58%20PM&until=06%2F29%2F2022%205%3A58%20PM&limit=1000

HTTP Verb = GET

Headers - X-Experience-API-Version

Body – N/A

Parameters:

<verb>: Filter statements according to verb URI

<agent>: The learner who performs the action

<related_agents>: True or False

<activity>: Filter statements according to a specific Activity ID

<related_activities>: True or False

<registration>: Filter statements according to a specific Registration ID
<statementId>: Filter statements according to a specific Statement ID
<voidedStatementId>: Filter statements according to a specific Voided Statement ID
<since>: Filter results from a specific date/time

<until>: Filter results until a specific date/time

<limit>: Limit the number of statements per one page

<last_statement_id>: When using pagination, provide the last statement ID from the last call

Response

Item	Value
Status	200/ OK
Headers	Standard HTTP headers
Body	List of statements

8.2 Performing a system-wide statement export:

First, create a call with the highest limit allowed by the system.

If, for example, the portal name is “exportexample” and the highest limit is 32,000 statements, the call should be:

https://exportexample.csod.com/services/api/LRS/statements?format=exact&limit=32000

Assuming that not all statements were exported with the first call, following calls should use the <last_statement_id> parameter to export the rest of the statements.

If, for example, the ID of the last statement in a call was “33913d4a-d6c3-4c6f-804b-4949c6fd32f6”, the next call should be:

https://exportexample.csod.com/services/api/LRS/statements?format=exact&lastStatementId=33913d4a-d6c3-4c6f-804b-4949c6fd32f6&limit=32000

9. Document API

The Document API is a set of API resources that enable storage and retrieval of data. This is also a way for passing data between a Training Delivery System such as an LMS and an Activity Provider.

The data stored in the Document APIs represents the current situation at a given point in time similar to SuspendData on SCORM. For example, this functionality is recommended for bookmarking data.

There is no limitation on the number “Documents” that can be stored as Document API.

Using a POST method instead of a PUT allows you to update individual properties of an object contained in a JSON document.

Note: The LRS only supports documents in the JSON format.

9.1 State API

State API is a scratch area for Activity Providers. Documents can be stored on two different levels:

Per user, per activity basis. For example: storing bookmarking data.

Per registration, per user, per activity basis. For example: storing data specific to a particular launch or attempt

It is the Activity Provider’s responsibly to clean up the data that is no longer required. PUT, POST, GET, and DELETE are supported for Single Document.

POST, GET, and DELETE are supported for Multiple Document.

More information on the supported methods is available on the Appendix.

9.2 The Activity Profile API

The Activity Profile API is similar to the State API, allowing for arbitrary key / document pairs to be saved which are related to an Activity. The Activity Profile API is used to store activity wide documents that aren't specific to an individual learner. This is used in any scenario where interaction between learners is required. For example collaboration activities, social interaction, or competition.

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

More information on the supported methods is available on the Appendix

9.3 The Agent Profile API

The Agent Profile API is similar to the State API, allowing for arbitrary key / document pairs to be saved which are related to an Agent. For example: user personal information and user settings. Documents can be stored and retrieved for Agents across Activities.

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.

More information on the supported methods is available on the Appendix.

9.4 Voided Statements

The Voided Statements functionality enables for making existing statements invalid. Since statements are immutable the only way to fix a statement is by voiding it and creating the new proper statement. Additionally a voided statement itself can't be voided.

When a statement is voided the flags it as a voided statement that cannot be retrieved in a GET call.

Note: The system does not validate that the statementRef references is a real statement in the system, a voiding statement could reference a non-existence statement in which change it will not change anything.

10. Considerations

This is an initial release of xAPI integration and the functionality is provided through the back-end. There is no integration with the LMS's user interface or with reporting.

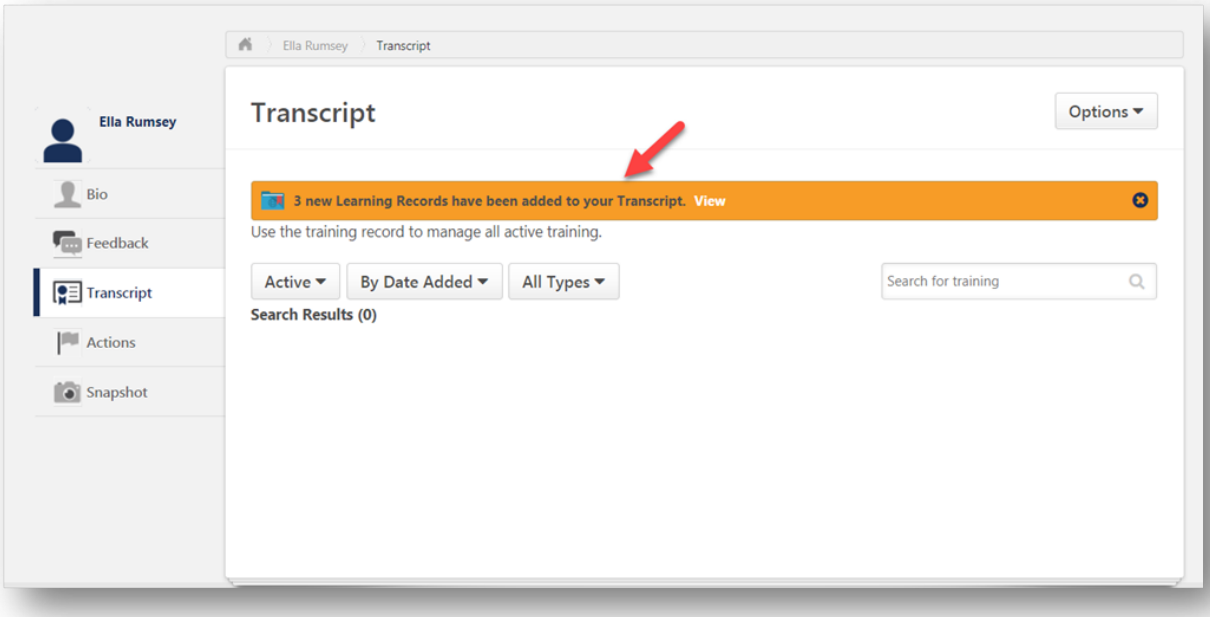
Reports are available through the back-end.

This version is mainly suited for early adopter clients who want to use CSOD LRS.

11. Transcript Integration

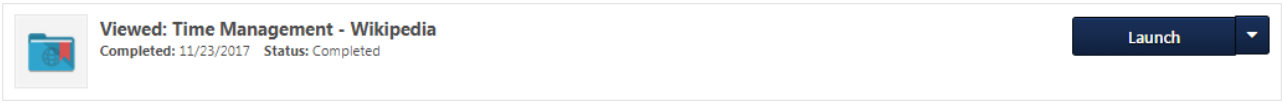
Statements of completed activities may appear on the User Transcript as Completed records.

Permission	Description
xAPI - View Learning Records on Transcript	Allows users to view xAPI Learning Records on the Transcript.



11.1 Transcript View

Completed xAPI Learning Records may be visible under the Completed filter on the Transcript. Users can launch completed xAPI records when a proper URL was provided as the object ID.



The information available for xAPI Learning Record includes the following items:

- Title - This field displays the xAPI Verb and the name of the xAPI learning record.
- Completed - This field displays the date the xAPI learning record completed as defined on the statement.
- Status - This field displays the status of the xAPI learning record. Note: The status for an xAPI learning record will always be set to Complete, because only completed learning records display on the transcript.

The following verbs are supported:

- Attended
- Completed
- Experienced
- Mastered
- Passed
- Read
- Watched
- Viewed

Please note that while the verb ID must be one of the mentioned above, the verb display, which will appear in the user's transcript, is not enforced by the system and is expected to comply with the Verb Display Learning Record Provider Requirements: "The "display" property MUST be used to illustrate the meaning which is already determined by the Verb IRI".

To display a completed activity on Transcript, the xAPI statement must include the following convention under the "contextActivity" attribute:

```
},
 "context":{
    "contextActivities":{
       "category":[
          {
             "id":"http://www.cornerstoneondemand.com/xapi/lms/transcript_display"
          }
       ]
    }
 }
```

12. Appendix

12.1 TinCan Restful API

URL	Method	Url Params	Body Params	Successful Result
Statement API				
/services/api/LRS/statements	GET	statementId voidedStatementId agent verb activity registration related_activities related_agents since until limit format attachments ascending		
Note: format set to “canonical” is not supported. Filter conditions for StatementRefs is not supported.				
/services/api/LRS/statements	PUT			
/services/api/LRS/statements	POST			
Document APIs **State API**				
/services/api/LRS/activities/state	GET	activityId IRI The Activity id associated with this state	None	200 OK Returns the State JSON document
/services/api/LRS/activities/state	POST	agent JSON object The Agent associated with this state stateId GUID	The JSON document to be stored	204 No Content State JSON document is stored

/services/api/LRS/activities/state	PUT	The id for this state, within the given context [registration] GUID (Optional) The registration id associated with this state	The JSON document to be updated	204 No Content State JSON document is updated
/services/api/LRS/activities/state	DELETE		None	204 No Content State JSON document is deleted
/services/api/LRS/activities/state	GET multiple	activityId IRI The Activity id associated with this state agent JSON object The Agent associated with this state [registration] GUID (Optional) The registration id associated with this state [since] Timestamp (Optional) Only ids of states stored since the specified timestamp (exclusive) are returned	None	200 OK Returns array of State Ids
/services/api/LRS/activities/state	DELETE multiple		None	204 No Content State document are deleted
Activity Profile API				
/services/api/LRS/activities	GET	activityId IRI The id associated with the Activities to load	None	200 OK Full Activity JSON

/services/api/LRS/activities/profile	GET	activityId IRI The Activity id associated with this profile	None	200 OK Returns the Activity Profile JSON document
/services/api/LRS/activities/profile	POST	profileId GUID The profile id associated with this	The JSON document to be	204 No Content Activity Profile
		profile	stored	JSON document is stored
/services/api/LRS/activities/profile	PUT		The JSON document to be updated	204 No Content Activity Profile JSON document is updated
/services/api/LRS/activities/profile	DELETE		None	204 No Content Activity Profile JSON document is deleted
/services/api/LRS/activities/profile	GET multiple	activityId IRI The Activity id associated with this profile [since] Timestamp (Optional) Only ids of profiles stored since the specified timestamp (exclusive) are returned	None	200 OK Returns array of Activity Profile Ids
Agent Profile API				
/services/api/LRS/agents	GET	agent JSON object The Agent representation to use in fetching expanded Agent information	None	200 OK Person JSON object

/services/api/LRS/agents/ current	GET	None	None	200 OK Current Logged in Agent JSON object
/services/api/LRS/agents/ profile	GET	agent JSON object The Agent associated with this profile	None	200 OK Returns the Agent Profile JSON document
/services/api/LRS/agents/ profile	POST	profileId GUID The profile id associated with this profile	The JSON document to be stored	204 No Content Agent Profile JSON document is stored
/services/api/LRS/agents/ profile	PUT		The JSON document to be updated	204 No Content Agent Profile JSON document is updated
/services/api/LRS/agents/ profile	DELETE		None	204 No Content Agent Profile JSON document is deleted
/services/api/LRS/agents/ profile	GET multiple	agent JSON object The Agent associated with this profile [since] Timestamp (Optional) Only ids of profiles stored since the specified timestamp (exclusive) are returned	None	200 OK Returns array of Agent Profile Ids

12.2 Statement Example


```
{
	“version”:”1.0.0”,
	“id”:”ab123cd4-e56f-g7h8-i90j-l234m5n67m8m”, “actor”:{
		“objectType”:”Agent”,
		“name”:”FirstName LastName”,
		“mbox”:”mailto:username@csod.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://exampleactivity.com/xapicourse”
		}
	},
	“result”:{
		“completion”:true
	},
	“context”:{
		“instructor”:{
			“objectType”:”Agent”,
			“mbox”:”mailto:username@csod.com”
	},
	“contextActivities”:{
		“parent”:[
			{
				“objectType”:”Activity”,
				“id”:”https://example.com/activity1”,
				“definition”:{
					“name”:{
						“en-US”:”Another Activity”
						}
					}
				}
			]
		}
	},
	“timestamp”:”2012-06-01T19:09:13.245Z”,
```

```
        “stored”:”2012-06-29T15:41:39.165Z”,
        “authority”: {
                “objectType”: “Group”,
                “member”: [
                {
                        “objectType”: “Agent”,
                        “account”: {
                                “homePage”: “http://portal.csod.com/”,
                                “name”: “username_in_csod”
                        }
                },
                {
                        “objectType”: “Agent”,
                        “account”: {
                                “homePage”: “http://portal.csod.com/services/api/oauth2/token”,
                                “name”: “1bup6dvajnaze”
                                }
                        }
                ]
        }
}
```

12.3 Creating an xAPI Activity Provider

Note: The information below assumes an understanding of the JavaScript language and the xAPI specification.

Getting started

There are two open source libraries that can help to create a JavaScript xApi Activity Provider:

1. JSO - OAuth 2.0 Client with JavaScript - Used to create the oauth 2.0 client to authenticate with the LRS.
2. xAPIWrapper js - A JavaScript wrapper to simplify communication to an LRS. Download those libraries.

Referencing Libraries

Once you have the libraries locally, add them to your web application:

```
<script type="text/javascript" src="./jso.js"></script>
<script type="text/javascript" src="./xapiwrapper.min.js"></script>
```

Obtain OAuth2 Token

After registering your Activity provider in cornerstone, use the clientId to obtain a jso object:

```
var jso = new JSO({
    "providerID": "{{name}}",
    "client_id": "{{clientId}}",
    "redirect_uri": "{{redirectUri}}",
    "authorization": {{portalUrl}} + '/services/api/oauth2/authorize',
    "scopes": { "request": ["write/mine", "read/mine"]},
    "debug": true
 });
```

Explanation for the JSO options:

- providerID: OPTIONAL This is just a name tag that is used to prefix data stored in the browser. It can be anything you'd like :)
- client_id: The client identifier of your client that is trusted by the LRS. This id, you receive from cornerstone when an activity provider is registered. As JSO uses the implicit grant flow, there is no need for the client secret.
- redirect_uri: The URI that the user will be redirected back to when completed. This should be the same URL that this page is presented on. This URI, is the same as the one registered for this activity provider.
- scopes.request: Control what scopes are requested in the authorization request. The valid scopes are `write/mine` and `read/mine`.
- debug: If debug is set to true, verbose logging will make it easier to debug problems with JSO.

Getting the token

To get a token, use the getToken function:

```
jso.getToken(function(token) { console.
        log(“The token is: “, token);
        var bearerToken = token.token_type + “ “ + token.access_token.split(“
“).join(“+”); //fix JSO bug that replaces ‘+’ in token with ‘ ‘.
}, opts);
```

You may also ensure that a token is available early in your application, to force all user interaction and redirection to happen before your application is fully loaded. To do that make a call to getToken, and wait for the callback before you continue.

Initialize XAPIWrapper

Create your own configuration object and pass it to the xapiwrapper object

```
var conf = {
  “endpoint” : {{portalUrl}} + “/services/api/LRS/”,
  “auth” : bearerToken,
};

ADL.XAPIWrapper.changeConfig(conf);
```

Explanation for the XAPIWrapper configuration options:

endpoint: The location of the LRS services.

auth: Should be set to the bearer Token received in the Jso.GetToken function.

Send a Sample Statement

Create a statement object:

```
var stmt =
    {
       “actor”:{
          “mbox”:”mailto:user@example.com”
       },
       “verb”:{
          “id”:”http://adlnet.gov/expapi/verbs/answered”,
          “display”:{
             “en-US”:”answered”
          }
       },
       “object”:{ “id”:”http://adlnet.gov/expapi/activities/
          question”
       }
    }
```

Send the statement to the LRS:

```
ADL.XAPIWrapper.sendStatement(stmt, function(resp, obj){
    ADL.XAPIWrapper.log(“[“ + obj.id + “]: “ + resp.status + “ - “ + resp.statusText);
});
```

Documentation Links

- JSO library getting started guide.
- xAPIWrapper library documentation.
- xAPIWrapper library example setup.

12.4 Error Codes

- ActivityNotFound = “Activity with activityId:’{0}’ Not Found”;
- ActivityProfileNotFound = “Activity Profile with activityId:’{0}’ and profileId:’{1}’ Not Found”;
- AgentProfileNotFound = “Agent Profile with Agent:’{0}’ and profileId:’{1}’ Not Found”;
- ConcurrencyConflict = “Check the current state of the resource and set the ‘If-Match’ header with the following ETag to resolve the conflict: ‘{0}’”;
- DeleteMultipleStatesFailed = “Unable to delete all State documents with Agent:’{0}’, Activity:’{1}’, registration:’{2}’ and since:’{3}’ parameters”;
- DocumentJsonParseError = “Document cannot be parsed as a JSON Object”;
- DuplicateStatementErrorMessage = “The following statement already exists: {0}”;
- InvalidAgentJsonObject = “Invalid agent Json object:{0}”;
- InvalidIRI = “’{0}’ must be a valid Internationalized Resource Identifier (IRI). Current value:{1}”;
- InvalidNonEmptyString = “’{0}’ must be a non empty string”;
- InvalidStatementJsonAdditionalProperties = “The property ‘{0}’ at ‘{1}’ is invalid”;
- InvalidStatementJsonGenericErrorWithPath = “An invalid snytax occurred at ‘{0}’”;
- InvalidStatementJsonGenericError = “The json is invalid.”;
- InvalidStatementJsonPattern = “’{0}’ must be a valid {1}. Current value:{2}”;
- InvalidStatementJsonRequiredProperty = “The required property ‘{0}’, is missing from ‘{1}’”;
- InvalidTimestampValue = “’{0}’ timestamp parameter must be formatted according to ISO 8601. Current value:{1}”;
- JsonContentType = “application/json; charset=utf-8”;
- LoggedInAgentNotInStatement = “The current user does not have permission to send a statement concerning other users.”;
- StatementJsonParseError = “Invalid JSON. {0}”;
- StatementNotAllowedToBeVoided = “You are not authorized to void the following statement: {0}”;
- StateNotFound = “State with Agent:’{0}’, Activity:’{1}’ and stateId:’{2}’ Not Found”;
- StatementNotFoundErrorMessage = “The requested statement was not found: {0}”;
- UnknownAgentInStatement = “The following user is unknown: {0}”;
- UserCannotAccessResource = “The current user has {0} permission and does not have access to this specific resource”;
- UserNotInRole = “The current user does not have the following permission: {0}”;
- XApiIntegrationDisabled = “Access Denied. xApi is not enabled”;
- XExperienceAPIVersionErrorMessage = “An X-Experience-API-Version header with a value between 1.0.0 and 1.1.0 is missing from the request”;